



---

# GAP SDK

**GreenWaves Technologies**

**Release-v4.28.1**

**Oct 20, 2022**



## CONTENTS

<b>1</b>	<b>Getting Started</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>7</b>
<b>3</b>	<b>Libraries</b>	<b>9</b>
<b>4</b>	<b>User and Developer Guides</b>	<b>11</b>
<b>5</b>	<b>Examples</b>	<b>43</b>
<b>6</b>	<b>Tools</b>	<b>215</b>
<b>7</b>	<b>API Reference</b>	<b>437</b>
	<b>Python Module Index</b>	<b>717</b>
	<b>Index</b>	<b>719</b>



## GETTING STARTED

Follow this guide to:

- Setup a command line development environment on Ubuntu
- Get the source code
- Build and run a helloworld

### 1.1 Install dependencies

You'll need to install some host dependencies using your package manager:

```
# Update your system
sudo apt update
sudo apt upgrade

# Install dependencies
sudo apt-get install -y \
    autoconf \
    automake \
    bison \
    build-essential \
    cmake \
    curl \
    doxygen \
    flex \
    git \
    graphicsmagick-libmagick-dev-compat \
    gtkwave \
    libftdi-dev \
    libftdi1 \
    libjpeg-dev \
    libqt5charts5-dev \
    libsdl2-dev \
    libsdl2-ttf-dev \
    libsndfile1-dev \
    libtool \
    libusb-1.0-0-dev \
    ninja-build \
    pkg-config \
```

(continues on next page)

(continued from previous page)

```
python3-pip \
qtbase5-dev \
rsync \
scons \
texinfo \
wget
```

## 1.2 Python Package Management

SDK and some tools are all based on Python3 (version > 3.8), you can use following command to set your default python to python3.

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 10
```

This will setup a “python” binary pointing at python3.

We strongly recommend to use [Anaconda3](<https://www.anaconda.com/>) to manage python packages

## 1.3 Download and install the toolchain

Now clone the GAP/RISC-V toolchain:

```
git clone https://github.com/GreenWaves-Techologies/gap_riscv_toolchain_ubuntu.git
cd gap_riscv_toolchain_ubuntu
# Depending on where you want to install it, you may need to run this command with sudo
./install.sh
```

## 1.4 Get the GAP SDK

Clone the actual sdk repository

- gitlab: you can find the url on the top, blue button which written “Clone”
- github: you can find the url on the top, green button which written “Code”

```
git clone <the url of this repository>
```

If you are using ssh to clone, don’t forget to put your ssh-key in your account.

## 1.5 Configure the SDK

You can either source sourceme.sh in the root sdk folder and then select the right board from the list, or directly source the board config.

```
source sourceme.sh

or

source config/<the target you want to use>.sh
```

For GAP8, if you directly source the board config, you need to source the appropriate config file for the board that you have. The SDK supports 3 boards (gapuino, gapoc\_a and gapoc\_b) and each of them can use version 1/2/3 of the GAP8 chip. Boards bought before 10/2019 contains GAP8 version 1 and use a USB B plug for JTAG while the ones bought after contains version 2/3 and use a USB micro B for JTAG.

Once the proper config file is sourced, you can proceed with the SDK build.

Note that after the SDK has been built, you can source another board config file to change the board configuration, in case you want to use a different board. In this case the SDK will have to be built again. As soon as the SDK has been built once for a board configuration, it does not need to be built again for this configuration, unless the SDK is cleaned.

## 1.6 Python requirements

Our modules (gapy runner) require a few additional Python packages that you can install with this command from GAP SDK root folder:

```
pip3 install -r requirements.txt
pip3 install -r doc/requirements.txt
```

## 1.7 SDK installation

First, use the following command to configure the shell environment correctly for the GAP SDK. It must be done for each terminal session:

```
cd path/to/gap_sdk
```

Choose which board

```
source sourceme.sh
```

Tip: You can add an “alias” command as follows in your .bashrc file:

```
alias GAP_SDK='cd path/to/gap_sdk && source sourceme.sh'
```

Typing *GAP\_SDK* will now change to the *gap\_sdk* directory and execute the *source* command.

Once in the SDK, run *make help* to get commands and get SDK ready to use.

```
$ make help
=====
 GAP SDK =====
```

(continues on next page)

(continued from previous page)

```
Main targets:  
- clean      : clean the SDK  
- all        : build the whole SDK with all tools  
- minimal    : get latest sources for all rtos and libs  
- gvsoc      : build GVSOC simulation platform  
- openocd.all : build OpenOCD tools to run simulation on boards  
- nntool     : build nntool
```

Then, depends on what you need, build the SDK accordingly

```
make <target>
```

## 1.8 Install OpenOCD Rules

- Copy openocd udev rules and reload udev rules

```
sudo cp <your openocd path>/openocd/contrib/60-openocd.rules /etc/udev/rules.d  
sudo udevadm control --reload-rules && sudo udevadm trigger
```

- Now, add your user to dialout group.

```
sudo usermod -a -G dialout <username>
```

This will require a logout / login to take effect

## 1.9 Build SDK Doc

SDK Doc is build and generated based on SPHINX [<https://www.sphinx-doc.org/en/master/>], the SDK will have installed all the necessary packages for you. You just need to run:

```
cd doc  
make html
```

This will generate the doc in HTML in

```
doc/_build/html/
```

and open the file index.html with your browser

**Within this doc, you can find all the api descriptions, how to use our tools (like nntool, gvsoc, profiler, etc), and some useful application notes.**

## 1.10 Build and run a HelloWorld

Finally try a test project. First connect your GAPuino to your PCs USB port. Now, you should be able to run your first helloworld on the board.

```
cd examples/<target name>/basic/helloworld
make clean all run PMSIS_OS=freertos platform=board
```

In details, *PMSIS\_OS*, *platform*, *io* are used to configure the RTOS to run the example on, specify the runner, and select the output for printf.

- *PMSIS\_OS* : RTOS (freertos/pulpוס)
- *platform* : board gvsoc rtl fpga (defult if not set is gvsoc)
- *io* : disable host(semihosting) uart rtl (defult if not set is semihosting)

After the build you should see an output resembling:

```
*** PMSIS HelloWorld ***

Entering main controller
[32 0] Hello World!
Cluster master core entry
[0 7] Hello World!
[0 0] Hello World!
[0 4] Hello World!
[0 5] Hello World!
[0 3] Hello World!
[0 1] Hello World!
[0 2] Hello World!
[0 6] Hello World!
Cluster master core exit
Test success !
Detected end of application, exiting with status: 0
Loop exited
commands completed
```

If this fails, ensure that you followed previous steps correctly (openocd install, udev rules). If libusb fails with a permission error, you might need to reboot to apply all changes.

If you need GAP tools for neural networks (nntool) or the Autotiler, please follow the next section

## 1.11 Console IO via uart

If you choose to boot your application from Flash, and/or you want to view the output of printf's in your code then you can first compile your application with the printf redirected on the UART with this command:

```
make clean all platform=board PMSIS_OS=your_os io=uart
```

You can also use a terminal program, like “cutecom”:

```
sudo apt-get install -y cutecom
cutecom&
```

Then please configure your terminal program to use /dev/ttyUSB1 with a 115200 baud rate, 8 data bits and 1 stop bit.

## 1.12 Upgrading/Downgrading the SDK

If you want to upgrade/downgrade your SDK to a new/old version:

```
cd gap_sdk
git checkout master && git pull
git checkout <release tag name>
# For minimal install
make clean minimal_sdk
# for full install
make clean sdk
```

You can find a list of releases in this repository.

## 1.13 Next Steps

Here are some next steps for exploring the GAP SDK:

- Try other *Examples*
- Discover *Tools* such as the Autotiler and NNTool
- Check out *User and Developer Guides* for other additional information

## INTRODUCTION

The GAP SDK allows you to compile and execute applications on GAP processors.

It provides:

- An extensive set of tools including:
  - *NNTool*: a set of tools to easily port NN graphs from various training packages to GAP chips.
  - *Autotiler*: A code generator which can generate a user algorithm (CNN, MatrixAdd, MatrixMult, FFT, MFCC, etc) with optimized memory management.
  - *GVSOC*: a lightweight and flexible instruction set simulator which can simulate GAP processors, allowing execution of programs on a virtual platform without any hardware limit.
- Two different operating systems: FreeRTOS and PulpOS
- a common API to create applications that can run on both OSes and interacts easily with other devices
- *Examples* to help you create your own application.

### 2.1 GAP Chips

To learn more about GAP chips check these pages:

TODO

### 2.2 Getting Started

Welcome to GAP SDK! See the documentation's *Getting Started* to start developing and *User and Developer Guides* for more advanced subjects.

### 2.3 Resources

- **Website:** <https://greenwaves-technologies.com>
- **Source code:** [https://github.com/GreenWaves-Technologies/gap\\_sdk](https://github.com/GreenWaves-Technologies/gap_sdk)
- **Releases:** [https://github.com/GreenWaves-Technologies/gap\\_sdk/releases](https://github.com/GreenWaves-Technologies/gap_sdk/releases)
- **Issues:** [https://github.com/GreenWaves-Technologies/gap\\_sdk/issues](https://github.com/GreenWaves-Technologies/gap_sdk/issues)



## LIBRARIES

### 3.1 GAP Lib

TODO

### 3.2 OpenMP

This repository contains the source code for GAPs OpenMP API.

It only supports GCC.

#### 3.2.1 Supported features

GAP OpenMP only support static scheduling and the following directives (pragma):

- `parallel`
- `critical`
- `for`
- `barrier`
- `atomic`
- `single`

#### 3.2.2 How to use

To use OpenMP in your application, simply add `CONFIG_OPENMP=1` to your Makefile.



## USER AND DEVELOPER GUIDES

### 4.1 General

#### 4.1.1 Asking For Help

You can ask for help on Github using issues.

Include any useful information such as:

- Information about your environment including but not limited to
  - SDK version (commit hash, branch or release number)
  - What board you are using
- What you want to do
- What is actually happening
- Steps to reproduce the issue

#### 4.1.2 Application Makefile options

##### Generic options

To compile and run your application, use `make clean all run`. You can also add options to this command. Here is a description of these options:

- `platform=<option>` is used to change the platform running the application. `<option>` includes:
  - `gvsoc` to run your application on GVSoC
  - `board` to run your application on a board
- `PMSTIS_OS=<os_name>` is used to change the OS running the application. `<os_name>` includes:
  - `pulpos`
  - `freertos`
- `GAPY_OPENOCD_CABLE=<cable>` to change the adapter used by OpenOCD to connect to the board. `<cable>` includes but not limited to:
  - `interface/ftdi/olimex-arm-usb-ocd-h.cfg`
  - `interface/jlink.cfg`
  - `interface/ftdi/sipeed-rv-debugger.cfg`

- `io=<option>` to change printf output type. `<option>` includes:
  - `host` to use JTAG output
  - `uart` to use UART output
- `runner_args=<options>` to specify arguments to the platform runner. Several arguments can be specified. Check the documentation of the platform to know the available arguments.
- `config_args=<options>` to specify options to customize the platform configuration. Check the documentation of the platform to know the available options.
- `LINK_SCRIPT=<path>` to specify a specific link script.
- `runner_args="<options>"` to pass runner specific options.
- `gdbserver=1` to launch execution with a GDB server so that GDB can be connected to the platform.
- `gdbport=<port>` to specify the GDB port when opening a GDB server.
- `CONFIG_BOOT_DEVICE=<value>` to specify the device from which to boot. `<value>` includes:
  - `spiflash` to boot from SPI flash
  - `hyperflash` to boot from hyperflash
  - `mram` to boot from mram
  - `spislave` to boot from SPI slave

## Gap9 options

- `CONFIG_XIP=<value>` to choose booting with xip or not. `<value>` includes:
  - `1` to boot using XIP
  - `0` to boot using standard L2 boot
- `CONFIG_BOOT_DEVICE_FREQUENCY=<value>` to specify the frequency used for booting from flash device. `<value>` gives the frequency in Hz.
- `CONFIG_FREQUENCY_PERIPH=<value>` to specify the frequency for the peripheral clock domain. `<value>` gives the frequency in Hz.
- `CONFIG_FREQUENCY_FC=<value>` to specify the frequency for the soc clock domain. `<value>` gives the frequency in Hz.
- `CONFIG_FREQUENCY_CLUSTER=<value>` to specify the frequency for the cluster clock domain. `<value>` gives the frequency in Hz.
- `CONFIG_FREQUENCY_SFU=<value>` to specify the frequency for the sfu clock domain. `<value>` gives the frequency in Hz.

## GVSOC options

- GV\_PROXY=1 to activate GVSOC telnet proxy to control it remotely.
- GV\_PROXY\_PORT=<value> to specify GVSOC telnet proxy port (default is 42951).
- GV\_DEBUG\_MODE=1 to activate GVSOC debug mode, which is required to allow trace activation. This is by default disabled, to simulate at maximum speed, and is automatically enabled when traces are enabled on the command-line. This mode can be enabled manually with this option when traces are enabled dynamically.

### 4.1.3 VSCode Integration

**Warning:** VSCode support is experimental. Use at your own risk.

SDK examples can be compiled, run and debug on boards from VSCode.

#### Prerequisites

To be able to debug your application, you will need to obtain a recent GDB from the official RISC-V GNU toolchain. You can find it here: <https://github.com/riscv-collab/riscv-gnu-toolchain>. It is possible to either compile it following the README there or to download a precompiled toolchain from the release page. Once acquired, either by compiling or downloading, please install/unpack it in a known location.

In this guide, it will be assumed that you used the XDG standard user directory `~/.local/bin` or another directory covered by your `$PATH` environment variable.

This way, you may access GDB using `riscv64-unknown-elf-gdb` directly. Also note that only GDB should be used from that toolchain, please continue to use the toolchain provided by GreenWaves Technologies for code compilation as otherwise you will encounter severe performance regressions.

#### Setup

VSCode is using some files such as `tasks.json` and `launch.json` to know how to build and run on the target.

Once the SDK is configured, you can generate a sample version of these files from an example with:

```
cd examples/pmsis/helloworld
make vscode
```

Everything should be properly setup for building, run and debug the example on GAP EVK with the embedded ftdi chip.

For using your customer boards, other jtag probe or even you want to change to use another gdb toolchain, you may need to modify the following lines in the `tasks.json` and the `launch.json`.

For changing the gdb toolchain: file: `launch.json`

```
"miDebuggerPath": "<absolute path to your gdb>,"
```

For changing the openocd, jtag probe: file: `tasks.json`

```
"command": "@OCD_install_path@ -c 'gdb_port 3333; telnet_port disabled; tcl_port disabled
↪ -f '@OPENOCD_CABLE_CFG@' -f '<sdk_path>/utils/openocd_tools/tcl/gap9revb.tcl'"
```

@OCD\_install\_path@ : Specify your openocd path @OPENOCD\_CABLE\_CFG@ : Specify your jtag probe config file path

### Launch VSCode

From your example folder, execute code .

You should see vscode with all the files from your example visible on the left panel. You can now edit and save all the files.

### Build the example

Click on “Terminal”->”Run Task”->”Build from Makefile”->”Continue without scanning the task output” to build your test.

Check any problem that occurs in the terminal window of vscode. If you don’t see anything happening, you may have run vscode without the --log trace option.

### Launch OpenOCD

Click on “Terminal”->”Run Task”->”Openocd”->”Continue without scanning the task output”

You should see on the terminal window, OpenOCD being launched and connecting to the target.

You can keep this terminal opened for several runs of your test, and you can kill it with the trash icon if something goes really wrong or you want to close it.

Once it’s launched, you will see:

```
Open On-Chip Debugger 0.10.0+dev-00841-g1449af5bd (2021-07-02-17:05)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "jtag". To override use
      'transport select <transport>'.
TAP: gap9.riscv

TAP: gap9.pulp

Info : clock speed 1000 kHz
jtag init
ret1=00000000
ret2=00000000
ret1=80007A16
ret=03
INIT: confreg polling done
Info : datacount=2 progbufsize=8
Info : Examined RISC-V core; found 10 harts
Info : hart 0: currently disabled
Info : hart 1: currently disabled
Info : hart 2: currently disabled
Info : hart 3: currently disabled
Info : hart 4: currently disabled
Info : hart 5: currently disabled
```

(continues on next page)

(continued from previous page)

```

Info : hart 6: currently disabled
Info : hart 7: currently disabled
Info : hart 8: currently disabled
Info : hart 9: XLEN=32, misa=0x40901124
examine done
Info : JTAG tap: gap9.riscv tap/device found: 0x20020bcb (mfg: 0x5e5 (<unknown>), part: 0x0020, ver: 0x2)
Info : JTAG tap: gap9.pulp tap/device found: 0x20021bcb (mfg: 0x5e5 (<unknown>), part: 0x0021, ver: 0x2)
Info : Listening on port 3333 for gdb connections
Ready for Remote Connections
Info : tcl server disabled
Info : telnet server disabled

```

## Launch the example

Click on “Run”->“Start Debugging” and then do normal vscode debug operations, like running, stopping, adding breakpoints and so on.

All the printf from the application is visible on the openocd terminal.

### 4.1.4 Flash applications and data

#### Flash an application

Flashing an application allows it to run without external intervention.

In order to flash an application you need to follow this procedure:

- Clean the project using `make clean`
- Compile and flash using `make all CONFIG_BOOT_DEVICE=<boot_device> io=uart`
  - `CONFIG_BOOT_DEVICE=<boot_device>` selects the flashing location. Option values are described in the [Application Makefile options](#) guide.
  - `io=uart` tells printf outputs to use UART instead of JTAG to avoid getting stuck doing semihost calls.
- select the correct boot mode, either via boot pads or via efuses.
- reboot

#### Flash data

##### Flash data using filesystems

###### Using ReadFS

ReadFS is a small read-only filesystem. To flash some files/directories simply add in the application Makefile:

- a `READFS_FILES` variable pointing to the files you want to include
- a `READFS_DIRS` variable pointing to the directories you want to include recursively
- a `READFS_DIRS_NO_REC` variable pointing to the directories you want to include non-recursively

The image will be generated and flashed automatically when executing the `make all` command.

#### 4.1.5 How to use CMake

##### Description

CMake is a cross-platform tool that manages the build process of a software. It intends to be an alternative way to build the SDK compared to actual Makefile way. CMake process is performed in two steps :

- Configuration: Setup cmake environnement.
- Build: Building the software using a specific build system (Unix Makefile, MinGW Makefile, Ninja, Visual Studio, ...)

In this release, CMake is implemented with FreeRTOS and GVSOC.

##### Prerequisites

- Please run the `sourceme.sh` script before using Cmake. Select the GAP9\_V2 configuration.

##### Use CMake to build your application

In order to configure your application with CMake, you need first to create a `CMakeList.txt` file next to it.

This file intends to tells CMake what are your application's sources then call SDK's root `CMakeList.txt` file.

This file must be implemented as the following way.

Please, have a look to the hello example before continuing this guide : `gap_sdk/rtos/pmsis/tests/api/quick/hello` Basically, you must adapt the “Panel control” section to your application. Here is a more detailed description of what does this file :

- In the “panel control” section, you need to specify your application name in the macro `TARGET_NAME` and add your application's source in the macro `TARGET_SRCS`.
- The “CMake pre initialization” section will call for CMake files that setup the environnement :
  - Build menuconfig target
  - Select flags depending on what board is used.
  - Find RISCV toolchain
- The “App's options interpretation” section consists in read the output of the KConfig configuration. For more information, check [How to use Kconfig](#)
- Finally, the “CMake post initialization” section will setup the following features:
  - Setup the OS (set name and path to call the dedicated `CMakeList.txt` file)
  - Link OS's CMake library to your application
  - Find GAPY tool
  - Build targets such as `dump_dis` `dump_size` `image` `flash` and `run`

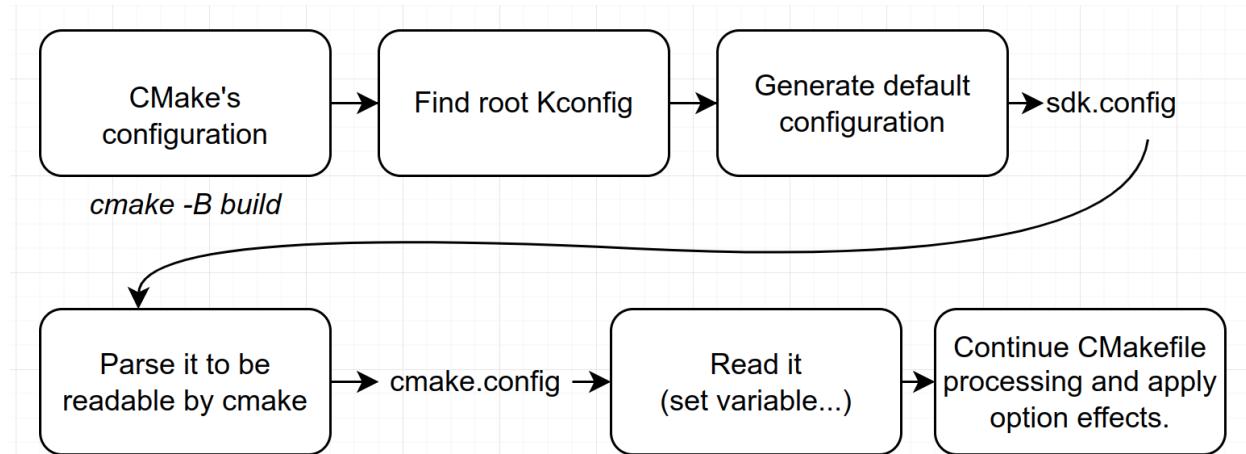
## CMake commands

To compile your project with CMake, you need to run the following commands :

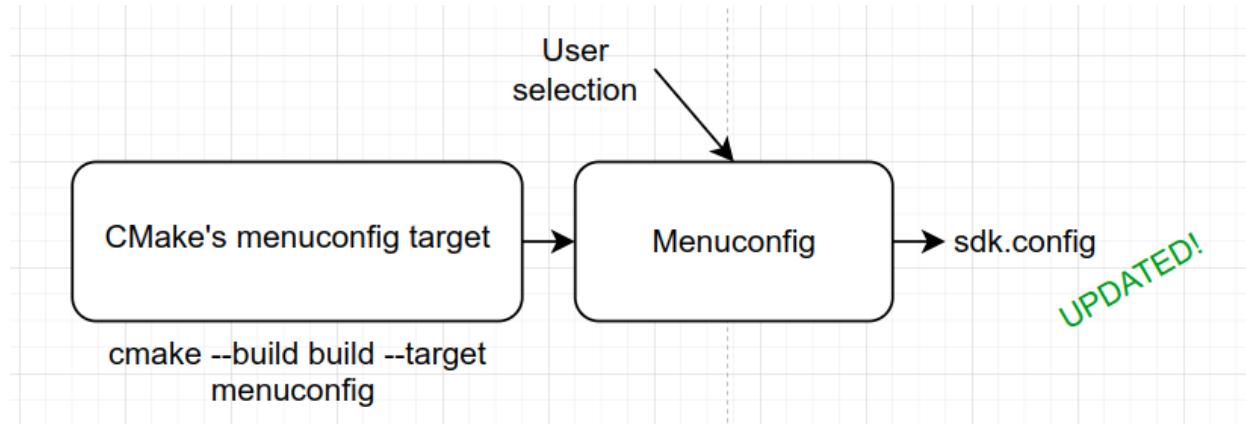
- `cmake -B <name_of_your_binary_directory>` Configure CMake in the specified directory. It is usually :  
`cmake -B build`
- `cmake --build <name_of_your_binary_directory> --target menuconfig` Run the SDK configuration step. It calls for menuconfig interface. Here you can select what options you want for your application. See [How to use Kconfig](#) for more informations.
- `cmake --build <name_of_your_binary_directory> --target run` Build your application then run it.

Here is a graphical view detailling what these three steps do.

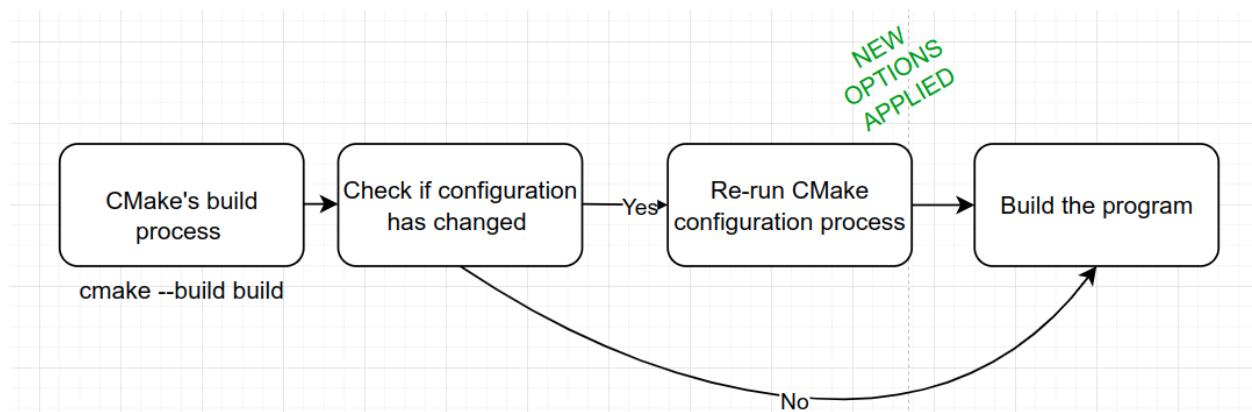
- Step 1



- Step 2



- Step 3



#### 4.1.6 How to use Kconfig

##### Description

KConfig is a language that describe application's settings. It is used here to describe how we want the SDK to behave. KConfig comes with a graphical interface, menuconfig, to configure these settings according to your application. This release of KConfig features works only by building your application with CMake. See [How to use CMake](#) guide for more information

##### Prerequisites

- Kconfig uses GAP\_SDK\_HOME environnement. Be sure that it has been set before running menuconfig command.
- You need to add export the following environnement variable. Please add it to your .bashrc or its equivalent.

```
export KCONFIG_CONFIG="sdk.config"
```

##### How it works

Menuconfig graphical interface is built from SDK's Kconfig files. Each one describes a list of options. It is recommended to distributed KConfig options into several Kconfig files, one per SDK stage.

KConfig files are organized as a tree. It means that a root Kconfig file will call sub Kconfig file and so on. At the end, all Kconfig file's content will be visible from menuconfig interface.

Here is an example of a KConfig file related to the RTOS folder of the SDK.

```

menu "RTOS Menu"
choice PMSIS_OS
    prompt "OS Selection"
    config PMSIS_OS_FREERTOS
        bool "Freertos"
        help
            Select FreeRTOS as GAP SDK OS. Default value
    config PMSIS_OS_PULPOS
        bool "Pulpos"
        help
  
```

(continues on next page)

(continued from previous page)

```

        Select Pulp OS as GAP SDK OS
endchoice
if PMSIS_OS_FREERTOS
    source "$(GAP_SDK_HOME)/rtos/pmsis/os/freeRTOS/Kconfig"
endif
if PMSIS_OS_PULPOS
    # TODO: Source PulpOS Kconfig here
endif
menu "PMSIS Menu"
    source "$(GAP_SDK_HOME)/rtos/pmsis/Kconfig"
    source "$(GAP_SDK_HOME)/rtos/pmsis/os/freeRTOS/vendors/gwt/gap9/pmsis/Kconfig"
    source "$(GAP_SDK_HOME)/rtos/pmsis/os/freeRTOS/vendors/gwt/pmsis/rtos/Kconfig"
endmenu
source "$(GAP_SDK_HOME)/rtos/pmsis/bsp/Kconfig"
endmenu

```

Menuconfig interface is executed from CMake command. See [How to use CMake](#) guide for more information. Here, you can select what options you want to set for your application.

Once you are done, Menuconfig produces an output file named  `sdk.config`. This file described all enabled options.

Here is an example of what the  `sdk.config` file look like

```

#
# Helloworld Menu
#
CONFIG_HELLOWORLD_CLUSTER=y
# end of Helloworld Menu
#
# GAP SDK Menu
#
# General options
#
CONFIG_CHIP_FAMILY_GAP9=y
CONFIG_PLATFORM_GVSOC=y
# CONFIG_PLATFORM_BOARD is not set
CONFIG_BOARD_GAP9_EVK=y
# CONFIG_BOARD_GAP9_EVK_AUDIO_ADDON is not set
# CONFIG_BOARD_GAP9_V2 is not set

```

Then, CMake uses a python script to parse this output file to be able to read it. Basically, it consists in setting variable to a specific value in a new file named  `cmake.config`

Here is an example of what  `cmake.config` look like:

```

# Helloworld Menu
set(CONFIG_HELLOWORLD_CLUSTER y)
# GAP SDK Menu
set(CONFIG_CHIP_FAMILY_GAP9 y)
set(CONFIG_PLATFORM_GVSOC y)
set(CONFIG_BOARD_GAP9_EVK y)

```

These variables are then tested in CMake configuration process to add sources to compile or add flags to the compilation process.

## How to add a new option

Depending on what is the purpose of your option, you need first to identify where it belong. Usually, it will be described in a Kconfig file located next to the files that will be impacted by your option.

- Assuming that you will create an option for a SDK’s software component that has not been covered yet, you need first to create the Kconfig file.
- In it, create a menu with a relevant name and describe your options. Check “Resources” section to learn kconfig syntax.
- Then, you need to link your new Kconfig file with an existing one that will be its parent using the source keyword. At this point, your options are visible in the menuconfig interface.
- Then you need to apply effects related to your option. In a CMakeLists.txt file located next to your Kconfig file and your option’s related source files, you can check your option in the following way :

```
if(DEFINED CONFIG_<your_option_name>
    #Add flag
    target_compile_options(<your_target> PRIVATE "-Dyour_flag")
    list(APPEND TARGET_SOURCE_FILES "another_src_file_to_compile.c")
endif()
```

In the helloworld example, An option consists in enabling cluster cores to print hello as the fabric controller.

```
if(DEFINED CONFIG_HELLOWORLD_CLUSTER)
    message(STATUS "[${TARGET_NAME} Options] Cluster enabled")
    target_compile_options(${TARGET_NAME} PRIVATE "-DCONFIG_HELLOWORLD_CLUSTER=1")
else()
    message(STATUS "[${TARGET_NAME} Options] Cluster disabled")
endif()
```

**Warning:** During the generation of sdk.config, all options name are prepended by CONFIG\_. Please don’t forget to add it while testing your options in CMakeList files. In the future this feature can be canceled in the python parsing process.

In the helloworld example, the flag CONFIG\_HELLOWORLD\_CLUSTER is then tested in a #ifdef preprocessor macro.

```
#if defined(CONFIG_HELLOWORLD_CLUSTER)
void pe_entry(void *arg)
{
    printf("Hello from (%d, %d)\n", pi_cluster_id(), pi_core_id());
}

void cluster_entry(void *arg)
{
    pi_cl_team_fork(0, pe_entry, 0);
}
#endif

static int test_entry()
{
#if defined(CONFIG_HELLOWORLD_CLUSTER)
    struct pi_device cluster_dev;
```

(continues on next page)

(continued from previous page)

```

struct pi_cluster_conf cl_conf;
struct pi_cluster_task cl_task;

pi_cluster_conf_init(&cl_conf);
pi_open_from_conf(&cluster_dev, &cl_conf);
if (pi_cluster_open(&cluster_dev))
{
    return -1;
}
pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_entry,NULL));
pi_cluster_close(&cluster_dev);
#endif

printf("Hello\n");

return 0;
}

static void test_kickoff(void *arg)
{
    int ret = test_entry();
    pmsis_exit(ret);
}

int main()
{
    return pmsis_kickoff((void *)test_kickoff);
}

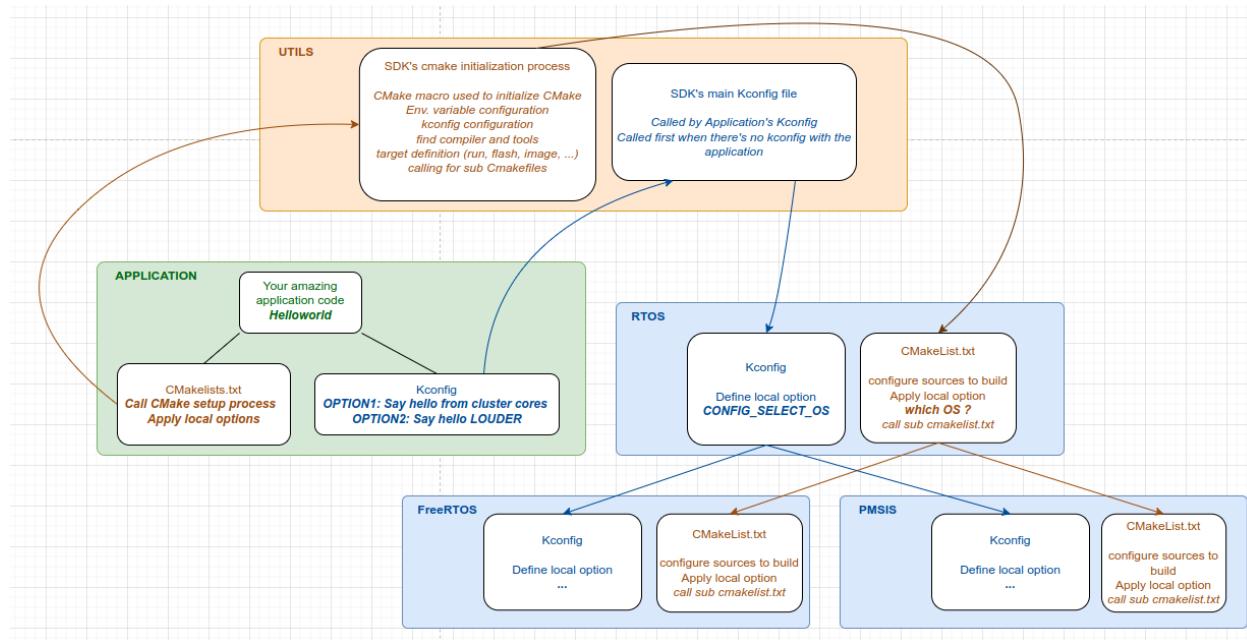
```

### Create options related to your application

When it is about creating options for your application, you need to understand how Kconfig files include each others. During its configuration process, CMake search for the “Root Kconfig” a.k.a the main one. It can be the SDK one located in `utils/kconfig/` directory or the application one located next to your application. In this last scenario, application Kconfig must call for the SDK one with the “source” keyword. Otherwise, You will not see any option in your menuconfig interface. Then, SDK’s kconfig will call for sub kconfig files as seen before.

Check hello example’s Kconfig file located here in the SDK : `rtos/pmsis/tests/api/quick/hello/Kconfig`

Here is an schematic about how Kconfig and Cmake files are organized in the SDK :



## Resources

Kconfig documentation : <https://www.kernel.org/doc/html/latest/kbuild/kconfig-language.html>

### 4.1.7 Using FreeRTOS with PMSIS API and drivers

#### Introduction

FreeRTOS is the default OS in GAP SDK. While it provides the standard FreeRTOS APIs to users, it also leverages PMSIS (Pulp Microcontroller Software Interface Standard), both for drivers and some standard system tasks. This guide explains the basic rules to use FreeRTOS and PMSIS together.

#### Startup

At startup, in the application `main`, `pmsis_kickoff` will have to be called first. This will start the scheduler, as well as a small event kernel. Those two are hard requirements to use any PMSIS driver.

**Warning:** `vTaskSchedulerStart` from FreeRTOS **must not** be called manually. Since `pmsis_kickoff` already does it, this would cause undefined behaviour.

## Tasks

TODO

---

**Note:** Some warning about multi tasks, and push async in particular.

---

### Inter tasks synchronisation

All the PMSIS API uses pi\_evt structures for synchronisation. These can be used either for semaphore like construct (initialized with pi\_evt\_sig\_init), or callbacks (initialized with pi\_evt\_callback\_no\_irq\_init).

These can also be used between tasks, and between a task and callbacks themselves. xTaskNotify and xTaskSemaphore can also be used, however, pi\_evt have the advantage of being directly usable inside PMSIS functions.

As an example, one could dispatch from a thread and block on another via blocking.

```
pi_evt_t evt_write;
pi_evt_t evt_read;
void task1(void)
{
    /*
     ...
    */
    while (something)
    {
        // wait on task, and reinit it
        pi_evt_wait_on(&evt_write);
        pi_evt_destroy(&evt_write);
        pi_evt_sig_init(&evt_write);
        pi_some_device_read(some_device, some_buffer, some_size, &evt_read);
    }
    /*
     ...
    */
}
void task2(void)
{
    /*
     ...
    */
    while (something)
    {
        pi_some_device_write(some_device, some_buffer, some_size, &evt_write)
        // wait on task, and reinit it
        pi_evt_wait_on(&evt_read);
        pi_evt_destroy(&evt_read);
        pi_evt_sig_init(&evt_read);
    }
    /*
     ...
    */
}
```

(continues on next page)

(continued from previous page)

```

    */
}

void main_task(void)
{
    /*
    ....
    */
    pi_evt_sig_init(&evt_write);
    pi_evt_sig_init(&evt_read);
    /*
    ....
    */
}

```

**Note:** `pi_evt_t` structures are by nature thread/event safe objects

## Timers

### Precise timers

**Warning:** This only applies to GAP9, on GAP8, these APIs are tick based.

PMSIS API provides access to precise hw timers. This API's resolution is in the order of 100 micro seconds. Its two main components are:

`pi_time_wait_us(uint32_t timeout_us)`: This API allows to block the current thread for `timeout_us` micro seconds.

`pi_evt_push_delayed_us(pi_evt_t *evt, uint32_t timeout_us)`: This API allows to execute a callback or unlock a semaphore `timeout_us` micro seconds in the future.

These APIs allow to completely replace FreeRTOS's `vTaskDelay` and `xTimerXXX` APIs. Those have the advantage of being much more precise and lightweight.

The only non direct replacement would be a periodic timer. It is however extremely straightforward to do:

```

/*
.....
*/

// example callback that re-pushes itself, effectively creating a periodic
// timer
void my_pi_periodic_timer( void *arg )
{
    pi_evt_t *evt = (pi_evt_t *) arg;
    // re-push the same event with same delay
    pi_evt_push_delayed_us(evt, 1000);
}

```

(continues on next page)

(continued from previous page)

```

void test_sw_timer( void )
{
    /*
     ....
    */
    pi_evt_t cb;
    // pass task itself as callback argument, for real case, using
    // a structure containing all desired information is recommended.
    pi_evt_callback_no_irq_init(&cb, my_pi_periodic_timer, (void*) &cb);
    // delayed push, 1ms
    pi_evt_push_delayed_us(&cb, 1000);
    /*
     ....
    */
}
/*
.....
*/

```

---

**Note:** For full documentation on those APIs, please see API documentation.

---

### Imprecise tick based timer

FreeRTOS provides imprecise timers, which are based on tick resolution.

---

**Note:** By default, FreeRTOS tick rate is 10ms on GAP9, 1ms on GAP8.

---

**Warning:** The use of these APIs in non legacy code is NOT recommended. These APIs are much less precise and incur larger performance cost than their PMSIS counterparts.

These are used via FreeRTOS APIs, and can be one-shot or periodic. Once a timer is done, it calls the callback given as argument.

An example of one shot timer is as follows:

```

/*
....
*/
// a timer array
TimerHandle_t xTimers[ TIMERS ];

// example timer callback - just print and release creating task
void vTimerCallbackOneShot( TimerHandle_t xTimer )
{
    char *taskname = pcTaskGetName(NULL);
    printf("%s : Callback at TICK = %d.\n", taskname, xTaskGetTickCount());

```

(continues on next page)

(continued from previous page)

```

    pi_evt_t *evt_block = (pi_evt_t *) pvTimerGetTimerID(xTimer);
    pi_evt_release(evt_block);
}

void test_sw_timer ( void )
{
    /*
    ....
    */
    // create a freertos timer, with a blocking event as argument
    xTimers[0] = xTimerCreate("Timer0", pdMS_TO_TICKS( 10 ), pdFALSE,
        (void*) &evt_block, vTimerCallback0);
    if (xTimers[0] == NULL)
    {
        exit(0);
    }
    // start the timer
    xTimerStart( xTimers[0], portMAX_DELAY );
    printf("%s created and started one shot timer. Waiting.\n", taskname);
    /*
    ....
    */
}
/*
....
*/

```

The callback will be called once, and only once. After which, the timer will stop on its own and need to be rearmed for further usage.

An example of a periodic timer is as follows:

```

/*
....
*/
// a timer array
TimerHandle_t xTimers[TIMERS];

static uint32_t call_nb = 0;
// example timer callback - just print and release after 10 cycles
void vTimerCallbackPeriodic( TimerHandle_t xTimer )
{
    call_nb++;
    char *taskname = pcTaskGetName(NULL);
    printf("%s : Callback at TICK = %d.\n", taskname, xTaskGetTickCount());
    pi_evt_t *evt_block_periodic = (pi_evt_t *) pvTimerGetTimerID(xTimer);
    if (call_nb >= 10)
    {
        pi_evt_release(evt_block_periodic);
        // stop the timer from triggering again
        xTimerStop(xTimers[1], portMAX_DELAY);
    }
}

```

(continues on next page)

(continued from previous page)

```

void test_sw_timer( void )
{
    /*
     ....
    */
    pi_evt_wait_on(&evt_block);
    xTimers[1] = xTimerCreate("Timer1", pdMS_TO_TICKS(10), pdTRUE,
        (void *) &evt_block_periodic, vTimerCallback0);
    if (xTimers[1] == NULL)
    {
        exit(0);
    }
    // start the timer
    xTimerStart( xTimers[1], portMAX_DELAY );
    /*
     ....
    */
}
/*
.....
*/

```

The periodic callback will be called 10 times, until call\_nb reaches 10. At that point, the timer will be stopped.

### Saving dynamic power

Any time nothing happens on the core, it will be clock gated. Thus saving power. To reach that state, the condition is to have no thread to schedule. As a consequence it is extremely important not to make threads artificially busy. This forbids most `while(true){wait_ms()}` constructs. Instead, code needs to wait on a semaphore or a pi\_task initialized via `pi_evt_sig_init()`.

example:

```

/* PMSIS includes */
#include "pmsis.h"

#define TIMERS ( 1 )
TimerHandle_t xTimers[TIMERS];

void vTimerCallback0( TimerHandle_t xTimer )
{
    char *taskname = pcTaskGetName(NULL);
    printf("%s : Callback at TICK = %d.\n", taskname, xTaskGetTickCount());
    pi_evt_t *evt_block = (pi_evt_t *) pvTimerGetTimerID(xTimer);
    pi_task_release(evt_block);
}

int main(void)
{
    printf("\n\n\t *** SW Timer Test ***\n\n");
}

```

(continues on next page)

(continued from previous page)

```

char *taskname = pcTaskGetName(NULL);
uint32_t ulValue = 0, wait_val = 0;

pi_evt_t evt_block;
// Initialize a blocking task, equivalent to a semaphore
pi_evt_sig_init(&evt_block);
printf("%s creating Timers.\n", taskname);
// create a freertos timer, with said blocking task as argument
xTimers[0] = xTimerCreate( "Timer0", pdMS_TO_TICKS( 10 ), pdFALSE,
                           (void *) &evt_block, vTimerCallback0 );
if (xTimers[0] == NULL)
{
    exit(0);
}

// start the timer
xTimerStart( xTimers[0], portMAX_DELAY );

printf("%s created and started timers. Waiting.\n", taskname);

pi_evt_wait_on(&evt_block);

printf("%s deleting timers and suspending.\n", taskname);
xTimerStop( xTimers[0], portMAX_DELAY );
xTimerDelete( xTimers[0], portMAX_DELAY );

pmsis_exit(0);
}

```

In this example, one task wait on an event, here coming from a one shot timer. During all the wait the task is blocked, and therefore never scheduled. Since no other tasks are present, FreeRTOS will execute its idle task, which will clock gate the core until the next IRQ which could change the system state. As such, 10ms of dynamic power are potentially saved.

#### 4.1.8 Console IO

GAP and GAP SDK provides you the possibilities of console (printf) with JTAG or UART.

##### Console IO via JTAG

The make option `io=host`, the default console mode, will direct the printf on the jtag. The printf in this case is implemented based on semihosting, works with openOCD.

**BETWEEN**, this mode ONLY can be used when boot from JTAG, it will cause the process hang if it's used in boot from flash.

## Console IO via UART

This mode will triggered by using the option `io=uart`, which will redirect the printf on the UART.

By default, the uart will be configured as:

- UART ID: 1
- Baudrate: 115200
- HW Flow Control : No
- Timeout : No

All these parameters can be configured by using following makefile options:

- UART ID: `CONFIG_IO_UART_ITF=<UART ID you want to use>`
- Baudrate: `CONFIG_IO_UART_BAUDRATE=<baudrate of uart>`
- Enable the HW Flow Control : `APP_CFLAGS += -DCONFIG_UART_IO_FLOW_CONTROL`
- Enable      Timeout      :                    `APP_CFLAGS += -DCONFIG_UART_IO_UART_TIMEOUT -DCONFIG_UART_IO_UART_TIMEOUT_US=<timeout in us>`

These options can be used separately to enable/disable the feature(s) you want. BEWARE, the HW Flow Control cannot be used with the FTDI2232 chip on GAP EVK.

## Disable the Console

If you don't need printf anymore, we strongly suggest you to disable the console with `io=disable`, which will help you to save the footprint of your application.

## 4.2 GAP9

### 4.2.1 Debugging an application with GDB

**Warning:** All instructions in this document are only valid with GAP9. **GAP8 has no support for debugging.**

---

#### Note:

- Cluster debugging is not yet available but will be in a future release.
  - GDB support is **only available when using a board**.
  - Debugging will not work if your application performs actions that might disable the JTAG connection (deep sleep as an example). To debug such applications please remove all such actions.
-

## Prerequisites

To be able to debug your application, you will need to obtain a recent GDB from the official RISC-V GNU toolchain. You can find it here: <https://github.com/riscv-collab/riscv-gnu-toolchain>. It is possible to either compile it following the README there or to download a precompiled toolchain from the release page. Once acquired, either by compiling or downloading, please install/unpack it in a known location. In this guide, it will be assumed that you used the XDG standard user directory `~/.local/bin` or another directory covered by your `$PATH` environment variable. This way, you may access GDB using `riscv64-unknown-elf-gdb` directly. Also note that only GDB should be used from that toolchain, please continue to use the toolchain provided by GreenWaves Technologies for code compilation as otherwise you will encounter severe performance regressions.

## Using GDB to debug your application

Firstly, instead of running the application on the platform with the usual command, you can open a GDB server with the following command:

```
make run gdbserver=1
```

Note the `gdbserver=1` which is the item of interest here. This will launch the execution but will stop before the application is started, waiting for GDB to connect and issue commands.

On the terminal, you should see:

```
Ready for Remote Connections
```

This means a gdbserver, connected to your board is now active on port `localhost:3333`.

If needed, another GDB port can be specified with the following option:

```
make run gdbserver=1 gdbport=3333
```

Now, in another terminal, execute the following command (assuming your application is named `test`):

```
riscv64-unknown-elf-gdb BUILD/GAP9_V2/GCC_RISCV_FREERTOS/test
```

This will open a GDB prompt, you can then enter the following GDB commands:

```
(gdb) target remote :3333  
(gdb) load
```

And you should see a message about code sections loading. This means that your application binary is now loaded on board. You may now use regular GDB features such as breakpoints. Once you have setup your breakpoints, you can run with `continue`.

If you need to reset the board use `monitor reset` and then load again.

## 4.2.2 Using eXecute In Place to extend internal memory

**Warning:** All instructions in this document are only valid with GAP9, **GAP8 has no support for eXecute In Place (XIP).**

---

**Note:** Data accesses are supported via XIP too, but at the cost of fairly increased latency, please only use it outside of critical paths.

---

### Introduction

GAP9 comes with XIP support by default. This can help you to use either internal eMRAM flash or an external Flash/RAM as if it was standard internal memory. This is particularly useful if your application is big (in terms of code), or if you have a lot of support data that does not require high-speed accesses (a device tree as an example). This document will present both code only and code+data XIP usage.

### Limitations

- It is **not** possible to write in flash (whether eMRAM flash or external flash) from XIP.
- When using XIP, actions that would lock it (closing the supporting device, reconfiguring XIP) can not be performed from XIP executed code. For those, some methods will need to be present in L2 memory.

### Using XIP for code execution

Using XIP for code execution requires two things:

- First, flash application binary in a supported flash (eMRAM or select OctoSPI flash)
- Second, when running, the boot device and XIP config need to be given to the runner.

Both of those can be done with the following command:

```
make clean all run CONFIG_XIP=1 CONFIG_BOOT_DEVICE=mram
```

`CONFIG_XIP=1` instructs the runner to use XIP linker scripts and flash generators. It will also set a compile-time macro with the same name in the code, used to protect memory controller access in firmware. `CONFIG_BOOT_DEVICE=mram` instructs the runner to flash the code in chosen flash, here the eMRAM flash.

And that's all, you now have XIP executed code running on board. A sample linker script for XIP can be found in `$GAP_SDK_ROOT/rtos/pmsis/os/freeRTOS/vendors/gwt/gap9/src/device/ld/GAP9_xip.ld`. It is to be noted that the ldscript only sees XIP addresses, which are virtual. There are no such things as eMRAM addresses, or external flash addresses, as those are *not* memory mapped in GAP architecture. Outside of XIP, those are only available via manual DMA accesses. As such, a binary compiled for XIP can execute indifferently from any external memory, as long as XIP code is properly configured to align `0x20000000` virtual memory address with the binary starting point in external memory. If using standard ROM boot this is done automatically, if using an SSBL, the SSBL programmer is responsible for it.

## Using XIP for data accesses

Using XIP for data accesses can be done on any flash/ram device. However, it can't be used to write in flashes (including internal eMRAM) and latency is higher than an L2 access. Thus, its main use case is for data that is not accessed in the application's critical paths. A typical use case would be a device tree, or any data which won't be intensively reused, and would thus only waste precious L2 space. Another use case would be to load a full piece of foreign code (client or provider binary blob).

In that case, the client or provider can compile their full application using XIP base addresses, which the firmware may load in external RAM. This, compiled with user-mode execution can allow full isolation of binaries from the rest of the system, as well as easier binary blob distribution.

Using XIP on data is based on existing memory controller drivers (eMRAM, Octospi notably), and an ad-hoc XIP driver.

On the memory controller driver, one only needs to add `xip_en = 1` to the configuration. Here is an example:

```
pi_default_ram_conf_t ram_conf;
pi_device_t ram;
pi_default_ram_conf_init(&ram_conf);
ram_conf.xip_en = 1; // activate xip support on the device
pi_open_from_conf(&ram,&ram_conf);
pi_flash_open(&ram);
uint32_t ram_buffer = pi_ram_alloc(BUFFER_SIZE);
```

Here, XIP support has been requested, and a buffer in ram (ram physical address) has been reserved.

Then, the XIP itself needs to be configured:

```
uint32_t page_mask = 0;
pi_device_t *xip_device = pi_l2_malloc(sizeof(pi_device_t));
for (int i = 0; i<XIP_NB_DCACHE_PAGE; i++)
{
    // Allocate L2 memory chunks for XIP pages
    pi_xip_dcache_page_alloc(i, XIP_PAGE_SIZE);
}
// request a certain number of those pages
pi_xip_free_page_mask_get(&page_mask,XIP_NB_DCACHE_PAGE);
pi_xip_conf_t xip_conf = {0};
// ram is not read only
xip_conf.ro = 0;
// adapt depending on the interface, octospi0 is 0, octospi1 is 1, mram is 2
xip_conf.per_id = XIP_DEVICE;
// choose a page size (see xip driver documentation for more detail)
xip_conf.page_size = XIP_PAGE_SIZE;
// use the pages that were requested
xip_conf.page_mask = page_mask;
// Prepare a mount size, expressed as a number of pages
// real page size is equal to PAGE_SIZE << 9 - smallest page size is 512B
uint32_t mount_size = BUFFER_SIZE / (PAGE_SIZE << 9);
// now mount the area on desired memory controller, at first xip virtual address
// as such, ram_buffer will be accessible via xip at address 0x20000000
pi_xip_mount(xip_device,0x20000000,ram_buffer,mount_size,0)
// access can then simply be done with:
*((uint32_t*)0x20000000) = SOME_VALUE; // Warn: only possible to write in ram
```

## Using two devices on the same interface

In the case where two devices (one on each available Chip Select) are used on the same interface, some care needs to be taken. Firstly, it is only possible to map one area on one device at a given time. As such, two possibilities arise:

- Either map each area in turn (umount one, mount the other)
- Map the full devices together

In case both devices are mounted together, the mount will begin with external address `0x0`, and with a size of `CS1_BASE + CS1_SIZE`, the base being the mapping base address given as input for the device on CS1. Due to a hardware limitation, `CS1_BASE` must be a multiple of `CS1_SIZE`. As such if `CS1_BASE` is bigger than `CS0_SIZE` there will be a “hole” in the mapping between `XIP_BASE+CS0_SIZE` and `XIP_BASE+CS1_BASE`.

### 4.2.3 Using AES to secure your memory accesses

**Warning:** All instructions in this document are only valid with GAP9, **GAP8 has no hw support** for AES.

---

**Note:** SW implementations are possible, but are out of the scope of this document.

---

#### Introduction

In GAP9, two different AES IPs can be found.

- AES Dual Core (AESDC) is directly linked with Octospi IPs
- AES is made for L2 to L2 encryption/decryption

This document will focus on AESDC. In particular, with methods to protect your code and data stored in external memory.

#### Limitations

Using AESDC, AES is restricted to ECB mode. Data must also be aligned on 16 bytes.

#### GAP9 encrypted flash access

In this section, it is considered that the flash content is already encrypted. This can be done via various PC tools which are not presented here. The encrypted file can then be flashed with usual methods. See relevant documentation for more information on this part.

To access the encrypted flash with seamless decryption, an aes configuration must first be setup. This is done like so:

```
// first declare a key
static uint8_t aes_key_ecb_128[16] = {
    0x00, 0x01, 0x02, 0x03,
    0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B,
    0x0C, 0x0D, 0x0E, 0x0F,
};
```

(continues on next page)

(continued from previous page)

```
// secondly prepare an aes configuration from it
static pi_aes_utils_conf_t aes_configurations[] = {
{
    /* ECB 128 */
    .enabled = 1,
    .mode = PI_AES_MODE_ECB,
    .key_len = PI_AES_KEY_128,
    .key = (uint32_t*) aes_key_ecb_128,
    .iv = NULL,
},
};
```

Detailed elements are described in AES pmsis\_api reference. In short, this is an ECB, 128 bits configuration with the key 0123456789ABCDEF. The first thing to do in a real case would be to use the encryption key used on PC.

Secondly, the flash configuration will be updated to use this AES configuration.

```
/* Init & open flash. */
pi_default_flash_conf_init(&flash_conf);
// add aes configuration
memcpy(&(flash_conf.flash.aes_conf), &aes_configurations[0], sizeof(pi_aes_utils_conf_t));
// open normally
pi_open_from_conf(&flash_dev, &flash_conf);
if (pi_flash_open(&flash_dev))
{
    printf("Error flash open !\n");
    pmsis_exit(-3);
}
```

And that is all, all read (and writes) to the flash will now be seamlessly encrypted/decrypted in HW. Simply execute a pi\_flash\_read at the adress where the encrypted binary was stored.

## GAP9 encryption and decryption of RAM

The way to proceed is essentially the same as for flash configuration, except that no flashing and no PC side preparation has to be done.

The same aes key and configuration is first prepared:

```
// first declare a key
static uint8_t aes_key_ecb_128[16] = {
    0x00, 0x01, 0x02, 0x03,
    0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B,
    0x0C, 0x0D, 0x0E, 0x0F,
};
// secondly prepare an aes configuration from it
static pi_aes_utils_conf_t aes_configurations[] = {
{
    /* ECB 128 */
    .enabled = 1,
    .mode = PI_AES_MODE_ECB,
```

(continues on next page)

(continued from previous page)

```
.key_len = PI_AES_KEY_128,
.key = (uint32_t*) aes_key_ecb_128,
.iv = NULL,
},
};
```

Then, this configuration is attached to the ram before opening it:

```
/* Init & open ram. */
pi_default_ram_conf_init(&ram_conf);
/* add aes to octospiram */
memcpy(&(ram_conf.ram.aes_conf), &aes_configurations[0], sizeof(pi_aes_utils_conf_t));
pi_open_from_conf(&ram_dev, &ram_conf);
pi_ram_open(&ram_dev);
```

Now, considering a buffer of data `secret_data` has been created, rewrite it securely like so:

```
pi_ram_alloc(&ram_dev, &ram_buff, BUFFER_SIZE)
pi_ram_write(&ram_dev, ram_buff, secret_data, (uint32_t) BUFFER_SIZE);
// and to access it, simply use:
pi_ram_read(&ram_dev, ram_buff, rcv_secret_data, (uint32_t) BUFFER_SIZE);
```

## Using with XIP

If using with XIP, nothing particular has to be done. Only one limitation applies, if two CS are used with XIP, then they must both be encrypted. XIP does not support per CS encryption as it is seen as a unique mapping.

### 4.2.4 Using low power modes

**Warning:** This guide is **only for GAP9**.

**Note:** To have more details about low-power modes, please refer to GAP9 datasheet.

One of GAP9 goals is to have an ultra-low power consumption. As such, the chip has a bunch of low power modes, including:

- Idle mode: A low-overhead mode to save power if no activity will occur during short amount of time.
- Light-sleep: Most power domains are turned off except for memory banks and IOs.
- Deep-Sleep Retentive: Every power domain is turned off, but some L2 memory banks can be saved depending on configuration.
- Deep-Sleep: Every power domain and memory bank is turned off.

### Low power modes

#### Idle mode

**Warning:** Not yet supported on FreeRTOS.

This mode is not exactly a sleep mode (the only wakeup source is a counter). It only gates GAP9 clocks for some time in order to save power. It reduces the overhead to enter/exit this mode but limits power-savings.

It only takes a couple of  $\mu$ s to do a full sleep-wakeup cycle.

You can see how to use this power mode in the [\*Idle mode example\*](#).

#### Light-Sleep

**Warning:** Not yet supported.

The chip powers down, keeps data in L2 memory banks and switchable IOs depending on existing configuration.

It only takes a couple of hundred  $\mu$ s to do a full sleep-wakeup cycle.

You can see how to use this power mode in [TODO this example](#).

#### Deep-Sleep Retentive

**Warning:** Not yet supported on FreeRTOS.

The chip powers down but keeps data in L2 memory banks depending on configuration. You can select which memory bank to keep and which one you want to turn off. Power consumption in this mode depends on the configuration. The more memory you keep retentive the more power consumed.

When waking-up from this mode all hardware configuration is lost. Peripherals need to be reopened.

It takes about 1.2ms to do a full sleep-wakeup cycle.

You can see how to use this power mode in [TODO this example](#).

#### Deep-sleep

The chip completely powers down, including memory banks.

When waking-up from this mode all hardware configuration is lost. Peripherals need to be reopened.

It takes about 2.5ms to do a full sleep-wakeup cycle.

You can see how to use this power mode in the [\*Deep Sleep example\*](#).

## Wakeup sources

In sleep mode, the wakeup process can be triggered from multiple sources.

### Counter

Before going to sleep, a counter is set to a determined amount of time. Once in sleep mode this counter will increment with each reference clock cycle. And when this counter is reached the wakeup sequence will start.

You can see how to use this wakeup source in [TODO this example](#).

### RTC

When the 32kHz slow clock is available, it can be configured to trigger interrupts at precise times over long ranges to wake-up the chip.

You can see how to use this wakeup source in the [Deep Sleep example](#).

### SPI

An SPI-slave can be set to listen during sleep mode for a wakeup command (0x3F). Once this command is received the wakeup process will start. To read the wakeup status, the master should send the status command (0x05). This status will be 0 until GAP9 wakes up and turns it to 1. To complete the wakeup and allows GAP9 to return to a nominal behavior the master should send an handshake command (0x71).

You can see how to use this wakeup source in [TODO this example](#).

### GPIO

A GPIO can be set to trigger the wakeup on a specific event. Once the event happens, the chip will wakeup.

The GPIOs that can be used as wakeup sources are detailed in the datasheet.

You can see how to use this wakeup source in [TODO this example](#).

## 4.2.5 Using the cluster

**Warning:** This guide is **only for GAP9**.

### Execution model

When the chip has just started-up and entered the main of the application, the whole cluster, including the cores, is off.

Once the cluster is powered-on using the cluster driver, all the cores start executing some start-up code to enter a loop where they are waiting for tasks sent by the fabric controller. The cluster cores have 2 different roles, which make them enter a different loop:

- The cluster controller core. It is core 8 on Gap9. Its role is to serve as the main controller of the cluster and thus, receives tasks directly from the fabric controller, and is then in charge of organizing the execution of these tasks over the whole cluster. After cluster power-up, it enters a loop where it is waiting for tasks sent by the fabric controller.

- Worker cores. They are cores 0 to 7 on gap9. Their role is to execute computation scheduled by the cluster controller core. For that, after cluster power-up, they enter a loop where they are waiting for work from it, like fork sections.

## Cluster stacks

Since the cluster is off when the chip is starting-up, the cores of the cluster do not need any stack at startup.

Once the cluster is powered-up and tasks are sent to it, the cluster cores must be allocated stacks. Default stacks can be allocated to them, or user can either allocate them or just give the desired stack size.

Depending on the role of the core, it will get a different stack, at a different moment:

- The cluster controller core must get a stack when the cluster is powered-up through the driver. This is because the cluster controller is the only core of the cluster which will keep its stack whatever the cluster task is executed, even with multiple tasks running.
- The worker cores will get a different stack for each cluster task, and so they will get it only when a cluster task is received.

## Cluster controller stack

The size of the cluster controller stack can be specified in cluster configuration:

```
struct pi_cluster_conf conf;
pi_cluster_conf_init(&conf);
conf.cc_stack_size = 0x2000;
pi_open_from_conf(&cluster_dev, (void *)&conf);
pi_cluster_open(&cluster_dev);
```

If no stack size is specified, a default size is chosen by the driver. The driver will allocate the stack during the driver open.

## Workers stacks

The size of each worker stack can be specified when a cluster task is setup:

```
pi_cluster_task_t task;
pi_cluster_task(&task, &entry, NULL);
pi_cluster_task_stacks(&task, NULL, 0x2000);
pi_cluster_send_task(&cluster_dev, &task);
```

The specified size is the size for the stack of one core. The driver will allocate one stack for each core. If no size is specified, the driver will take a default one. The driver will try to reuse the same stack to avoid allocating it everytime a task is sent. Thus, it will allocate it the first time a task is sent and will reallocate it if a task with a different stack size is sent.

To avoid that, the application can allocate the stacks itself and give them to the task:

```
pi_cluster_task_t task;
pi_cluster_task(&task, &entry, NULL);
pi_cluster_task_stacks(&task, my_stacks, 0x2000);
pi_cluster_send_task(&cluster_dev, &task);
```

Be careful that in this case, the allocated area must contain a stack for each core of the task, while the specified size if for the stack of one core.

---

**Note:** When they are allocated by the user, the stacks can be reused from one task to another since the tasks are executed sequentially on the cluster.

---

## Temporary memory

Since cluster tasks are executed sequentially, they don't always need exclusive memory areas. For example, the stacks of a task can easily be reused from one task to another. It's also true for some data buffers which are used only for temporary data during the task execution.

To ease managing such use-cases, a per-task allocator is provided, called the scratch allocator. This is a linear allocator, so buffers must be freed in reverse order they are allocated. Also, each task is having its own allocator. Such memory can be freely use during the execution of the task, but it will lose its content when the task is done, since it can be used by another task. For permanent data which should be kept untouched, the classic allocator can be used.

To use it, the application must first declare which part of the cluster memory is reserved for scratch allocators when the cluster is opened:

```
struct pi_cluster_conf conf;
pi_cluster_conf_init(&conf);
conf.scratch_size = 0x8000;
```

The specified size is removed from the cluster memory classic allocator, and is available to each cluster task.

Then memory from the scratch allocator can be allocated and freed using `pi_cl_11_scratch_alloc` and `pi_cl_11_scratch_free`.

## Temporary stacks

The scratch allocator is particularly interesting in order to have different sizes for stacks and temporary buffers between different tasks, depending on the specific needs of each task.

For that, stacks can be allocated within the scratch area, using the scratch allocator:

```
pi_cluster_task_t task;
pi_cluster_task(&task, &entry, NULL);
void *stacks = pi_cl_11_scratch_alloc(&cluster_dev, &task, STACK_SIZE*NB_WORKERS);
pi_cluster_task_stacks(&task, stacks, STACK_SIZE);
```

They do not need to be freed, they can be allocated only once and given several times to the same task.

## Cluster multi-tasking

By default, the cluster will execute the tasks in order until completion. This can cause some real-time issues if one task is too long, as it will prevent other small tasks from executing in time.

The cluster driver does not support task preemption, but supports a kind of cooperative scheduling where long tasks can regularly check if more important tasks need to execute. However, this is not done with context switching, to avoid allocating several tasks, and requires the current task to return from its entry point to let another task execute. Later on, the interrupted task will be resumed by entering again its entry point, and this will be up to the interrupted task to resume from where it stopped by keeping some kind of internal state.

For that, cluster task can first be assigned a priority:

```
pi_cluster_task_t task;
pi_cluster_task(&task, &entry, NULL);
pi_cluster_task_priority(&task, 1);
```

The default priority of a task is 0 and the driver only supports 2 priorities.

Tasks with a priority other than 0 must be enqueued with pi\_cluster\_enqueue\_task or pi\_cluster\_enqueue\_task\_async instead of pi\_cluster\_send\_task or pi\_cluster\_send\_task\_async. Be careful that pi\_cluster\_enqueue\_task and pi\_cluster\_enqueue\_task\_async do not support automatic stack allocation.

Then a low-priority task is supposed to call pi\_cl\_task\_yield from time to time to check if it must leave its entry point:

```
void cluster_entry(void *args)
{
    my_args_t *my_args = (my_args_t *)args;
    for (; my_args->index < 16; my_args->index++)
    {
        // Do some computation

        // Check if we should leave to let another task execute
        if (pi_cl_task_yield())
        {
            return;
        }
    }
}
```

Once there is no more high-priority tasks to execute, the low-priority task will be resumed. It is important that it continues from the iteration where it stopped.

#### 4.2.6 Dynamic Voltage and Frequency Scaling

For saving power and having the best power efficiency, GAP equipped with PMU (power management unit) and FLL (frequency locked loop) to allow you:

1. Power on/off some power domain.
2. Change the voltage of the SoC.
3. Change the frequency of each frequency domain.

The user can use them to adjust the voltage and frequency dynamically in runtime, to achieve the best power efficiency of their application.

## Power on/off

This has been implemented in the drivers, for example, the cluster domain can be powered on/off by using the API of opening and closing the cluster.

## Change the voltage

GAP9's voltage can be configured from 0.65V to 0.8V, which can be controlled dynamically by using api:

```
int pi_pmu_voltage_set(pi_pmu_voltage_domain_e domain, uint32_t voltage)
```

## Change the frequency

GAP9 has 4 frequency domains: SoC domain, Periph domain, Cluster domain and Cluster domain. Each domain's frequency can be configured dynamically from 0 to 370 MHz.

### Voltage and Frequency

With different voltage, the GAP9, different domains' maximum frequency will be limited as below:

- 0.65V : 240MHz
- 0.70V : TBC
- 0.75V : TBC
- 0.80V : 370MHz

Due to the voltage can limit the max frequency, when configuring the frequency and voltage, please follow these rules:

1. When increasing the frequency, increase the voltage first.
2. When decreasing the frequency, decrease the frequency first.

## 4.2.7 FLL Control and Configuration

GAP9 is equipped with a quadruple FLL, which is used for generating clocks for the SoC, Cluster, SFU and Periph Domains. It can generate the clock in open-loop mode or based on REF FAST clock (close-loop mode).

For more details about FLL, please read the section 4.2.4 FLL in datasheet.

## 4.2.8 BOOT IN OPEN LOOP MODE

If the REF FAST Clock is not presented, we can boot the chip in Open Loop Mode. To enable this mode in boot, you just need to add this in your Makefile:

```
APP_CFLAGS += -DCONFIG_OPEN_LOOP_BOOT
```

#### 4.2.9 OPEN/CLOSE LOOP Mode Switch

By using the following API, you could switch the FLL mode between open loop and close loop:

```
// Pass all the DCO to open loop
pi_fll_ioctl(PI_FREQ_DOMAIN_ALL, PI_FLL_IOCTL_FULL_DCO_OPEN_LOOP_SET, 0);

// Pass all the DCO to close loop
pi_fll_ioctl(PI_FREQ_DOMAIN_ALL, PI_FLL_IOCTL_FULL_DCO_CLOSE_LOOP_SET, 0);
```

BEWARE, when the DCOs are running in open loop mode, some features based on the REF FAST Clock may need to change its clock source as well. For example, the Timers. All the timers used by the runtime (system tick and high precision timer) should be re-configured accordingly.

```
if (op_mode) //in close loop, change to open loop
{
    printf("Pass to open loop\n");
    pi_timer_ioctl(NULL, PI_TIMER_IOCTL_SYS_TIMER_SRC_SET, (void *) PI_TIMER_SRC_FLL);
    pi_fll_ioctl(PI_FREQ_DOMAIN_ALL, PI_FLL_IOCTL_FULL_DCO_OPEN_LOOP_SET, 0);
    op_mode = 0;
}
else // in open loop, change to close loop
{
    printf("Pass to close loop, need clk\n");
    pi_fll_ioctl(PI_FREQ_DOMAIN_ALL, PI_FLL_IOCTL_FULL_DCO_CLOSE_LOOP_SET, 0);
    pi_timer_ioctl(NULL, PI_TIMER_IOCTL_SYS_TIMER_SRC_SET, (void *) PI_TIMER_SRC_REF_CLK_
FAST);
    op_mode = 1;
}
```

**EXAMPLES**

Examples for GAP series processors.

## 5.1 GAP9

### 5.1.1 Basic

These examples show how to use GAP9 basic features, such as using interfaces, external devices, cluster, etc.

#### Hello World

##### Requirements

No specific requirement. This example should run without issue on all chips/boards/OSes.

##### Description

This example is a classic Hello World. It prints an hello world string on all available cores.

You can run this on GVSOC or your board:

```
# Run on GVSOC
make clean all run platform=gvsoc

# Run on real board
make clean all run platform=board
```

You should have an output looking like this (order may vary):

```
*** PMSIS HelloWorld ***

Entering main controller
[32 0] Hello World!
Cluster master core entry
[0 2] Hello World!
[0 0] Hello World!
[0 1] Hello World!
[0 3] Hello World!
```

(continues on next page)

(continued from previous page)

```
[0 4] Hello World!
[0 5] Hello World!
[0 6] Hello World!
[0 7] Hello World!
Cluster master core exit
Test success !
```

**Code**

C

```
/* PMSIS includes */
#include "pmsis.h"

/* Task executed by cluster cores. */
void cluster_helloworld(void *arg)
{
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    printf("[%d %d] Hello World!\n", cluster_id, core_id);
}

/* Cluster main entry, executed by core 0. */
void cluster_delegate(void *arg)
{
    printf("Cluster master core entry\n");
    /* Task dispatch to cluster cores. */
    pi_cl_team_fork(pi_cl_cluster_nb_cores(), cluster_helloworld, arg);
    printf("Cluster master core exit\n");
}

/* Program Entry. */
int main(void)
{
    printf("\n\n\t *** PMSIS HelloWorld ***\n\n");
    printf("Entering main controller\n");

    uint32_t errors = 0;
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    printf("[%d %d] Hello World!\n", cluster_id, core_id);

    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;

    /* Init cluster configuration structure. */
    pi_cluster_conf_init(&cl_conf);
    cl_conf.id = 0; /* Set cluster ID. */
                    // Enable the special icache for the master core
    cl_conf.icache_conf = PI_CLUSTER_MASTER_CORE_ICACHE_ENABLE |
                        // Enable the prefetch for all the cores, it's a 9bits mask (from
                        // bit 2 to bit 10), each bit correspond to 1 core
                        PI_CLUSTER_ICACHE_PREFETCH_ENABLE |
```

(continues on next page)

(continued from previous page)

```

    // Enable the icache for all the cores
    PI_CLUSTER_ICACHE_ENABLE;

    /* Configure & open cluster. */
    pi_open_from_conf(&cluster_dev, &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        printf("Cluster open failed !\n");
        pmsis_exit(-1);
    }

    /* Prepare cluster task and send it to cluster. */
    struct pi_cluster_task cl_task;

    pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_delegate,
    ↪NULL));

    pi_cluster_close(&cluster_dev);

    printf("Test success !\n");

    return errors;
}

```

Makefile

```

# User Test
-----
APP          = test
# App sources
APP_SRCS      = helloworld.c
# App includes
APP_INC       =
# Compiler flags
APP_CFLAGS    =
# Linker flags
APP_LDFLAGS   =
# Custom linker
APP_LINK_SCRIPT =

include $(RULES_DIR)/pmsis_rules.mk

```

## Helloworld C++

### Requirements

No specific requirement.

### Description

These example shows how to use C++ on GAP9.

### Code

C++

```
/* PMSIS includes */
#include "pmsis.h"
#include "math.h"

/* Task executed by cluster cores. */
void cluster_helloworld(void *arg)
{
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    printf("[%d %d] Hello World!\n", cluster_id, core_id);
}

/* Cluster main entry, executed by core 0. */
void cluster_delegate(void *arg)
{
    printf("Cluster master core entry\n");
    /* Task dispatch to cluster cores. */
    pi_cl_team_fork(pi_cl_cluster_nb_cores(), cluster_helloworld, arg);
    printf("Cluster master core exit\n");
}

/* Program Entry. */
int main(void)
{
    printf("\n\n\t **** PMSIS HelloWorld ***\n\n");
    printf("Entering main controller\n");

    uint32_t errors = 0;
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    printf("[%d %d] Hello World!\n", cluster_id, core_id);

    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;

    /* Init cluster configuration structure. */
    pi_cluster_conf_init(&cl_conf);
    cl_conf.id = 0; /* Set cluster ID. */
    /* Configure & open cluster. */
```

(continues on next page)

(continued from previous page)

```

pi_open_from_conf(&cluster_dev, &cl_conf);
if (pi_cluster_open(&cluster_dev))
{
    printf("Cluster open failed !\n");
    pmsis_exit(-1);
}

/* Prepare cluster task and send it to cluster. */
struct pi_cluster_task cl_task;

pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_delegate,NULL));

pi_cluster_close(&cluster_dev);

printf("Test success !\n");

pmsis_exit(errors);
}

```

### Makefile

```

# User Test
#-----
APP      = test
# App sources
APP_SRCS_CXX = helloworld.cpp
# App includes
APP_INC      =
# Compiler flags
APP_CFLAGS   =
# Linker flags
APP_LDFLAGS  =
# Custom linker
APP_LINK_SCRIPT  =
include $(RULES_DIR)/pmsis_rules.mk

```

### BSP

These examples show how to use various external devices with GAP9.

### AK4332 Examples

These examples show some uses case about the AK4332 driver.

#### AK4332 PCM example

##### Description

This example use the audio addon AK4332 device to generate a sine wave.

Here, the AK4332 is set to receive PCM data in MSB justified mode.

This example shows :

- How to configure the I2S audio stream to continuously send PCM data from a header file.
- How to configure the AK4332 and how to control it through its API.

How to run the Example

CMake

This example is compatible with KConfig options system. Here, the example's KConfig file impose to select the following options :

- Platform : Board
- Board : GAP9\_EVK
- Board option : Audio Addon
- BSP dependencies : AK4332

**Warning:** Be sure that PMSIS\_OS environnement variable has not been set to “pulpos”. Otherwise, the compilation won't work

First, make sure you sourced the SDK with the sourceme.sh script and selected the GAP9 EVK mode.

Then, configure CMake with the following command:

```
cmake -B build
```

Build the program and run it with the following command:

```
cmake --build build --target run
```

Makefile

This example can be run with Make in the following configuration :

- Platform : Board
- Hardware : GAP9\_EVK + Audio addon
- Board option : Audio addon
- PMSIS\_OS : Freertos or PulpOS

First, make sure you sourced the SDK with the sourceme.sh script and selected the GAP9 EVK mode.

Then, build the example with the following command :

```
make all run platform=board PMSIS_OS=freertos
```

or

```
make all run platform=board PMSIS_OS=pulpos
```

AK4332 driver

Please find a full documentation about the AK4332 driver here : [DAC](#)

AK4332 driver consists in several configuration level.

## 1. BSP Configuration

The AK4332 is configured from the BSP. According to the way the audio addon v1.1 is (2 AK4332 driven from a single I2C bus, both sharing the same slave address) its BSP provide an API to select a mode selection between **mono left**, **mono right** and **stereo** configurations.

The selected mode will affect the way AK4332(s) is(are) driven.

The configuration define which I2C and I2S interface to use but also a set a function pointer that represent the driver **Dynamic API**. This API changes depending on the previously selected mode.

Here is the list of function provided by the Dynamic API :

- **Enable** : Electrically enable the device
- **Init** : Load a configuration and program AK4332's registers
- **Start** : initiate the powerup sequence of the device
- **Set volume** : Modify the volume of the device (also possible on-the-fly)
- **Stop** : Initiate the power down sequence of the device.
- **Disable** : Electrically disable the device. **Warning, this function resets the device's registers**

## 2. Initialization structure

The initialization structure is the user view of all AK4332's settings.

There are organized according to AK4332's major block :

- Clock
- Serial audio interface
- Digital to Analog converter
- HeadPhone

This structure can be fully set by the user and loaded from a preset available in the driver.

### 3. Configuration register structure

The configuration register structure is a direct representation of AK4332 registers except those responsible of the power management.

This structure is meant to be populated by the initialization structure, not by the user. Then it will be used to write AK4332 registers in the right way.

AK4332 Design rules

*PLL configuration examples can be found in AK4332 Datasheet p.43 and p.44.*

#### 1 - REFCLK

Must be set between **76.8 kHz** and **768kHz**.

This clock depends on the input clock source value and **PLD** register value according to the following formula :

$$\text{REFCLK} = \text{Source Clock} / (\text{PLD} + 1)$$

The input clock source can be **MCLK** or **BCLK**. This setting is handled by the **src** member of the initialization structure : **clk pll src**

Set PLD value to adjust REFCLK in the right range.

PLD value can be set in the following member of the initialization structure : **clk pll pld**

*Example:*

For an input clock source value of 3,072 MHz, PLD must be set **between 3 and 39**

- if PLD = 3, REFCLK = 768kHz
- if PLD = 39, REFCLK = 76.8kHz

#### 2 - PLLCLK

PLL Clock is set from the clock source value and registers value such as **PLD** and **PLM** According to this formula :

$$\text{PLLCLK} = \text{REFCLK} * (\text{PLM} + 1) \text{ or } \text{PLLCLK} = \text{Source Clock} * (\text{PLM} + 1) / (\text{PLD} + 1)$$

It must be set to **24,5760 MHz** (in case of 48kHz or 32kHz base rate) or **22.5792MHz** (in case of 44.1 kHz base rate). Adjust PLM (and PLD) value to match the right value.

*Example:*

For an input clock source value of 3,072 MHz and a sampling frequency of 48 kHz, a valid configuration can be :

PLD = 3 & PLM = 31

The PLLCLK value is then **3072000 \* 32 / 4 = 2,45760 MHz**

### 3 - DACMCLK

DAC's master clock. This clock depends on :

- The source clock, set by **clk.dac\_clksrc** member of the initialization structure (MCKI or PLLCLK)
- If **dac\_clksrc** = PLLCLK, this clock can be divided by a prescaler : **MDIV** which is set by **clk pll.mdiv** member of the initialization structure.

*Example:*

According to previous examples, if PLL source is selected and MDIV set to 0, **DACMCLK = PLLCLK = 2,45760 MHz**

**Master clock configuration CM bits from CMS registers.**

These bits must be set according to **DACMCLK / FS** result

*Example:*

If **DACMCLK = 24,576MHz** and **FS = 48kHz**, CM bits must be set to **CM\_512\_FS**.

CM bits are represented by **clk.cm** member of the initialization structure

### 4 - PLLMD

Must be set according to **REFCLK** value.

- If REFCLK  $\geq$  256kHz -> **clk pll.mode** must be set to **PLLMD\_HIGH\_F\_MODE**
- If REFCLK  $<$  256kHz -> **clk pll.mode** must be set to **PLLMD\_HIGH\_L\_MODE**

AK4332 Example configuration

#### 1 - Input Settings

- Sampling Frequency : 48kHz
- WordSize : 32bits

- Channel number : 2
- I2S Clock :  $48000 * 32 * 2 = 3,072 \text{ MHz}$

## 2 - AK4332 Configuration

This example is based on a preset named “AK4332\_PCM\_SLAVE\_PLL\_BCLK\_32WS\_48KHZ” Here are its settings:

- **Clock Source** : BCLK
- **Sampling frequency** : 48 kHz
- **Master clock setting (CM)** : CM\_512\_FS
- **WordSize** : 32bits
- **Clock source** : Sampling Frequency \* Channel number \* Word size = 3,072 MHz
- **PLL Status** : Enabled

*See AK4332 Design rules tabs for more details about how to set the following parameters.*

## 3 - PLL Settings

- PLD = 3 -> REFCLK = 768kHz
- PLM = 31 -> PLLCLK = 24,5760 MHz
- PLLMD = 0 (REFCLK > 256kHz)
- MDIV = 0 -> DACMCLK = PLLCLK = 24,576 MHz = CM \* FS (512 \* 48000)

*See AK4332 Datasheet p.44 for more details.*

## Code

### Example

```
#####
//#####
/***
* @file      ak4332_pcm.c
* @author    Mickael Cottin-Bizonne, GreenWaves Technologies,
*            (mickael.cottin.bizonne@greenwaves-technologies.com)
* @version   1
* @date     2022-07-04
*
* @copyright Copyright (C) 2022 GreenWaves Technologies
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

(continues on next page)

(continued from previous page)

```

* @brief      This example show how to use the AK4332 device in the following
configuration :
    - Mode          : Slave
    - Data type     : PCM
    - SAI format    : MSB justified
    - Clock source  : Bit clock
    - Sampling frequency : 48 kHz
    - Word size     : 32 bits
    - PLL           : Enabled

Once configure, the driver :
    - Power up the device.
    - Play the sinus defined in ak4332_sine.h at a minimum volume level
    - Linearly update the volume until the maximum level
    - Stop the device for a while
    - Repeat the 2 previous steps then shutdown the device
*/
////////////////////////////////////////////////////////////////////////#
# #####
// Includes ****
#include <stdio.h>
#include "pmsis.h"
#include "bsp/bsp.h"
#include "ak4332_sine.h"

#define CONFIG_AK4332_WORD_SIZE      32
#define CONFIG_AK4332_NB_CHANNELS    2
#define CONFIG_AK4332_SAMPLING_RATE 48000
#define CONFIG_AK4332_WS_DELAY       0
#define CONFIG_AK4332_WS_TYPE        1
#define CONFIG_AK4332_NB_BUFFERS    16

#define APP_VOLUME_DELAY            200000
#define APP_VOLUME_MIN              0
#define APP_VOLUME_MAX              100
#define APP_VOLUME_STEP             10

static int nb_buffers;
static pi_evt_t end_task;
static pi_evt_t channel_tasks[CONFIG_AK4332_NB_CHANNELS][2];
pi_device_t i2s;

static void handle_buffer_end(void *arg)
{
    int arg_i = (int)arg;
    int channel = arg_i >> 8;
    int iter = arg_i & 0xff;
    pi_i2s_channel_write_async(&i2s, channel, &ak4332_sine, AK4332_SINE_SIZE_BYTES, pi_
    ↵evt_callback_no_irq_init(&channel_tasks[channel][iter], handle_buffer_end, (void_
    ↵*)((channel << 8) + iter)));
}

```

(continues on next page)

(continued from previous page)

```

}

int test_entry()
{
    int retval = 0;
    pi_device_t ak4332_dev;
    struct pi_i2s_conf i2s_conf;
    pi_ak4332_conf_t* ak4332_conf = (pi_ak4332_conf_t*)bsp_ak4332_get_conf_from_mode(BSP_
    ↵AK4332_STEREO);

    pi_open_from_conf(&ak4332_dev, ak4332_conf);
    if(pi_ak4332_open(&ak4332_dev))
    {
        retval = -1;
    }
    else
    {
        pi_i2s_conf_init(&i2s_conf);

        i2s_conf.frame_clk_freq = CONFIG_AK4332_SAMPLING_RATE;
        i2s_conf.itf           = ak4332_conf->i2s_itf;
        i2s_conf.word_size     = CONFIG_AK4332_WORD_SIZE;
        i2s_conf.channels      = CONFIG_AK4332_NB_CHANNELS;
        i2s_conf.options       = PI_I2S_OPT_TDM | PI_I2S_OPT_FULL_DUPLEX;
        i2s_conf.ws_delay      = CONFIG_AK4332_WS_DELAY;
        i2s_conf.ws_type       = CONFIG_AK4332_WS_TYPE;

        pi_open_from_conf(&i2s, &i2s_conf);
        if (pi_i2s_open(&i2s))
        {
            retval = -1;
        }
        else
        {
            for (int channel = 0; channel < CONFIG_AK4332_NB_CHANNELS; channel++)
            {
                struct pi_i2s_channel_conf i2s_channel_conf;

                pi_i2s_channel_conf_init(&i2s_channel_conf);

                // Enabled RX for slot i with pingpong buffers
                i2s_channel_conf.options     = PI_I2S_OPT_IS_RX | PI_I2S_OPT_ENABLED | PI_
                ↵I2S_OPT_PINGPONG;
                i2s_channel_conf.word_size  = CONFIG_AK4332_WORD_SIZE;
                i2s_channel_conf.block_size = AK4332_SINE_SIZE_BYTES;
                i2s_channel_conf.pingpong_buffers[0] = &ak4332_sine;
                i2s_channel_conf.pingpong_buffers[1] = &ak4332_sine;
                pi_i2s_channel_conf_set(&i2s, channel, &i2s_channel_conf);
            }

            nb_buffers = CONFIG_AK4332_NB_BUFFERS;
            pi_evt_sig_init(&end_task);
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        for (int channel = 0; channel < CONFIG_AK4332_NB_CHANNELS; channel++)
        {
            pi_i2s_channel_write_async(&i2s, channel, &ak4332_sine, AK4332_SINE_SIZE_
→BYTES, pi_evt_callback_no_irq_init(&channel_tasks[channel][0], handle_buffer_end,_
→(void *)((channel << 8) | 0)));
            pi_i2s_channel_write_async(&i2s, channel, &ak4332_sine, AK4332_SINE_SIZE_
→BYTES, pi_evt_callback_no_irq_init(&channel_tasks[channel][1], handle_buffer_end,_
→(void *)((channel << 8) | 1)));
        }

        //Select a preset (dedicated to this example) into the AK4332 device
        ak4332_init_struct_t akinit = pi_ak4332_load_init_preset(AK4332_PCM_SLAVE_
→PLL_BCLK_32WS_48KHZ);

        // Electricaly enable the AK4332. This step must be done before initializing_
→and starting the device
        pi_ak4332_enable(&ak4332_dev);

        // Load AK4332 preset and program configuration registers
        pi_ak4332_init(&ak4332_dev, akinit);

        // Start the device
        pi_ak4332_start(&ak4332_dev);

        // Start I2S transmission
        pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL);

        // Update the volume as a ramp up
        for(uint8_t vol = APP_VOLUME_MIN; vol <= APP_VOLUME_MAX; vol += APP_VOLUME_
→STEP)
        {
            pi_ak4332_set_volume(&ak4332_dev, vol);
            pi_time_wait_us(APP_VOLUME_DELAY);
        }

        // Stop the device (pause)
        pi_ak4332_stop(&ak4332_dev);

        // Stop I2S transmission
        pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_STOP, NULL);

        // Stop I2S transmission
        pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_STOP, NULL);

        // Electricaly disable the AK4332 -> This resets AK4332 registers.
        pi_ak4332_disable(&ak4332_dev);

        // Free the device
        pi_ak4332_close(&ak4332_dev);
    }
}

```

(continues on next page)

(continued from previous page)

```

    return retval;
}

int main(void)
{
    int retval = test_entry();
    pmsis_exit(retval);
    return 0;
}

```

CMakeLists.txt

```

# Copyright (c) 2022 GreenWaves Technologies SAS
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# 1. Redistributions of source code must retain the above copyright notice,
#    this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. Neither the name of GreenWaves Technologies SAS nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

cmake_minimum_required(VERSION 3.16)

#####
# Panel Control
#####
set(TARGET_NAME "ak4332_pcm")
set(TARGET_SRCS ak4332_pcm.c)

#####
# CMake pre initialization
#####
set(CONFIG_GAP_SDK_HOME $ENV{GAP_SDK_HOME})

```

(continues on next page)

(continued from previous page)

```

include($ENV{GAP_SDK_HOME}/utils/cmake/setup.cmake)
setup_cmake()

project(${TARGET_NAME} C ASM)
add_executable(${TARGET_NAME} ${TARGET_SRCS})

#####
# App's options interpretation
#####

if(NOT DEFINED CONFIG_BOARD_GAP9_EVK)
    message(WARNING "Please select BOARD_GAP9_EVK option to make the example works")
endif()

#####
# CMake post initialization
#####
setupos(${TARGET_NAME})

```

Makefile

```

APP = ak4332_pcm
APP_SRCS = ak4332_pcm.c
APP_CFLAGS += -Os -g

CONFIG_AK4332=1

include $(RULES_DIR)/pmsis_rules.mk

```

Source signal

```

#####
// This file defines a sine wave coded on 32 bits.
// N Samples : 768
// Period      : 48 samples
// Amplitude   : Half of signed integer range
// N period    : 16
//
// Note:      Table size is available at the end of the file with
//             AK4332_SINE_SIZE and AK4332_SINE_SIZE_BYTES macros.
#####

int32_t ak4332_sine[] = {
0,
140151432,
277904834,
410903207,
536870912,
653652607,
759250125,
851856663,

```

(continues on next page)

(continued from previous page)

929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,

(continues on next page)

(continued from previous page)

1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,

(continues on next page)

(continued from previous page)

```
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,
```

(continues on next page)

(continued from previous page)

536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,

(continues on next page)

(continued from previous page)

```
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,
```

(continues on next page)

(continued from previous page)

-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,

(continues on next page)

(continued from previous page)

```
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,
```

(continues on next page)

(continued from previous page)

```
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,
```

(continues on next page)

(continued from previous page)

```
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,
```

(continues on next page)

(continued from previous page)

```
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,
```

(continues on next page)

(continued from previous page)

```
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,
```

(continues on next page)

(continued from previous page)

536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,

(continues on next page)

(continued from previous page)

929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,

(continues on next page)

(continued from previous page)

1073741824,  
1064555814,  
1037154958,  
992008094,  
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
0,  
140151432,  
277904834,  
410903207,  
536870912,  
653652607,  
759250125,  
851856663,  
929887697,  
992008094,  
1037154958,  
1064555814,  
1073741824,  
1064555814,  
1037154958,  
992008094,

(continues on next page)

(continued from previous page)

```
929887697,  
851856663,  
759250125,  
653652607,  
536870912,  
410903207,  
277904834,  
140151432,  
0,  
-140151432,  
-277904834,  
-410903207,  
-536870912,  
-653652607,  
-759250125,  
-851856663,  
-929887697,  
-992008094,  
-1037154958,  
-1064555814,  
-1073741824,  
-1064555814,  
-1037154958,  
-992008094,  
-929887697,  
-851856663,  
-759250125,  
-653652607,  
-536870912,  
-410903207,  
-277904834,  
-140151432,  
};  
  
#define AK4332_SINE_SIZE (sizeof(ak4332_sine)/sizeof(ak4332_sine[0]))  
#define AK4332_SINE_SIZE_BYTES (sizeof(ak4332_sine))
```

## AK4332 PDM example

### Description

This example use the audio addon AK4332 device to generate a sine wave.

Here, the AK4332 is set to receive PDM Direct stream digital data.

This example shows :

- How to configure a SFU Graph to produce the PDM signal.
- How to configure the I2S audio stream to continuously send PDM data from a header file.
- How to configure the AK4332 and how to control it through its API.

How to run the Example

## CMake

This example is compatible with KConfig options system. Here, the example's KConfig file impose to select the following options :

- Platform : Board
- Board : GAP9\_EVK
- Board option : Audio Addon
- BSP dependencies : AK4332

**Warning:** Be sure that PMSIS\_OS environnement variable has not been set to “pulpos”. Otherwise, the compilation won't work

First, make sure you sourced the SDK with the sourceme.sh script and selected the GAP9 EVK mode.

Then, configure CMake with the following command:

```
cmake -B build
```

This command will also build the SFU Graph which is mandatory for this example.

The example's CMakefile build it at the beginning.

```
execute_process(COMMAND SFU -i ${CMAKE_SOURCE_DIR}/Graph.src -C
    WORKING_DIRECTORY ${CMAKE_BINARY_DIR})
```

Build the program and run it with the following command:

```
cmake --build build --target run
```

## Makefile

This example can be run with Make in the following configuration :

- Platform : Board
- Hardware : GAP9\_EVK + Audio addon
- Board option : Audio addon
- PMSIS\_OS : Freertos or PulpOS

First, make sure you sourced the SDK with the sourceme.sh script and selected the GAP9 EVK mode.

Then, configure your build system by setting the following environment variables

```
export PMSIS_OS=freertos # or pulpos
export PMSIS_PLATFORM=board
```

Then, build the SFU graph with the following target :

```
make graph
```

Then, build and run the example with the following command :

```
make all run
```

AK4332 driver

Please find a full documentation about the AK4332 driver here : [DAC](#)

AK4332 driver consists in several configuration level.

### 1. BSP Configuration

The AK4332 is configured from the BSP. According to the way the audio addon v1.1 is (2 AK4332 driven from a single I2C bus, both sharing the same slave address) its BSP provide an API to select a mode selection between **mono left**, **mono right** and **stereo** configurations.

The selected mode will affect the way AK4332(s) is(are) driven.

The configuration define which I2C and I2S interface to use but also a set a function pointer that represent the driver **Dynamic API**. This API changes depending on the previously selected mode.

Here is the list of function provided by the Dynamic API :

- **Enable** : Electrically enable the device
- **Init** : Load a configuration and program AK4332's registers
- **Start** : initiate the powerup sequence of the device
- **Set volume** : Modify the volume of the device (also possible on-the-fly)
- **Stop** : Initiate the power down sequence of the device.
- **Disable** : Electrically disable the device. **Warning, this function resets the device's registers**

### 2. Initialization structure

The initialization structure is the user view of all AK4332's settings.

There are organized according to AK4332's major block :

- Clock
- Serial audio interface
- Digital to Analog converter
- HeadPhone

This structure can be fully set by the user and loaded from a preset available in the driver.

### 3. Configuration register structure

The configuration register structure is a direct representation of AK4332 registers except those responsible of the power management.

This structure is meant to be populated by the initialization structure, not by the user. Then it will be used to write AK4332 registers in the right way.

Example constraints

#### 1 - SFU Constraints

The PDM\_OUT component from the SFU can only accept PDM data coded on 24bits.

Sending higher values will cause the PDM\_OUT component to crash. It will produce noise.

The sine wave encoded in ak4332\_sine24.h is defined according to this constraint.

In this mode, the I2S clock value is not set according to the following formula : **nChannel \* WordSize \* SamplingFrequency**.

It depends on the Upsampling parameter of the PDM\_OUT node which is set in the graph description.

In this example, this parameter is set to **4**. According to the SFU Documentation, the Upsampling coefficient is then **64**.

The right I2S clock is calculated according to this formula : **Upsampling factor \* Sampling Frequency**

#### 2 - AK4332 PDM requirements

In PDM configuration, the AK4332 must configure its Sampling frequency to **44100 kHz or 48000 KHz**.

It must also set its master clock configuration setting to **256 \* Fs**.

This last constraint will affect the way **MDIV** register is set. See **AK4332 Configuration** section below for more explanations

SFU Configuration

The SFU is set to send data from a memory (MEM\_IN) to a component (PDM\_OUT).

During its initialization phase, 2 sets of data are enqueued for double buffering purpose. Afterward, while one set is played, another one is loaded.

A callback is set to enqueue the data again until a max number of iteration is reached.

During this enqueueing phase, the AK4332 is powered up as the I2S flow to play data coming from PDM\_OUT component.

The application yield until the maximum number of data to be enqueued is reached  
Once reached, the application wait for the last transfer to be completed.  
Then, every resources are shutted down.

AK4332 Design rules

*PLL configuration examples can be found in AK4332 Datasheet p.43 and p.44.*

### 1 - REFCLK

Must be set between **76.8 kHz** and **768kHz**.

This clock depends on the input clock source value and **PLD** register value according to the following formula :

$$\text{REFCLK} = \text{Source Clock} / (\text{PLD} + 1)$$

The input clock source can be **MCLK** or **BCLK**. This setting is handled by the **src** member of the initialization structure : **clk pll src**

Set PLD value to adjust REFCLK in the right range.

PLD value can be set in the following member of the initialization structure : **clk pll pld**

*Example:*

For an input clock source value of 3,072 MHz, PLD must be set **between 3 and 39**

- if PLD = 3, REFCLK = 768kHz
- if PLD = 39, REFCLK = 76,8kHz

### 2 - PLLCLK

PLL Clock is set from the clock source value and registers value such as **PLD** and **PLM** According to this formula :

$$\text{PLLCLK} = \text{REFCLK} * (\text{PLM} + 1) \text{ or } \text{PLLCLK} = \text{Source Clock} * (\text{PLM} + 1) / (\text{PLD} + 1)$$

It must be set to **24,5760 MHz** (in case of 48kHz or 32kHz base rate) or **22.5792MHz** (in case of 44.1 kHz base rate).  
Adjust PLM (and PLD) value to match the right value.

*Example:*

For an input clock source value of 3,072 MHz and a sampling frequency of 48 kHz, a valid configuration can be :

PLD = 3 & PLM = 31

The PLLCLK value is then **3072000 \* 32 / 4 = 2,45760 MHz**

### 3 - DACMCLK

DAC's master clock. This clock depends on :

- The source clock, set by `clk.dac_clksrc` member of the initialization structure (MCKI or PLLCLK)
- If `dac_clksrc` = PLLCLK, this clock can be divided by a prescaler : **MDIV** which is set by `clk pll.mdiv` member of the initialization structure.

*Example:*

According to previous examples, if PLL source is selected and MDIV set to 0, **DACMCLK = PLLCLK = 2,45760 MHz**

#### Master clock configuration CM bits from CMS registers.

These bits must be set according to **DACMCLK / FS** result

*Example:*

If **DACMCLK = 24,576MHz** and **FS = 48kHz**, CM bits must be set to **CM\_512\_FS**.

CM bits are represented by `clk.cm` member of the initialization structure

**Warning: According to the AK4332 documentation, in PDM mode, CM bits must be set to CM\_256\_FS**

### 4 - PLLMD

Must be set according to **REFCLK** value.

- If REFCLK >= 256kHz -> `clk pll.mode` must be set to **PLLMD\_HIGH\_F\_MODE**
- If REFCLK < 256kHz -> `clk pll.mode` must be set to **PLLMD\_HIGH\_L\_MODE**

AK4332 Example configuration

#### 1 - Input Settings

- Sampling Frequency : 48kHz
- WordSize : 24bits (*PDM\_OUT constraint*)
- Upsampling factor : 64
- I2S Clock :  $48000 * 64 = 3,072 \text{ MHz}$

#### 2 - AK4332 Configuration

This example is based on a preset named “AK4332\_PDM\_DSD\_SLAVE\_PLL\_BCLK\_32WS\_48KHZ” Here are its settings:

- **Clock Source** : BCLK
- **Sampling frequency** : 48 kHz
- **Master clock setting (CM)** : CM\_256\_FS (*Mandatory in PDM mode*)

- **WordSize** : 32bits -> *But the data will remains coded on 24bits*
- **Clock source** : Sampling Frequency \* Upsampling factor = 3,072 MHz
- **PLL Status** : Enabled

*See AK4332 Design rules tabs for more details about how to set the following parameters.*

### 3 - PLL Settings

- PLD = 3 -> REFCLK = 768kHz
- PLM = 31 -> PLLCLK = 24,5760 MHz
- PLLMD = 0 (REFCLK > 256kHz)
- MDIV = 1 -> DACMCLK = PLLCLK / 2 = 12,288 MHz = CM \* FS (256 \* 48000)

*See AK4332 Datasheet p.44 for more details.*

### Code

#### Example

```
//#####
//#####
/***
* @file      ak4332_pdm.c
* @author    Mickael Cottin-Bizonne, GreenWaves Technologies,
*            (mickael.cottin.bizonne@greenwaves-technologies.com)
* @version   1
* @date     2022-07-06
*
* @copyright Copyright (C) 2022 GreenWaves Technologies
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*
* @brief      This example show how to use the AK4332 device in the following
* configuration :
*             - Mode          : Slave
*             - Data type     : PDM
*             - SAI format    : I2S
*             - Clock source  : Bit clock
*             - Sampling frequency : 48 kHz
*             - Word size     : 32 bits --> See following note
*             - PLL           : Enabled
*
* @note       This example uses the SFU to generate a PDM signal. The input signal
* must be coded
```

(continues on next page)

(continued from previous page)

```

on 24 bits otherwise, the PDM_OUT component from the SFU will crash. To
be compatible
with this constraint, the AK4332 is set to receive 32bits data.

Once configure, the driver :
- Power up the device.
- Play the sinus defined in ak4332_sine24.h at a minimum volume level
- Linearly update the volume until the maximum level
- Stop the device
*/
//#####
//#####

```

---

```

// Includes
*****  

#include <stdio.h>
#include "pmsis.h"
#include "bsp/bsp.h"
#include "ak4332_sine24.h"

#include "Graph_L2_Descr.h"
#include "SFU_RT.h"

// Defines
*****  

// The upsampling factor is defined in the Graph.Src file. See SFU Documentation for
more details
#define CONFIG_SFU_UPSAMPLING_FACTOR      (4 << 4)
#define CONFIG_AK4332_SAMPLING_RATE        48000

#define CONFIG_AK4332_I2S_CLOCK           (CONFIG_AK4332_SAMPLING_RATE * CONFIG_SFU_
UPSAMPLING_FACTOR)
#define CONFIG_AK4332_PDM_POLARITY        0
#define CONFIG_AK4332_PDM_DIFF_MODE       0

#define APP_VOLUME_DELAY                 200000
#define APP_VOLUME_MIN                  0
#define APP_VOLUME_MAX                  100
#define APP_VOLUME_STEP                 10
#define APP_N_DATA_TO_ENQUEUE           200

pi_device_t i2s;
SFU_uDMA_Channel_T *ChanInCtxt;
volatile int enqueue_cnt = 0;
volatile int enqueue_done = 0;

/**
 * @brief Handle the enqueueing process.
 *         Enqueue data until the desired number is reached.

```

(continues on next page)

(continued from previous page)

```

/*
 * @param arg Unused. Only match the prototype.
 */
static void handle_sfu_end(void* arg)
{
    if(enqueue_cnt < APP_N_DATA_TO_ENQUEUE)
    {
        SFU_Enqueue_uDMA_Channel(ChanInCtxt, &ak4332_sine24, AK4332_SINE24_SIZE_BYTES);
        enqueue_cnt++;
    }
    else
    {
        enqueue_done = 1;
    }
}

// ****
// *****

int test_entry()
{
    int retval = 0;

    pi_device_t ak4332_dev;
    struct pi_i2s_conf i2s_conf;
    pi_ak4332_conf_t* ak4332_conf = (pi_ak4332_conf_t*)bsp_ak4332_get_conf_from_mode(BSP_
AK4332_STEREO);

    pi_open_from_conf(&ak4332_dev, ak4332_conf);

    if(pi_ak4332_open(&ak4332_dev))
    {
        retval = -1;
    }
    else
    {
        pi_i2s_conf_init(&i2s_conf);
        i2s_conf.frame_clk_freq = CONFIG_AK4332_I2S_CLOCK;
        i2s_conf.itf = ak4332_conf->i2s_itf;
        i2s_conf.format |= PI_I2S_FMT_DATA_FORMAT_PDM;
        i2s_conf.pdm_polarity = CONFIG_AK4332_PDM_POLARITY;
        i2s_conf.pdm_diff = CONFIG_AK4332_PDM_DIFF_MODE;

        pi_open_from_conf(&i2s, &i2s_conf);
        if (pi_i2s_open(&i2s))
        {
            retval = -1;
        }
        else
        {
            // SFU Configuration
            StartSFU(0, 1);
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

ChanInCtxt = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_
→T));
    // Get in and out uDMA channels
SFU_Allocate_uDMA_Channel(ChanInCtxt, 0, &SFU_RTD(Graph));
SFU_uDMA_Channel_Callback(ChanInCtxt, handle_sfu_end, NULL);

/*
    WARNING PDM OUT component can only accept an input signal coded on 24_
→bits max.
    Enqueuing 2 times for double buffering.
    Afterward, while one set a data is played, another one is loaded.
*/
SFU_Enqueue_uDMA_Channel(ChanInCtxt,
                        &ak4332_sine24,
                        AK4332_SINE24_SIZE_BYTES);
SFU_Enqueue_uDMA_Channel(ChanInCtxt,
                        &ak4332_sine24,
                        AK4332_SINE24_SIZE_BYTES);

// Connect Channels to SFU
SFU_GraphConnectIO(SFU_Name(Graph), In1), ChanInCtxt->ChannelId, 0, &SFU_
→RTD(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), Out1), ak4332_conf->i2s_itf, 1, &SFU_
→RTD(Graph));

// Select a preset (dedicated to this example) into the AK4332 device
ak4332_init_struct_t akinit = pi_ak4332_load_init_preset(AK4332_PDM_DSD_
→SLAVE_PLL_BCLK_32WS_48KHZ);

// Electricaly enable the AK4332. This step must be done before starting the_
→device
pi_ak4332_enable(&ak4332_dev);

// Load the preset into the device
pi_ak4332_init(&ak4332_dev, akinit);

// Load and start Graph
SFU_StartGraph(&SFU_RTD(Graph));

// Write the configuration and start the device
pi_ak4332_start(&ak4332_dev);

// Start I2S transmission
pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL);

// Update the volume as a ramp up
for(uint8_t vol = APP_VOLUME_MIN; vol <= APP_VOLUME_MAX; vol += APP_VOLUME_
→STEP)
{
    pi_ak4332_set_volume(&ak4332_dev, vol);
    pi_time_wait_us(APP_VOLUME_DELAY);
}

```

(continues on next page)

(continued from previous page)

```

    }

    // Waiting for all data to be enqueued
    while(!enqueue_done) pi_yield();

    // Waiting for the last transfer
    SFU_WaitGraphEnd(&SFU_RTD(Graph), 0);

    // Stop the device (pause)
    pi_ak4332_stop(&ak4332_dev);

    // Electricaly disable the AK4332 -> This resets AK4332 registers.
    pi_ak4332_disable(&ak4332_dev);

    // Free the device
    pi_ak4332_close(&ak4332_dev);

    // Stop I2S transmission
    pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_STOP, NULL);

    // Stop the SFU than free it
    StopSFU(1);
    pi_l2_free(ChanInCtxt, sizeof(SFU_uDMA_Channel_T));
}

}

return retval;
}

int main(void)
{
    int retval = test_entry();
    pmsis_exit(retval);
    return 0;
}

```

Graph.src

```

SFU_CreateGraph("Graph");

Filters:
/* {CRFB_INT,           {B0,      B1,      B2,      B3,      B4,      mB5,
                         mA0,      mA1,      mA2,      mA3,      mA4,
                         mG0,      mG1,
                         PRS0,      PRS1,      LBS0,      LBS1,
                         QP,      QN
                     }
   }
 */
Modulator_Lin = {CRFB_INT,           {5540,                  70669,                  458739,
←          2098672,          4665325,      -8388608,
                           -5540,                  -70669,                  -458739,
←          -2098672,          -4665325,

```

(continues on next page)

(continued from previous page)

```

        -4248414,           -1500259,
        -23, -23, -8, -8,
        0x2000000,          -0x2000000
    }
};

```

**Nodes:**

```

In1 = Node(MEM_IN);
Out1 = Node(PDM_OUT, 4, Modulator_Lin);

```

**Connects:**

```
Connect(In1, Out1);
```

```
SFU_CloseGraph();
```

CMakeLists.txt

```

# Copyright (c) 2022 GreenWaves Technologies SAS
# All rights reserved.

#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# 1. Redistributions of source code must retain the above copyright notice,
#    this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. Neither the name of GreenWaves Technologies SAS nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

```

```
cmake_minimum_required(VERSION 3.16)
```

```
#####
# Panel Control
#####
set(TARGET_NAME "ak4332_pdm")
```

(continues on next page)

(continued from previous page)

```

message(" ")
message("---- Building SFU Graph ---")
message(" ")
execute_process(COMMAND           SFU -i ${CMAKE_SOURCE_DIR}/Graph.src -C
                 WORKING_DIRECTORY ${CMAKE_BINARY_DIR})
message(" ")
message("-----")
message(" ")

LIST(APPEND TARGET_SRCS ak4332_pdm.c
      ${ENV{SFU_RUNTIME}}/SFU_RT.c
      ${CMAKE_BINARY_DIR}/Graph_L2_Descr.c
)

LIST(APPEND APP_COMPILE_OPTIONS -I${ENV{SFU_RUNTIME}}/include
      -I${CMAKE_BINARY_DIR}
      -I${ENV{GAP_SDK_HOME}}/tools/autotiler_v3/Emulation
)
#####
# CMake pre initialization
#####

set(CONFIG_GAP_SDK_HOME ${ENV{GAP_SDK_HOME}})
include(${ENV{GAP_SDK_HOME}}/utils/cmake/setup.cmake)
setup_cmake()

project(${TARGET_NAME} C ASM)
add_executable(${TARGET_NAME} ${TARGET_SRCS})
target_compile_options(${TARGET_NAME} PUBLIC ${APP_COMPILE_OPTIONS})

#####
# App's options interpretation
#####

if(NOT DEFINED CONFIG_BOARD_GAP9_EVK)
    message(WARNING "Please select BOARD_GAP9_EVK option to make the example works")
endif()

#####
# CMake post initialization
#####
setupos(${TARGET_NAME})

```

Makefile

APP	= ak4332_pdm
APP_SRCS	+= ak4332_pdm.c \$(TARGET_BUILD_DIR)/Graph_L2_Descr.c \${SFU_RUNTIME}/SFU_RT.c
APP_CFLAGS	+= -Os -g -I\$(TARGET_BUILD_DIR) -I\$(SFU_RUNTIME)/include

(continues on next page)

(continued from previous page)

```
CONFIG_AK4332=1

graph:
    mkdir -p $(TARGET_BUILD_DIR)
    cd $(TARGET_BUILD_DIR) && SFU -i $(CURDIR)/Graph.src -C

include $(RULES_DIR)/pmsis_rules.mk
```

Source signal

```
#####
// This file defines a sine wave coded on 24bits (but stored on int32_t type)
// N Samples : 768
// Period     : 48 samples
// Amplitude  : Half of signed integer range
// N period   : 16
//
// Note:    Table size is available at the end of the file with
//           AK4332_SINE24_SIZE and AK4332_SINE24_SIZE_BYTES macros.
#####

int32_t ak4332_sine24[] = {
    0,
    1094933,
    2171131,
    3210181,
    4194303,
    5106660,
    5931641,
    6655129,
    7264747,
    7750062,
    8102772,
    8316841,
    8388607,
    8316841,
    8102772,
    7750062,
    7264747,
    6655129,
    5931641,
    5106660,
    4194303,
    3210181,
    2171131,
    1094933,
    0,
    -1094933,
    -2171131,
    -3210181,
```

(continues on next page)

(continued from previous page)

```
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194304,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194303,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,
```

(continues on next page)

(continued from previous page)

```
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194304,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,
```

(continues on next page)

(continued from previous page)

```
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,
```

(continues on next page)

(continued from previous page)

```
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194304,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,
```

(continues on next page)

(continued from previous page)

```
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,
```

(continues on next page)

(continued from previous page)

```
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,
```

(continues on next page)

(continued from previous page)

```
4194304,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,
```

(continues on next page)

(continued from previous page)

7264747,
7750062,
8102772,
8316841,
8388607,
8316841,
8102772,
7750062,
7264747,
6655129,
5931641,
5106660,
4194304,
3210181,
2171131,
1094933,
0,
-1094933,
-2171131,
-3210181,
-4194303,
-5106660,
-5931641,
-6655129,
-7264747,
-7750062,
-8102772,
-8316841,
-8388607,
-8316841,
-8102772,
-7750062,
-7264747,
-6655129,
-5931641,
-5106660,
-4194304,
-3210181,
-2171131,
-1094933,
0,
1094933,
2171131,
3210181,
4194303,
5106660,
5931641,
6655129,
7264747,
7750062,
8102772,
8316841,

(continues on next page)

(continued from previous page)

```
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194304,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194304,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,
```

(continues on next page)

(continued from previous page)

```
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194304,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,
```

(continues on next page)

(continued from previous page)

```
4194303,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194304,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,
```

(continues on next page)

(continued from previous page)

```
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,
```

(continues on next page)

(continued from previous page)

```
-4194303,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194303,  
-5106660,  
-5931641,  
-6655129,
```

(continues on next page)

(continued from previous page)

-7264747,  
-7750062,  
-8102772,  
-8316841,  
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194303,  
-3210181,  
-2171131,  
-1094933,  
0,  
1094933,  
2171131,  
3210181,  
4194303,  
5106660,  
5931641,  
6655129,  
7264747,  
7750062,  
8102772,  
8316841,  
8388607,  
8316841,  
8102772,  
7750062,  
7264747,  
6655129,  
5931641,  
5106660,  
4194304,  
3210181,  
2171131,  
1094933,  
0,  
-1094933,  
-2171131,  
-3210181,  
-4194304,  
-5106660,  
-5931641,  
-6655129,  
-7264747,  
-7750062,  
-8102772,  
-8316841,

(continues on next page)

(continued from previous page)

```
-8388607,  
-8316841,  
-8102772,  
-7750062,  
-7264747,  
-6655129,  
-5931641,  
-5106660,  
-4194304,  
-3210181,  
-2171131,  
-1094933  
};  
  
#define AK4332_SINE24_SIZE      (sizeof(ak4332_sine24)/sizeof(ak4332_sine24[0]))  
#define AK4332_SINE24_SIZE_BYTES (sizeof(ak4332_sine24))
```

## Hello World

### Requirements

No specific requirement. This example should run without issue on GAP9 EVK.

### Description

This example shows you how to use READFS on GAP9 with mram or external flash, w/ or w/o XIP.

```
# Boot from JTAG  
make clean all run  
  
# Boot from MRAM with XIP  
make clean all io=uart CONFIG_BOOT_DEVICE=mram CONFIG_XIP=1
```

### Code

C

```
/*  
 * Copyright (C) 2017 ETH Zurich, University of Bologna and GreenWaves Technologies  
 * All rights reserved.  
 *  
 * This software may be modified and distributed under the terms  
 * of the BSD license. See the LICENSE file for details.  
 *  
 * Authors: Germain Haugou, ETH (germain.haugou@iis.ee.ethz.ch)  
 */
```

(continues on next page)

(continued from previous page)

```

#include "pmsis.h"
#include "bsp/fs.h"
#include <bsp/bsp.h>
#include "stdio.h"
#include <bsp/flash/spiflash.h>
#include <bsp/fs/readfs.h>

#define ITER      ( 10 )
#define BUFF_SIZE ( 64 )

#define PAD_GPIO_LED2    (PI_PAD_086)
#define BLINK_DELAY_US   (500 * 1000)

static PI_L2 char buff[2][BUFF_SIZE];
static int count_done = 0;
static pi_fs_file_t *file[2];
static struct pi_device fs;
static struct pi_device flash;

#ifdef USE_MRAM
struct pi_mram_conf flash_conf;
#elif defined(USE_SPIFLASH)
static struct pi_mx25u51245g_conf flash_conf;
#else
static struct pi_default_flash_conf flash_conf;
#endif

static uint32_t index0 = 0;
static uint32_t index1 = 0;

static int exec_tests()
{
    int errors = 0;

    int size0 = file[0]->size;
    int size1 = file[1]->size;
    int rd_size0, rd_size1;

    do{
        if (size0 > BUFF_SIZE)
            rd_size0 = BUFF_SIZE;
        else
            rd_size0 = size0;

        if (size1 > BUFF_SIZE)
            rd_size1 = BUFF_SIZE;
        else
            rd_size1 = size1;

        pi_fs_read(file[0], buff[0], rd_size0);
        pi_fs_read(file[1], buff[1], rd_size1);
    }
}

```

(continues on next page)

(continued from previous page)

```

size0 -= rd_size0;
size1 -= rd_size1;

for (int i=0; i<rd_size0; i++)
{
    unsigned char expected;
    expected = (index0%128) & 0x7f;
    if (expected != buff[0][i])
    {
        printf("Error, buffer: %d, index: %d, expected: 0%x, read: 0%x\n", 0, i, ↵
expected, buff[0][i]);
        return -6;
    }
    index0++;
}

for (int i=0; i<rd_size1; i++)
{
    unsigned char expected;
    expected = (index1 %128) | 0x80;
    if (expected != buff[1][i])
    {
        printf("Error, buffer: %d, index: %d, expected: 0%x, read: 0%x\n", 1, i, ↵
expected, buff[1][i]);
        return -6;
    }
    index1++;
}

}while(size0 && size1);
index0 = 0;
index1 = 0;

return errors;
}

#define QUOTE(name) #name
#define STR(macro) QUOTE(macro)

static int test_entry()
{
    struct pi_readfs_conf conf;
    pi_readfs_conf_init(&conf);

#ifdef USE_MRAM
    pi_mram_conf_init(&flash_conf);
#elif defined(USE_SPIFLASH)
    pi_mx25u51245g_conf_init(&flash_conf);
#else
    pi_default_flash_conf_init(&flash_conf);
#endif
}

```

(continues on next page)

(continued from previous page)

```

pi_open_from_conf(&flash, &flash_conf);

if (pi_flash_open(&flash))
    return -1;
printf("flash opened\n");

conf.fs.flash = &flash;

pi_open_from_conf(&fs, &conf);

printf("lala\n");
if (pi_fs_mount(&fs))
    return -2;
printf("fs mounted\n");

file[0] = pi_fs_open(&fs, STR(FILE0), 0);
if (file[0] == NULL) return -3;

file[1] = pi_fs_open(&fs, STR(FILE1), 0);
if (file[1] == NULL) return -4;

printf("fs opened\n");
if (exec_tests())
    return -5;

pi_fs_close(file[0]);
pi_fs_close(file[1]);

pi_fs_unmount(&fs);
pi_flash_close(&flash);

printf("FS read and check success\n");

return 0;
}

int main(void)
{
    printf("Test start\n");
    int ret = 0;
    pi_device_t gpio_led;
    struct pi_gpio_conf gpio_conf;

    pi_gpio_conf_init(&gpio_conf);

    gpio_conf.port = PAD_GPIO_LED2 / 32;

    pi_open_from_conf(&gpio_led, &gpio_conf);

    if (pi_gpio_open(&gpio_led))
    {
        pmsis_exit(-1);
}

```

(continues on next page)

(continued from previous page)

```

    }

    /* set pad to gpio mode */
    pi_pad_set_function(PAD_GPIO_LED2, PI_PAD_FUNC1);

    /* configure gpio output */
    pi_gpio_flags_e flags = PI_GPIO_OUTPUT;
    pi_gpio_pin_configure(&gpio_led, PAD_GPIO_LED2, flags);

    int gpio_val = 1;

    printf("gpio set\n");

    for(int i=0; i<ITER; i++)
    {
        pi_gpio_pin_write(&gpio_led, PAD_GPIO_LED2, gpio_val);
        ret += test_entry();
        gpio_val ^= 1;
        pi_time_wait_us(BLINK_DELAY_US);
    }

    printf("Test success !\n");
    pmsis_exit(ret);
    return 0;
}

```

Makefile

```

FS_TYPE ?= read_fs
FLASH_TYPE ?= MRAM

USE_PMSIS_BSP=1

FILE0_NAME = flash_file_0.bin
FILE0 = files/$(FILE0_NAME)
FILE1_NAME = flash_file_1.bin
FILE1 = files/$(FILE1_NAME)
FILES = $(FILE0) $(FILE1)
FILE0_PATH = $(FILE0_NAME)
FILE1_PATH = $(FILE1_NAME)

ifeq '$(FLASH_TYPE)' 'MRAM'
APP_CFLAGS += -DUSE_MRAM
READFS_FLASH = target/chip/soc/mram
else
APP_CFLAGS += -DUSE_SPIFLASH
endif
APP_CFLAGS += -DFLASH_TYPE=$(FLASH_TYPE)

READFS_FILES = $(FILES)

APP_CFLAGS += -DFS_READ_FS

```

(continues on next page)

(continued from previous page)

```

APP_CFLAGS += -DFILE0=$(FILE0_PATH)
APP_CFLAGS += -DFILE1=$(FILE1_PATH)

APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

include $(RULES_DIR)/pmsis_rules.mk

```

## Blink LED

### Description

Blinks LED2 (GPIO86) On GAP9 EVK

### Instructions

None

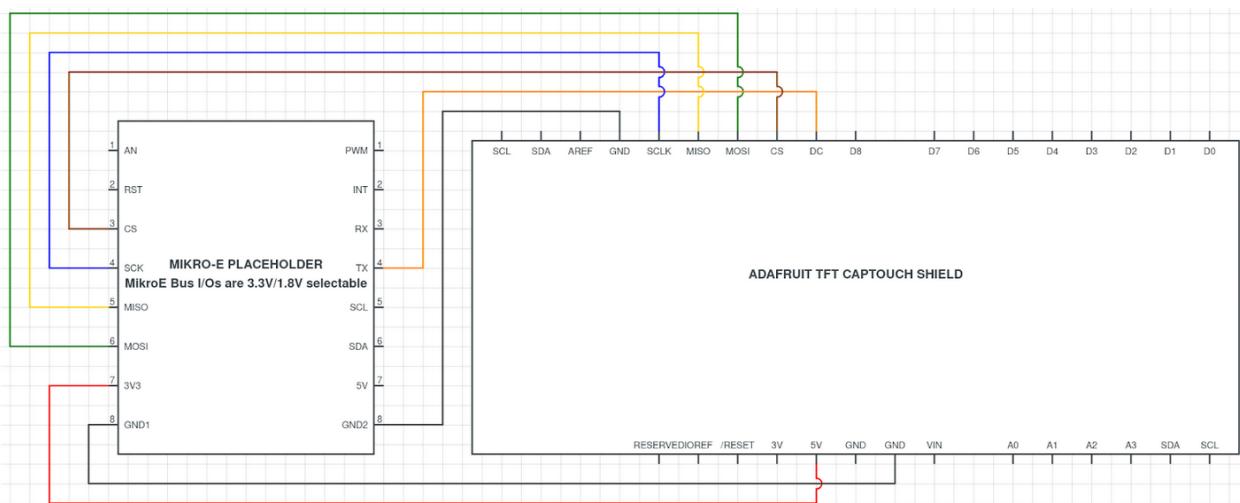
### Exercized hardware

- LED-2 (GPIO86)

### Display Example

### Requirements

HW: Display ili9341 GAP9\_EVK: Jumper J8 on the right Connection Schematic for GAP9\_EVK:



## Description

This example shows how to display images in RGB and grayscale in the Adafruit display (ili9341).

## Code

### Main

```
/*
 * Copyright (C) 2019 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 *
 * Authors: Germain Haugou, ETH (germain.haugou@iis.ee.ethz.ch)
 */

#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera.h"
#include "gaplib/ImgIO.h"
#include "bsp/display/ili9341.h"
#include "bsp/camera/ov5647.h"

#define __XSTR(__s) __STR(__s)
#define __STR(__s) #__s

#ifndef FROM_CAMERA
    pi_device_t i2c = {0};
    #define WIDTH      320
    #define HEIGHT     240
#else
    #define WIDTH      128
    #define HEIGHT     128
#endif

struct pi_device ili;
static pi_buffer_t buffer;

PI_L2 unsigned char ImageBuff[WIDTH*HEIGHT*3];
char *ImageName = __XSTR(IMAGE_PATH);

#ifndef FROM_CAMERA
static int open_camera(struct pi_device *device)
{
    printf("Opening CSI2 camera\n");

    struct pi_ov9281_conf cam_conf;
    pi_ov9281_conf_init(&cam_conf);
    #if VGA

```

(continues on next page)

(continued from previous page)

```

printf("Setting VGA Mode\n");
cam_conf.format=PI_CAMERA_VGA;
#elsif WXGA
printf("Setting WXGA Mode\n");
cam_conf.format=PI_CAMERA_WXGA;
#else
printf("Setting QVGA Mode\n");
cam_conf.format=PI_CAMERA_QVGA;
#endif
pi_open_from_conf(device, &cam_conf);
if (pi_camera_open(device))
    return -1;

return 0;
}
#endif

static int open_display(struct pi_device *device)
{
    struct pi ili9341_conf ili_conf;
    pi_ili9341_conf_init(&ili_conf);
    pi_open_from_conf(device, &ili_conf);
    if (pi_display_open(device))
        return -1;
    //orientation at 270 degrees
    if (pi_display_ioctl(device, PI_ILI_IOCTL_ORIENTATION, (void *)PI_ILI_ORIENTATION_
270))
        return -1;
    return 0;
}

int main(void)
{
    printf("Entering main controller\n");

    /* Open display */
    printf("opening display...\n");
    if (open_display(&ili))
    {
        printf("Failed to open display\n");
        pmsis_exit(-1);
    }
    printf("Display opened!\n");

#ifdef FROM_CAMERA
    struct pi_device camera;
    pi_evt_t task;

    if (open_camera(&camera))
    {
        printf("Failed to open camera\n");
        pmsis_exit(-2);
    }
}

```

(continues on next page)

(continued from previous page)

```

    }

    printf("Camera Opened!\n");
    pi_buffer_init(&buffer, PI_BUFFER_TYPE_L2, ImageBuff);
    pi_buffer_set_stride(&buffer, 0);
    pi_buffer_set_format(&buffer, WIDTH, HEIGHT, 1, PI_BUFFER_FORMAT_GRAY);

    pi_buffer_set_data(&buffer, ImageBuff);
    while(1) {

        pi_camera_capture_async(&camera, ImageBuff, WIDTH*HEIGHT, pi_evt_sig_init(&
    ↪task));
        pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);
        pi_evt_wait(&task);
        pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);

        pi_display_write(&ili, &buffer, 0, 0, WIDTH, HEIGHT);
    }

    pi_camera_close(&camera);

#else
    printf("\nLoading Image %s from File\n", ImageName);
    if (ReadImageFromFile(ImageName, WIDTH, HEIGHT, 3, ImageBuff, WIDTH * HEIGHT * 3,
    ↪IMGIO_OUTPUT_CHAR, 0))
    {
        printf("Failed to load image %s\n", ImageName);
        pmsis_exit(-6);
    }

#define DISPLAY_RGB
/* Convert RGB8 to RGB565 for the display */
for (int i=0; i<WIDTH*HEIGHT; i++) ((unsigned short *) ImageBuff)[i] =_=
    (((uint16_t)ImageBuff[3*i]&0xf8)<<8) | (((uint16_t)ImageBuff[3*i+1]&0xfc)<<3) | (((uint16_
    ↪t)ImageBuff[3*i+2]&0xf8)>>3));
    pi_buffer_format_e format = PI_BUFFER_FORMAT_RGB565;
#else
/* Convert RGB8 to GRAY8 for the display */
for (int i=0; i<WIDTH*HEIGHT; i++) ImageBuff[i] = ImageBuff[3*i];
pi_buffer_format_e format = PI_BUFFER_FORMAT_GRAY;
#endif

    pi_buffer_init(&buffer, PI_BUFFER_TYPE_L2, ImageBuff);
    pi_buffer_set_stride(&buffer, 0);
    pi_buffer_set_format(&buffer, WIDTH, HEIGHT, 1, format);

    pi_buffer_set_data(&buffer, ImageBuff);
    pi_display_write(&ili, &buffer, 0, 0, WIDTH, HEIGHT);

#endif

    return 0;
}

```

## Makefile

```

APP = display_example
APP_SRCS += display_example.c
APP_CFLAGS += -O3 -g

CONFIG_GAP_LIB_IMGIO = 1
CONFIG_ILI9341      = 1
CONFIG_DISPLAY       = 1

IMAGE_PATH=$(CURDIR)/ILSVRC2012_val_00011158_128.ppm

APP_CFLAGS += -DLOGS -DIMAGE_PATH=$(IMAGE_PATH)

RGB ?= 1
ifeq ($(RGB), 1)
    APP_CFLAGS += -DDISPLAY_RGB
endif

CAMERA ?= 0
ifeq ($(CAMERA), 1)
    APP_CFLAGS += -DFROM_CAMERA
endif

include $(RULES_DIR)/pmsis_rules.mk

```

## Cluster

These examples show how to use cluster features on GAP9.

## GVSOC

These examples show how to use GVSoC with GAP9.

### I2S control

#### Requirements

None

#### Description

This example shows how to run GVSOC with a telnet proxy opened, and how to connect a python script to it to control the behavior of the I2S testbench dynamically.

## Code

Python

```
#!/usr/bin/env python3

import argparse
import gv.gvsoc_control as gvsoc

parser = argparse.ArgumentParser(description='Control GVSOC')

parser.add_argument("--host", dest="host", default="localhost", help="Specify host name")
parser.add_argument("--port", dest="port", default=42951, type=int, help="Specify host port")

args = parser.parse_args()

gv = gvsoc.Proxy(args.host, args.port)

testbench = gvsoc.Testbench(gv)

# Open SAI 0 interface
i2s_0 = testbench.i2s_get(0)

i2s_0.open(sampling_freq=44100, word_size=16, nb_slots=1, is_ext_clk=True, is_ext_ws=True)

# Setup slot 0 RX with input wav file
i2s_0.slot_open(slot=0, is_rx=True, word_size=16, is_msb=True, sign_extend=False, left_align=False)
i2s_0.slot_rx_file_reader(slot=0, filetype="wav", filepath='../sound_0.wav')

# Setup slot 0 TX with output wav file. The test will open it with a loopback to redirect sdi to sdo
i2s_0.slot_open(slot=0, is_rx=False, word_size=16, is_msb=True, sign_extend=False, left_align=False)
i2s_0.slot_tx_file_dumper(slot=0, filetype="au", filepath='../sound_out.au')

# Start the clock
i2s_0.clk_start()

# Run for 1 s
gv.run(1000000000000000)

# Stop the clock and the RX slot
i2s_0.slot_stop(slot=0, stop_rx=True, stop_tx=False)
i2s_0.clk_stop()

# Run for 10 ms
gv.run(10000000000)

# Now continue with a different file
```

(continues on next page)

(continued from previous page)

```
i2s_0.slot_rx_file_reader(slot=0, filetype="wav", filepath='../sound_1.wav')
i2s_0.clk_start()

# Run for 1 s
gv.run(1000000000000)

# Stop everything and quit
i2s_0.clk_stop()
i2s_0.slot_close(slot=0)
i2s_0.close()

gv.quit()
gv.close()
```

C

```
/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 */

#include "pmsis.h"
#include "stdio.h"

#define NB_ELEM 32
#define BUFFER_SIZE (NB_ELEM * 2)

static int16_t *buffers[2];

int main(void)
{
    pi_device_t i2s;

    // We will will samples from slot 0 in memory and also redirect them to TX with the
    // bypass

    // Open I2S for 44100, 16 bits, 1 slot
    struct pi_i2s_conf i2s_conf;
    pi_i2s_conf_init(&i2s_conf);

    i2s_conf.itf = 0;
    i2s_conf.format = PI_I2S_FMT_DATA_FORMAT_I2S;
    i2s_conf.word_size = 16;
    i2s_conf.channels = 1;
    i2s_conf.options = PI_I2S_OPT_TDM | PI_I2S_OPT_EXT_CLK | PI_I2S_OPT_EXT_WS;

    pi_open_from_conf(&i2s, &i2s_conf);
```

(continues on next page)

(continued from previous page)

```

if (pi_i2s_open(&i2s))
    return -1;

// Open slot 0 RX
struct pi_i2s_channel_conf channel_conf;
pi_i2s_channel_conf_init(&channel_conf);

channel_conf.options = PI_I2S_OPT_PINGPONG | PI_I2S_OPT_IS_RX | PI_I2S_OPT_ENABLED;
buffers[0] = pi_l2_malloc(BUFFER_SIZE);
buffers[1] = pi_l2_malloc(BUFFER_SIZE);
if (buffers[0] == NULL || buffers[1] == NULL)
{
    return -1;
}
channel_conf.pingpong_buffers[0] = buffers[0];
channel_conf.pingpong_buffers[1] = buffers[1];

channel_conf.block_size = BUFFER_SIZE;
channel_conf.word_size = 16;
channel_conf.format = PI_I2S_FMT_DATA_FORMAT_I2S | PI_I2S_CH_FMT_DATA_ORDER_MSB;

// Open slot 0 TX in bypass mode
if (pi_i2s_channel_conf_set(&i2s, 0, &channel_conf))
    return -1;

pi_i2s_channel_conf_init(&channel_conf);

channel_conf.options = PI_I2S_OPT_IS_TX | PI_I2S_OPT_ENABLED | PI_I2S_OPT_LOOPBACK;

if (pi_i2s_channel_conf_set(&i2s, 0, &channel_conf))
    return -1;

if (pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL))
    return -1;

// Print the first buffer of samples
void *read_buffer;
int size;
pi_i2s_channel_read(&i2s, 0, &read_buffer, &size);

for (int i=0; i<NB_ELEM; i++)
{
    printf("Sample %d: %d\n", i, buffers[0][i]);
}

// Then do nothing and let the bypass propagate samples to TX
while(1)
{
    pi_time_wait_us(100000000);
}

```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

### Makefile

```
APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

# Activate proxy mode
GV_PROXY = 1
GV_PROXY_PORT = 30000

# Activate GVSOC debug mode so that traces can be enabled dynamically
GV_DEBUG_MODE = 1

CONFIG_TESTBENCH = 1
CONFIG_I2S = 1

# This target should be launched from another terminal and will connect to gvsoc
# to launch a few commands
proxy:
    ./gvcontrol --host=localhost --port=$(GV_PROXY_PORT)

gen:
    sox -n -r 44100 --bits 16 sound_0.wav synth 3 sine 1000 vol 0.995
    sox -n -r 44100 --bits 16 sound_1.wav synth 3 sine 2000 vol 0.995

include $(RULES_DIR)/pmsis_rules.mk
```

## I2S control stereo

### Requirements

None

## Description

This example shows how to run GVSOC with a proxy opened, and how to connect a python script to it to control the behavior of the I2S testbench dynamically using WAV files in stereo mode.

## Code

Python

```
#!/usr/bin/env python3

import argparse
import gv.gvsoc_control as gvsoc

parser = argparse.ArgumentParser(description='Control GVSOC')

parser.add_argument("--host", dest="host", default="localhost", help="Specify host name")
parser.add_argument("--port", dest="port", default=42951, type=int, help="Specify host port")

args = parser.parse_args()

gv = gvsoc.Proxy(args.host, args.port)

testbench = gvsoc.Testbench(gv)

# Open SAI 0 interface
i2s_0 = testbench.i2s_get(0)

i2s_0.open(sampling_freq=44100, word_size=16, nb_slots=2, is_ext_clk=True, is_ext_ws=True)

# Setup slot 0 and 1 RX with input wav file
i2s_0.slot_open(slot=0, is_rx=True, word_size=16, is_msb=True, sign_extend=False, left_align=False)
i2s_0.slot_open(slot=1, is_rx=True, word_size=16, is_msb=True, sign_extend=False, left_align=False)
# They will share the same files
i2s_0.slot_rx_file_reader(slots=[-1, 0, -1, 1, -1], filetype="wav", filepath='../.../sound_all.wav')

# Setup slot 0 TX with output wav file. The test will open it with a loopback to redirect sdi to sdo
i2s_0.slot_open(slot=0, is_rx=False, word_size=16, is_msb=True, sign_extend=False, left_align=False)
i2s_0.slot_open(slot=1, is_rx=False, word_size=16, is_msb=True, sign_extend=False, left_align=False)
i2s_0.slot_tx_file_dumper(slots=[-1, 0, 0, -1, 1], filetype="wav", filepath='../.../sound_out.wav')

# Start the clock
```

(continues on next page)

(continued from previous page)

```
i2s_0.clk_start()

# Run for 5 s
gv.run(100000000000000)

# Stop the clock and the RX slot
i2s_0.slot_stop(slot=0)
i2s_0.slot_stop(slot=1)

# Stop everything and quit
i2s_0.clk_stop()
i2s_0.slot_close(slot=0)
i2s_0.slot_close(slot=1)
i2s_0.close()

gv.quit()
gv.close()
```

C

```
/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 */
#include "pmsis.h"
#include "stdio.h"

#define NB_ELEM 32
#define BUFFER_SIZE (NB_ELEM * 2)

static int16_t *buffers[2];

int main(void)
{
    pi_device_t i2s;

    // We will will samples from slot 0 in memory and also redirect them to TX with the bypass
    // Open I2S for 44100, 16 bits, 1 slot
    struct pi_i2s_conf i2s_conf;
    pi_i2s_conf_init(&i2s_conf);

    i2s_conf.itf = 0;
    i2s_conf.format = PI_I2S_FMT_DATA_FORMAT_I2S;
    i2s_conf.word_size = 16;
```

(continues on next page)

(continued from previous page)

```

i2s_conf.channels = 1;
i2s_conf.options = PI_I2S_OPT_TDM | PI_I2S_OPT_EXT_CLK | PI_I2S_OPT_EXT_WS;

pi_open_from_conf(&i2s, &i2s_conf);

if (pi_i2s_open(&i2s))
    return -1;

// Open slot 0 RX
struct pi_i2s_channel_conf channel_conf;
pi_i2s_channel_conf_init(&channel_conf);

channel_conf.options = PI_I2S_OPT_PINGPONG | PI_I2S_OPT_IS_RX | PI_I2S_OPT_ENABLED;
buffers[0] = pi_l2_malloc(BUFFER_SIZE);
buffers[1] = pi_l2_malloc(BUFFER_SIZE);
if (buffers[0] == NULL || buffers[1] == NULL)
{
    return -1;
}
channel_conf.pingpong_buffers[0] = buffers[0];
channel_conf.pingpong_buffers[1] = buffers[1];

channel_conf.block_size = BUFFER_SIZE;
channel_conf.word_size = 16;
channel_conf.format = PI_I2S_FMT_DATA_FORMAT_I2S | PI_I2S_CH_FMT_DATA_ORDER_MSB;

// Open slot 0 TX in bypass mode
if (pi_i2s_channel_conf_set(&i2s, 0, &channel_conf))
    return -1;

pi_i2s_channel_conf_init(&channel_conf);

channel_conf.options = PI_I2S_OPT_IS_TX | PI_I2S_OPT_ENABLED | PI_I2S_OPT_LOOPBACK;

if (pi_i2s_channel_conf_set(&i2s, 0, &channel_conf))
    return -1;

//if (pi_i2s_channel_conf_set(&i2s, 1, &channel_conf))
//    return -1;

if (pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL))
    return -1;

// Print the first buffer of samples
void *read_buffer;
int size;
pi_i2s_channel_read(&i2s, 0, &read_buffer, &size);

for (int i=0; i<NB_ELEM; i++)
{
    printf("Sample %d: %d\n", i, buffers[0][i]);
}

```

(continues on next page)

(continued from previous page)

```
// Then do nothing and let the bypass propagate samples to TX
while(1)
{
    pi_time_wait_us(10000000);
}

return 0;
}
```

Makefile

```
APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

# Activate proxy mode
GV_PROXY = 1
GV_PROXY_PORT = 30000

# Activate GVSOC debug mode so that traces can be enabled dynamically
GV_DEBUG_MODE = 1

CONFIG_TESTBENCH = 1
CONFIG_I2S = 1

# This target should be launched from another terminal and will connect to gvsoc
# to launch a few commands
proxy:
    ./gvcontrol --host=localhost --port=$(GV_PROXY_PORT)

gen:
    sox -n -r 44100 --bits 16 sound_0.wav synth 3 sine 1000 vol 0.995
    sox -n -r 44100 --bits 16 sound_1.wav synth 3 sine 2000 vol 0.995
    sox -n -r 44100 --bits 16 sound_2.wav synth 3 sine 4000 vol 0.995
    sox -n -r 44100 --bits 16 sound_3.wav synth 3 sine 8000 vol 0.995
    sox -n -r 44100 --bits 16 sound_4.wav synth 3 sine 16000 vol 0.995
    sox -M sound_0.wav sound_1.wav sound_2.wav sound_3.wav sound_4.wav sound_all.wav

include $(RULES_DIR)/pmsis_rules.mk
```

**Memory access****Requirements**

None

**Description**

This example shows how to run GVSOC with a telnet proxy opened, and how to connect a python script to it to do memory accesses.

**Code**

Python

```
#!/usr/bin/env python3

import argparse
import gv.gvsoc_control as gvsoc

parser = argparse.ArgumentParser(description='Control GVSOC')

parser.add_argument("--host", dest="host", default="localhost", help="Specify host name")
parser.add_argument("--port", dest="port", default=42951, type=int, help="Specify host port")

args = parser.parse_args()

gv = gvsoc.Proxy(args.host, args.port)

axi = gvsoc.Router(gv)

# Wait 1ms to be sure the ROM has booted since we are writing at beginning of binary
gv.run(1000000000)

# Send a value to the simulated test
axi.mem_write_int(0x1c000000, 4, 0x11223344)

# And wait until it sends back a special value
while True:
    gv.run(1000000000)

    result = axi.mem_read_int(0x1c000000, 4)
    if result == 0x55667788:
        break;

gv.quit()
gv.close()
```

C

```

/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 *
 */

#include "pmsis.h"
#include "stdio.h"

int main(void)
{
    while(*	volatile int *)0x1c000000 != 0x11223344)
    {
    }

    printf("Received expected value\n");

    *(volatile int *)0x1c000000 = 0x55667788;

    while(1);

    return 0;
}

```

Makefile

```

APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

# Activate proxy mode
GV_PROXY = 1
GV_PROXY_PORT = 30000

# Activate GVSOC debug mode so that traces can be enabled dynamically
GV_DEBUG_MODE = 1

# This target should be launched from another terminal and will connect to gvsoc
# to launch a few commands
proxy:
    ./gvcontrol --host=localhost --port=$(GV_PROXY_PORT)

include $(RULES_DIR)/pmsis_rules.mk

```

## Telnet proxy

### Requirements

None

### Description

This example shows how to run GVSOC with a telnet proxy opened, and how to connect a python script to it to control its behavior dynamically.

### Code

Python

```
#!/usr/bin/env python3

import argparse
import gv.gvsoc_control as gvsoc

parser = argparse.ArgumentParser(description='Control GVSOC')

parser.add_argument("--host", dest="host", default="localhost", help="Specify host name")
parser.add_argument("--port", dest="port", default=42951, type=int, help="Specify host port")

args = parser.parse_args()

gv = gvsoc.Proxy(args.host, args.port)

# Run for 10 ms
gv.run(100000000)

# Activate FC instructions and dump to file log
gv.trace_add('fc/insn:log')

# Run for 10 ms with traces enabled
gv.run(100000000)

# Deactivate FC instructions
gv.trace_remove('fc/insn:log')

# Run until the end without traces
gv.run()

# Close proxy connection
gv.close()
```

C

```

/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 *
 */

#include "pmsis.h"
#include "stdio.h"

int main(void)
{
    printf("(%ld, %ld) Entering main controller\n", pi_cluster_id(), pi_core_id());

    return 0;
}

```

Makefile

```

APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

# Activate proxy mode
GV_PROXY = 1
GV_PROXY_PORT = 30000

# Activate GVSOC debug mode so that traces can be enabled dynamically
GV_DEBUG_MODE = 1

# This target should be launched from another terminal and will connect to gvsoc
# to launch a few commands
proxy:
    ./gvcontrol --host=localhost --port=$(GV_PROXY_PORT)

include $(RULES_DIR)/pmsis_rules.mk

```

**Trace control****Requirements**

None

**Description**

This example shows how to run control GVSOC traces during execution from the simulated SW.

**Code**

C

```
/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 */

#include "pmsis.h"
#include "stdio.h"

#ifndef __PLATFORM_GVSOC__
#include <pmsis/platforms/gvsoc.h>
#endif

int main(void)
{
    // Activate FC instruction traces and dump to file log
    gv_trace_enable("fc/insn:log");

    printf("(%ld, %ld) Entering main controller\n", pi_cluster_id(), pi_core_id());

    // Deactivate traces
    gv_trace_disable("fc/insn:log");

    return 0;
}
```

Makefile

```
APP = test
APP_SRCS = test.c
APP_CFLAGS += -O3 -g

# Activate GVSOC debug mode so that traces can be enabled dynamically
```

(continues on next page)

(continued from previous page)

```
GV_DEBUG_MODE = 1

include $(RULES_DIR)/pmsis_rules.mk
```

## Uart user model

### Requirements

None

### Description

This example shows how to run GVSOC with a custom board description and a user model.

Traces of the model can be obtained with command:

```
make all run runner_args="--trace=my_uart_device --trace-level=debug"
```

The model must be compiled separately and before launching the test with this command:

```
make -f gvsoc.mk build
```

### Code

#### GVSOC Model

Python

```
# 
# Copyright (C) 2020 GreenWaves Technologies
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# 
import gsytree as st

# Components must always derive from st.Component
class My_uart_device(st.Component):
```

(continues on next page)

(continued from previous page)

```
def __init__(self, parent, name):

    super(My_uart_device, self).__init__(parent, name)

    # Add properties
    # By declaring the baudrate as a property, it is made available to the model
    # implementation and the value can be overwritten from the makefile

    self.add_property('baudrate', 200000)

    # Add t
    self.add_property('vp_component', 'my_uart_device_impl')
```

C++

```
/*
 * Copyright (C) 2021 GreenWaves Technologies, SAS
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <vp/vp.hpp>
#include <vp/itf/uart.hpp>
#include <queue>
#include <vp/time/time_scheduler.hpp>

typedef enum
{
    UART_RX_STATE_WAIT_START,
    UART_RX_STATE_DATA,
    UART_RX_STATE_WAIT_STOP
} uart_rx_state_e;

typedef enum
{
    UART_TX_STATE_IDLE,
    UART_TX_STATE_START,
    UART_TX_STATE_DATA,
    UART_TX_STATE_STOP
} uart_tx_state_e;
```

(continues on next page)

(continued from previous page)

```

// Our device derives from time_scheduler so that callbacks can be scheduled
// based on timestamps
class My_uart_device : public vp::time_scheduler
{
public:
    My_uart_device(js::config *config);

    // Called when instantiating architecture
    int build();
    // Called when architecture is instantiated, to call interfaces
    void start();

private:
    // Callback called when the padframe is updating one of the pads of our interface
    static void sync(void *_this, int data, int sck, int rts);

    // Start RX sampling after start bit is received
    void start_rx_sampling();

    // Callback called when it is time to sample RX bit
    static void handle_rx(void *_this, vp::time_event *event);

    // Callback called when it is time to send bit
    static void handle_tx(void *_this, vp::time_event *event);

    void check_tx();

    // For tracing
    vp::trace trace;

    // Slave uart interface
    vp::uart_slave in;

    // Half the baudrate period in ps
    int64_t semi_period;

    // Time event for RX callbacks
    vp::time_event *rx_event;

    // Time event for TX callbacks
    vp::time_event *tx_event;

    // RX FSM state
    uart_rx_state_e rx_state;

    // True if we are waiting for RX start bit
    bool rx_wait_start;

    // Current RX bit received from interface
    int current_data;

```

(continues on next page)

(continued from previous page)

```

// Number of bits received
int rx_sampled_bits;

// Current RX byte
uint8_t rx_sampled_byte;

// Received bytes
std::queue<uint8_t> rx_bytes;

// TX FSM state
uart_tx_state_e tx_state;

// Number of bits to be sent
int tx_pending_bits;

// Byte to be sent
uint8_t tx_pending_byte;
};

My_uart_device::My_uart_device(js::config *config)
    : vp::time_scheduler(config)
{
}

// In this method, only declarations are allowed, no interface call is allowed
int My_uart_device::build()
{
    // Retrieve time engine so that we can post time events
    this->engine = (vp::time_engine*)this->get_service("time");

    traces.new_trace("trace", &trace, vp::DEBUG);

    // Incoming uart interface with associated callback
    this->in.set_sync_full_meth(&My_uart_device::sync);
    new_slave_port("uart", &in);

    // Get baudrate from JSON config and compute period. We take half the period to be able to
    // sample in the middle of the cycle.
    int baudrate = this->get_js_config()->get_int("baudrate");
    this->semi_period = baudrate ? 1000000000000ULL / this->get_js_config()->get_int(
        "baudrate") / 2 : 0;

    this->rx_wait_start = true;
    this->tx_state = UART_TX_STATE_IDLE;

    // Init time events with associated callbacks
    this->rx_event = this->time_event_new(My_uart_device::handle_rx);
    this->tx_event = this->time_event_new(My_uart_device::handle_tx);
}

```

(continues on next page)

(continued from previous page)

```

    return 0;
}

void My_uart_device::start()
{
    // Init uart TX to 1, so that start bit is seen when we start transmitting
    this->in.sync_full(1, 2, 2);
}

void My_uart_device::check_tx()
{
    // In case our TX FSM is idle and they are bytes to be sent, schedule an event
    // to send a byte
    if (this->tx_state == UART_TX_STATE_IDLE && this->rx_bytes.size() > 0)
    {
        this->enqueue(this->tx_event, this->semi_period * 2);
    }
}

void My_uart_device::start_rx_sampling()
{
    // Enqueue the sampling event after 1.5 cycles, so that it starts sampling in the
middle of the cycle.
    this->enqueue(this->rx_event, this->semi_period * 3);
}

void My_uart_device::handle_rx(void *_this, vp::time_event *event)
{
    My_uart_device *_this = (My_uart_device *)_this;

    _this->trace.msg(vp::trace::LEVEL_TRACE, "Sampling data (data: %d)\n", _this->
    current_data);

    // We have to sample one bit.
    // Check in which state we are and either get data or stop bit, and enqueue the next
event
    // in 1 cycle
    switch (_this->rx_state)
    {
        case UART_RX_STATE_DATA:
            _this->rx_sampled_bits++;
            _this->rx_sampled_byte = (_this->rx_sampled_byte >> 1) | (_this->current_
            data << 7);
            if (_this->rx_sampled_bits == 8)
            {
                _this->trace.msg(vp::trace::LEVEL_DEBUG, "Received byte (value: 0x%u)\n",
                _this->rx_sampled_byte);
                _this->rx_bytes.push(_this->rx_sampled_byte);
                _this->check_tx();
            }
    }
}

```

(continues on next page)

(continued from previous page)

```

        _this->rx_state = UART_RX_STATE_WAIT_STOP;
    }
    break;

    case UART_RX_STATE_WAIT_STOP:
        if (_this->current_data == 1)
        {
            _this->trace.msg(vp::trace::LEVEL_TRACE, "Received stop bit\n");
            _this->rx_wait_start = true;
        }
        break;
    }

    if (!_this->rx_wait_start)
    {
        _this->enqueue(_this->rx_event, _this->semi_period * 2);
    }
}

void My_uart_device::handle_tx(void *_this, vp::time_event *event)
{
    My_uart_device *_this = (My_uart_device *)_this;

    // In case we are idle and there are bytes, pop the first one and start sending.
    if (_this->tx_state == UART_TX_STATE_IDLE && _this->rx_bytes.size() > 0)
    {
        _this->tx_pending_byte = _this->rx_bytes.front();
        _this->rx_bytes.pop();
        _this->trace.msg(vp::trace::LEVEL_DEBUG, "Sending byte (value: 0x%x)\n", _this->
        tx_pending_byte);
        _this->tx_pending_bits = 8;
        _this->tx_state = UART_TX_STATE_START;
    }

    // We have to send one bit.
    // Check in which state we are and either send data or stop bit, and enqueue the
    // next event
    // in 1 cycle
    switch (_this->tx_state)
    {
        case UART_TX_STATE_START:
            _this->trace.msg(vp::trace::LEVEL_TRACE, "Sending start bit\n");
            _this->in.sync_full(0, 2, 2);
            _this->tx_state = UART_TX_STATE_DATA;
            break;

        case UART_TX_STATE_DATA:
            _this->trace.msg(vp::trace::LEVEL_TRACE, "Sending bit (value: %d)\n", _this->
            tx_pending_byte & 1);
            _this->in.sync_full(_this->tx_pending_byte & 1, 2, 2);
            _this->tx_pending_byte >>= 1;
    }
}

```

(continues on next page)

(continued from previous page)

```

        _this->tx_pending_bits--;
        if (_this->tx_pending_bits == 0)
        {
            _this->tx_state = UART_TX_STATE_STOP;
        }
        break;

case UART_TX_STATE_STOP:
    _this->trace.msg(vp::trace::LEVEL_TRACE, "Sending stop bit\n");
    _this->in.sync_full(1, 2, 2);
    _this->tx_state = UART_TX_STATE_IDLE;
    break;

}

if (_this->tx_state != UART_TX_STATE_IDLE)
{
    _this->enqueue(_this->tx_event, _this->semi_period * 2);
}
else
{
    // If we are done, check if we can send another byte
    _this->check_tx();
}
}

void My_uart_device::sync(void *_this, int data, int sck, int rts)
{
    My_uart_device *_this = (My_uart_device *)_this;

    _this->trace.msg(vp::trace::LEVEL_TRACE, "Handle edge (data: %d, sck: %d, rts: %d)\n",
    ↵", data, sck, rts);

    // This is called when pads changed. Remember the value of the data pad so that we
    ↵get the value
    // when the FSM will sample
    _this->current_data = data;

    // In case we are waiting for start bit and we receive 0, start sampling
    if (_this->rx_wait_start)
    {
        if (data == 0)
        {
            _this->trace.msg(vp::trace::LEVEL_TRACE, "Received start bit\n");
            _this->rx_wait_start = false;
            _this->rx_state = UART_RX_STATE_DATA;
            _this->start_rx_sampling();
            _this->rx_sampled_bits = 0;
        }
    }
}

```

(continues on next page)

(continued from previous page)

```
// Mandatory constructor to return the class instance
extern "C" vp::component *vp_constructor(js::config *config)
{
    return new My_uart_device(config);
}
```

gvsoc.mk

```
# To compile a model, we have to first declare it. The name chosen here must
# match the one specified in the python script
IMPLEMENTATIONS += my_uart_device_impl

# Then associate sources
my_uart_device_impl_SRCS = my_uart_device_impl.cpp

-include $(INSTALL_DIR)/rules/vp_models.mk
```

Application

C

```
#include <stdio.h>
#include <stdint.h>
#include <pmsis.h>

#define BUFFER_SIZE 256

PI_L2 uint8_t tx_buffer[BUFFER_SIZE];

PI_L2 uint8_t rx_buffer[BUFFER_SIZE];

struct pi_uart_conf conf;
struct pi_device uart;

int main(void)
{
    printf("Entered test\n");

    pi_uart_conf_init(&conf);

    conf.enable_tx = 1;
    conf.enable_rx = 1;
    conf.uart_id = 2;
    conf.baudrate_bps = 115200;

    pi_open_from_conf(&uart, &conf);

    if (pi_uart_open(&uart))
        return -1;
```

(continues on next page)

(continued from previous page)

```

for (int i=0; i<BUFFER_SIZE; i++)
{
    tx_buffer[i] = i;
}

pi_evt_t task;

// We first enqueue a read copy so that we can receive what we will send.
// This must be asynchronous as the transfer cannot finish before
// we enqueue the write.
// The event used here is a blocking event, which means we must explicitely wait for it
// to know when the transfer is finished.
pi_uart_read_async(&uart, rx_buffer, BUFFER_SIZE, pi_evt_sig_init(&task));

// This one is not using any event and is thus blocking.
pi_uart_write(&uart, tx_buffer, BUFFER_SIZE);

// This will block execution until we have received the whole buffer.
pi_evt_wait(&task);

int error = 0;
for (int i=0; i<BUFFER_SIZE; i++)
{
    //printf("%d: @%p: %x @%p: %x\n", i, &rx_buffer[i], rx_buffer[i], &tx_buffer[i], tx_
    ↵buffer[i]);
    if (rx_buffer[i] != tx_buffer[i])
    {
        printf("Error at index %d, expected 0x%02x, got 0x%02x\n", i, tx_buffer[i], rx_
    ↵buffer[i]);
        error++;
    }
}

if (error)
{
    printf("Got %d errors\n", error);
}
else
{
    printf("Test success\n");
}

return error;
}

```

Makefile

```

APP = test
APP_SRCS = test.c
APP_CFLAGS = -O3 -g

CONFIG_UART = 1

```

(continues on next page)

(continued from previous page)

```
# Overwrite the default target so that GVSOC simulates our board
# First name is the class name, second one is the python module
export GAPY_PY_TARGET=My_board@my_board

# Append current directory to python path so that it finds our board and module
export PYTHONPATH:=$(CURDIR):$PYTHONPATH

# The baudrate specified here will be propagated to the model
override config_args += --config-opt=**/my_uart_device/baudrate=115200

include $(RULES_DIR)/pmsis_rules.mk
```

my\_board.py

```
# Copyright (C) 2021 GreenWaves Technologies, SAS
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from gap.gap9.gap9_evk import Gap9_evk
from my_uart_device import My_uart_device

# Declare a board derived from gap9 board
class My_board(Gap9_evk):

    def __init__(self, parent, name):
        super(My_board, self).__init__(parent, name)

        # Gap9 EVK board instantiates gap9 with name 'chip',
        # get it
        gap = self.get_component('chip')

        # And instantiates and connect our model to it
        uart_device = My_uart_device(self, 'my_uart_device')

        self.bind(gap, 'uart2', uart_device, 'uart')
```

## VCD Trace control

### Requirements

None

### Description

This example shows how to control GVSOC VCD custom traces during execution from the simulated SW.

### Code

C

```
/*
 * Copyright (C) 2021 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 *
 */

#include "pmsis.h"
#include "stdio.h"
#include <pmsis/platforms/gvsoc.h>

int main(void)
{
    // Open the 2 traces from their path in the simulated SW and get the descriptor in
    // return
    int trace = gv_vcd_open_trace("/user/kernel/active");
    int trace_state = gv_vcd_open_trace("/user/kernel/state");

    // Now play with the traces

    // Set the integrer trace to 1.
    // Before that it was in the special Z state (middle state in yellow in the view).
    // Since we provided a map file, this value should be seen as a box with active
    // keyword
    gv_vcd_dump_trace(trace, 1);

    // Now set a new string to the string trace
    gv_vcd_dump_trace_string(trace_state, "state0");

    // Deactivate VCD tracing. In the view, this should make a big hole with no traces
    // at all.
    gv_vcd_configure(0, NULL);
```

(continues on next page)

(continued from previous page)

```

printf("Hello\n");

// Now give other values
gv_vcd_configure(1, NULL);

// This will set the special Z value to the trace, like if it was released.
gv_vcd_release_trace(trace);
gv_vcd_dump_trace_string(trace_state, "state1");

gv_vcd_dump_trace(trace, 1);
gv_vcd_dump_trace_string(trace_state, "state2");

printf("Hello\n");

gv_vcd_release_trace(trace);
gv_vcd_dump_trace_string(trace_state, "state3");
}

```

Makefile

```

APP = example
APP_SRCS = example.c
APP_CFLAGS += -O3 -g

# Activate GVSOC debug mode so that traces can be enabled dynamically
# Otherwise GVSOC is run without trace support
GV_DEBUG_MODE = 1

# Now create user custom VCD traces

# Each trace can have the following fields:
#   - tag: tag which can be used to activate this trace from the command-line. Not ↴ supported for now
#   - type: Type of the trace, can be int or string
#   - path: The path of the VCD trace in the simulated system.
#   - vcd_path: The path of the VCD trace in the VCD hierarchy, including its extension
#   - view_path: The path of the VCD trace in the gtkwave view
#   - filter: For integer traces, an optional gtkwave map file

# First one is an integer trace with an associated map file to see some values with a box
override config_args += --config-opt=**/gvsoc/events/traces/kernel/tag=overview
override config_args += --config-opt=**/gvsoc/events/traces/kernel/type=int
override config_args += --config-opt=**/gvsoc/events/traces/kernel/path=/user/kernel/
↪ active
override config_args += --config-opt=**/gvsoc/events/traces/kernel/vcd_path=user.kernel.
↪ active[31:0]
override config_args += --config-opt=**/gvsoc/events/traces/kernel/view_path=overview.
↪ kernel
override config_args += --config-opt=**/gvsoc/events/traces/kernel/filter=$(CURDIR)/map_
↪ file.txt

```

(continues on next page)

(continued from previous page)

```
# Second one is a string trace
override config_args += --config-opt=**/gvsoc/events/traces/kernel_state/tag=state
override config_args += --config-opt=**/gvsoc/events/traces/kernel_state/type=string
override config_args += --config-opt=**/gvsoc/events/traces/kernel_state/path=/user/
↪kernel/state
override config_args += --config-opt=**/gvsoc/events/traces/kernel_state/vcd_path=user.
↪kernel.state
override config_args += --config-opt=**/gvsoc/events/traces/kernel_state/view_
↪path=overview.kernel

include $(RULES_DIR)/pmsis_rules.mk
```

## Peripherals

### I2C

Here are various examples using the PMSIS *I2C*.

#### Scan

#### Requirements

Only works on GAP9.

#### Description

Performs a scan of an I2C interface and reports found peripherals.

Done with the accelerometer & gyroscope module LSM6DS0 of ST (<https://www.sparkfun.com/products/18020>), and the color camera 5647 of Omnivision (<https://www.arducam.com/product/arducam-ov5647-standard-raspberry-pi-camera-b0033/>).

#### Hardware setup

- LSM6DS0

SDA & SCL of the module connected to the SDA & SCL pins of the CN10 of the EVK. GND & 3V3 of the module connected to the GND & 1V8 pins of the CN8 of the EVK.

- Camera

Connected to the EVK through the CSI-2 port.

## Commands

```
# Create a build repository
cmake -B build

# Manage options
cmake --build build --target menuconfig

# Run the example
cmake --build build --target run
```

## Code

C

```
/*
 * I2C Example
 *
 * Scanning interfaces 1 & 3 of the I2C.
 * For each, trying to write at an address to see if there is a peripheral.
 */

/**Include***/
#include "pmsis.h"
#include <stdio.h>
#include <stdint.h>

// Global variable of the I2C device
static struct pi_device i2c_device;

// With addresses on 7 bits, we can have 128 peripherals maximum, per interface.
// See I2C documentation for more details.
#define MAX_PERIPHERALS 128

/* I2C init function
 *
 * Before opening I2C bus, you must:
 * - Set pads for the 2 lines SDA and SCL. Depend of the i2c's interface.
 * - Create an I2C configuration *pi_i2c_conf_t*:
 *     - pi_i2c_conf_init call: this will set all fields of the configuration structure
 *     with a default value.
 *         See documentation for more details.
 *     - Personalizing configuration fields.
 *     - pi_open_from_conf call to attach the configuration to the device.
 * Then, you can open the device.
 */
static uint8_t i2c_init(uint8_t itf, uint8_t addr)
{
```

(continues on next page)

(continued from previous page)

```

if (itf == 1)
{
    // Setting pads 42 & 43 to Alternate 0 function to enable I2C1 peripheral
    pi_pad_set_function(PI_PAD_042, PI_PAD_FUNC0);
    pi_pad_set_function(PI_PAD_043, PI_PAD_FUNC0);
}
else if (itf == 3)
{
    // Setting pads 46 & 47 to Alternate 2 function to enable I2C3 peripheral
    pi_pad_set_function(PI_PAD_046, PI_PAD_FUNC2);
    pi_pad_set_function(PI_PAD_047, PI_PAD_FUNC2);
}
else if (itf == 0 || itf == 2)
{
    printf("No implementation have been made for this interface.\n");
    printf("Please, feel free to do it or change for interface 1 or 3.\n");
    return 1;
}
else
{
    printf("Wrong interface number. Please scan one of our 4 interfaces (0 to 3).\n"
    ↪");
    return 1;
}

// Creating an I2C configuration
pi_i2c_conf_t i2c_conf;

pi_i2c_conf_init(&i2c_conf);
i2c_conf.itf = itf;
i2c_conf.max_baudrate = 100000;
pi_i2c_conf_set_slave_addr(&i2c_conf, addr << 1, 0);
//pi_i2c_conf_set_wait_cycles(conf, 2048);

pi_open_from_conf(&i2c_device, &i2c_conf);

// Trying to open the device
if (pi_i2c_open(&i2c_device))
{
    printf("I2C open failed\n");
    pmsis_exit(-1);
}

return 0;
}

static void i2c_close(void)
{
    pi_i2c_close(&i2c_device);
}

static uint8_t i2c_write(uint8_t *buf, uint16_t size)

```

(continues on next page)

(continued from previous page)

```

{
    /* Here we write 'buf' to the bus.
     * The transfert starts with a START bit and end with a END bit due to the options given.
     * See documentation for more details.
     */
    int32_t res = pi_i2c_write(&i2c_device, buf, size, PI_I2C_XFER_START | PI_I2C_XFER_STOP);

    // Checking the returned value to see if the transfert went well
    if (res == PI_OK)
    {
        /* We received an ACK (OK) */
        return 1;
    }
    /* We received a NACK (ERROR) */
    return 0;
}

int32_t scan(uint8_t itf)
{
    printf("Scanning interface %d\n", itf);
    uint8_t buf[1];
    uint8_t peripherals[MAX_PERIPHERALS];

    uint8_t found_nb = 0;
    for (uint8_t i = 0; i < MAX_PERIPHERALS; i++)
    {
        if (i2c_init(itf, i))
        {
            return 0;
        }
        peripherals[i] = i2c_write(buf, 1);
        if (peripherals[i] != 0)
        {
            found_nb++;
        }
        i2c_close();
    }

    if (found_nb)
    {
        printf("Interface %d: %d peripherals found:\n", itf, found_nb);
        for (uint8_t i = 0; i < MAX_PERIPHERALS; i++)
        {
            if (peripherals[i] != 0)
            {
                printf("0x%02X, ", i);
            }
        }
        printf("\n");
    }
}

```

(continues on next page)

(continued from previous page)

```

else
{
    printf("Interface %d: No peripheral found\n", itf);
}

return 0;
}

int main(void)
{
    /* We will scan interfaces 1 & 3 of the I2C. */
    int ret = scan(1) + scan(3);
    return ret;
}

```

## I3C

Here are the examples using the PMSIS *I3C*.

### Scan

#### Requirements

Only works on GAP9.

#### Description

Performs a scan of an I3C interface and reports found peripherals.

Done with the accelerometer & gyroscope module LSM6DS0 of ST (<https://www.sparkfun.com/products/18020>), and the color camera 5647 of Omnivision (<https://www.arducam.com/product/arducam-ov5647-standard-raspberry-pi-camera-b0033/>).

#### Hardware setup

- LSM6DS0

SDA & SCL of the module connected to the SDA & SCL pins of the CN10 of the EVK. GND & 3V3 of the module connected to the GND & 1V8 pins of the CN8 of the EVK.

- Camera

Connected to the EVK through the CSI-2 port.

**Commands**

```
# Create a build repository
cmake -B build

# Manage options
cmake --build build --target menuconfig

# Run the example
cmake --build build --target run
```

**Code**

C

```
/*
 * I3C Example - Scan
 *
 * Scanning the I3C's interface.
 * Trying to write at an address to see if there is a peripheral.
 */

/**Include***/
#include "pmsis.h"
#include <stdio.h>
#include <stdint.h>

// Global variable of the I3C device
static struct pi_device i3c_device;

// With addresses on 7 bits, we can have 128 peripherals maximum, per interface.
// See I3C documentation for more details.
#define MAX_PERIPHERALS 128

/* I3C init function
 *
 * Before opening I3C bus, you must:
 * - Set pads for the 2 lines SDA and SCL.
 * - Create an I3C configuration *pi_i3c_conf_t*:
 *     - pi_i3c_conf_init call: this will set all fields of the configuration structure
 *     with a default value.
 *         See documentation for more details.
 *     - Personalizing configuration fields.
 *     - pi_open_from_conf call to attach the configuration to the device.
 * Then, you can open the device.
 */
static void i3c_init(uint8_t addr)
{
```

(continues on next page)

(continued from previous page)

```

// Setting pads 46 & 47 to Alternate 0 function to enable I3C peripheral
pi_pad_set_function(PI_PAD_046, PI_PAD_FUNC0);
pi_pad_set_function(PI_PAD_047, PI_PAD_FUNC0);

// Creating an I3C configuration
pi_i3c_conf_t i3c_conf;

pi_i3c_conf_init(&i3c_conf);
i3c_conf.max_baudrate = 100000;
pi_i3c_conf_set_slave_addr(&i3c_conf, addr);

pi_open_from_conf(&i3c_device, &i3c_conf);

// Trying to open the device
if (pi_i3c_open(&i3c_device))
{
    printf("I3C open failed\n");
    pmsis_exit(-1);
}
}

static void i3c_close(void)
{
    pi_i3c_close(&i3c_device);
}

static uint8_t i3c_write(uint8_t *buf, uint16_t size)
{
    /* Here we write 'buf' to the bus.
     * The transfert starts with a START bit and end with a END bit due to the options given.
     * See documentation for more details.
     */
    int32_t res = pi_i3c_write(&i3c_device, buf, size);

    // Checking the returned value to see if the transfert went well
    if (res == PI_OK)
    {
        /* We received an ACK (OK) */
        return 1;
    }
    /* We received a NACK (ERROR) */
    return 0;
}

int32_t scan()
{
    printf("Scanning interface\n");
    uint8_t buf[1];
    uint8_t peripherals[MAX_PERIPHERALS];

    uint8_t found_nb = 0;
}

```

(continues on next page)

(continued from previous page)

```
for (uint8_t i = 0; i < MAX_PERIPHERALS; i++)
{
    i3c_init(i);

    peripherals[i] = i3c_write(buf, 1);
    if (peripherals[i] != 0)
    {
        found_nb++;
    }
    i3c_close();
}

if (found_nb)
{
    printf("I3C interface: %d peripherals found:\n", found_nb);
    for (uint8_t i = 0; i < MAX_PERIPHERALS; i++)
    {
        if (peripherals[i] != 0)
        {
            printf("0x%02X, ", i);
        }
    }
    printf("\n");
}
else
{
    printf("I3C interface: No peripheral found\n");
}

return 0;
}

int main(void)
{
    /* We will scan the I3C's interfaces. */
    int ret = scan();
    return ret;
}
```

## Scan

### Requirements

Only works on GAP9.

## Description

Performs a transfer of an I3C interface. Sending few bytes on the bus, and read one.

Done with the accelerometer & gyroscope module LSM6DS0 of ST (<https://www.sparkfun.com/products/18020>).

## Hardware setup

- LSM6DS0

SDA & SCL of the module connected to the SDA & SCL pins of the CN10 of the EVK. GND & 3V3 of the module connected to the GND & 1V8 pins of the CN8 of the EVK.

## Commands

```
# Create a build repository
cmake -B build

# Manage options
cmake --build build --target menuconfig

# Run the example
cmake --build build --target run
```

## Code

C

```
/*
 * I3C Example - Transfer
 *
 * Transferring data on the I3C's bus.
 * Trying to write at an address and read a byte.
 */

/**Include****/
#include "pmsis.h"
#include <stdio.h>
#include <stdint.h>

// Global variable of the I3C device
static struct pi_device i3c_device;

/* Address of the peripheral slave on the bus.
 You can find it using the scan example. */
#define SLAVE_ADDR 0x6B

// Size of the data to tranfer
```

(continues on next page)

(continued from previous page)

```

#define DATA_SIZE 3

/* I3C init function
*
* Before opening I3C bus, you must:
* - Set pads for the 2 lines SDA and SCL.
* - Create an I3C configuration *pi_i3c_conf_t*:
*   - pi_i3c_conf_init call: this will set all fields of the configuration structure,
*   with a default value.
*   See documentation for more details.
*   - Personalizing configuration fields.
*   - pi_open_from_conf call to attach the configuration to the device.
* Then, you can open the device.
*/
static void i3c_init(uint8_t addr)
{
    // Setting pads 46 & 47 to Alternate 0 function to enable I3C peripheral
    pi_pad_set_function(PI_PAD_046, PI_PAD_FUNC0);
    pi_pad_set_function(PI_PAD_047, PI_PAD_FUNC0);

    // Creating an I3C configuration
    pi_i3c_conf_t i3c_conf;

    pi_i3c_conf_init(&i3c_conf);
    i3c_conf.max_baudrate = 100000;
    pi_i3c_conf_set_slave_addr(&i3c_conf, addr);

    pi_open_from_conf(&i3c_device, &i3c_conf);

    // Trying to open the device
    if (pi_i3c_open(&i3c_device))
    {
        printf("I3C open failed\n");
        pmsis_exit(-1);
    }
}

static void i3c_close(void)
{
    pi_i3c_close(&i3c_device);
}

static uint8_t i3c_write(uint8_t *buf, uint16_t size)
{
    /* Here we write 'buf' to the bus.
     * The transfert starts with a START bit and end with a END bit due to the options
     * given.
     * See documentation for more details.
     */
    int32_t res = pi_i3c_write(&i3c_device, buf, size);
}

```

(continues on next page)

(continued from previous page)

```

// Checking the returned value to see if the transfert went well
if (res == PI_OK)
{
    /* We received an ACK (OK) */
    return 1;
}
/* We received a NACK (ERROR) */
return 0;
}

int main(void)
{
    /* We will transfer some data through the I3C's bus. */
    printf("*****\nTransferring data\n");

    uint8_t tx_buf[DATA_SIZE+1] = {'G', 'W', 'T', '\0'};
    uint8_t *rx_buf = pi_l2_malloc(sizeof(uint8_t));
    uint8_t ret = 0;

    i3c_init(SLAVE_ADDR);

    /* Creating an event to be notified when the tranfer is finished
     ie. we sent and read all the data we wanted to send and read */
    pi_evt_t event;
    pi_evt_sig_init(&event);

    // Listening the bus to catch 1 byte
    pi_i3c_read_async(&i3c_device, rx_buf, 1, &event);

    // Sending few bytes
    ret = i3c_write(tx_buf, DATA_SIZE);
    if (!ret)
    {
        printf("Error when writting at 0x%02X.\n", SLAVE_ADDR);
        goto error;
    }

    // Waiting end of tranfer
    pi_evt_wait(&event);

    /* Once read/write have been done,
     * you can check buffers.
     *
     * Here we will print what we read.
     */
    printf("Byte read: 0x%02X.\n", rx_buf[0]);

    i3c_close();
    return 0;
}

error:
    i3c_close();

```

(continues on next page)

(continued from previous page)

```
    return -1;  
}
```

## I2S

Here are some examples using the *I2S* API.

### Input

#### Requirements

Running on board only:

1. Audio Input on SAI0 (CN4) in I2S with 44100 sample rate, you can change it in i2s\_config.h
2. The audio output has been configured to use the both AK4332 DAC, which is on our audio add-on board, for stereo audio output.
3. For having a better audio quality, we are using an external clock of 11289600 Hz from our external bluetooth device, which is connected on the pin2.

### Description

This example shows you how to use I2S in -> memory -> I2S Out without using SFU. This example can be executed on both GAP9 EVK and GVSOC

### Code

#### PDM 4 mics input to file

#### Requirements

This example requires the Audio Add-On board.

### Description

This example shows how to use I2S and SFU to retrieve data from the four TDK T3902 microphones onto the Audio Add-on boards. The TDK microphones input PDM data to SFU which converts to 48 KHz PCM and save onto 4 different files, one for each microphone.

**Code**

C

```
/*
 * Copyright (C) 2022 GreenWaves Technologies
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include "pmsis.h"

#include "Graph_L2_Descr.h"
#include "SFU_RT.h"
#include "Gap.h"
#include "wav_out.h"
#include "testbench.h"

#define DURATION_US    2000000
#define SAMPLING_RATE 48000
#define WORD_SIZE 32
#define SAI_ITF          (1)

#define BUFF_SIZE (256*1024)

#define SAI_ID           (48)
#define SAI_SCK(itf)      (48+(itf*4)+0)
#define SAI_WS(itf)       (48+(itf*4)+1)
#define SAI_SDI(itf)      (48+(itf*4)+2)
#define SAI_SDO(itf)      (48+(itf*4)+3)

#define MIC_A_R1 0
#define MIC_A_L1 1
#define MIC_B_R1 2
#define MIC_B_L1 3

/*
 * On the Audio Add-On board there are 4 mics:
 ****
 *
 *      MicA - L1      MicA - R1      *
 *      SFU Channel 1   SFU Channel 0   *
 */

```

(continues on next page)

(continued from previous page)

```

/*
*      MicB - L1          MicB - R1          *
*      SFU Channel 3      SFU Channel 2      *
*                                         *
***** */

*/



static int open_i2s_PDM(struct pi_device *i2s, unsigned int SAIIn, unsigned int Frequency,
→ unsigned int Polarity, unsigned int Diff)
{
    struct pi_i2s_conf i2s_conf;
    pi_i2s_conf_init(&i2s_conf);

    // polarity: b0: SDI: slave/master, b1:SDO: slave/master           1:RX, 0:TX
    i2s_conf.options |= PI_I2S_OPT_REF_CLK_FAST;
    i2s_conf.frame_clk_freq = Frequency;                                // In pdm mode, the
→ frame_clk_freq = i2s_clk
    i2s_conf.itf = SAIIn;                                                 // Which sai interface
    i2s_conf.format |= PI_I2S_FMT_DATA_FORMAT_PDM;                      // Choose PDM mode
    i2s_conf.pdm_polarity = Polarity;                                     // 2b'11 slave on both SDI
→ and SDO (SDO under test)
    i2s_conf.pdm_diff = Diff;                                            // Set differential mode on
→ pairs (TX only)

//     i2s_conf.options |= PI_I2S_OPT_EXT_CLK;                         // Put I2S CLK in input mode
→ for safety

    pi_open_from_conf(i2s, &i2s_conf);

    if (pi_i2s_open(i2s))
        return -1;

    pi_pad_set_function(SAI_SCK(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SDI(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SDO(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_WS(SAI_ITF), PI_PAD_FUNC0);

    return 0;
}

int main(void)
{
    SFU_uDMA_Channel_T *ChanOutCtxt0;
    SFU_uDMA_Channel_T *ChanOutCtxt1;
    SFU_uDMA_Channel_T *ChanOutCtxt2;
    SFU_uDMA_Channel_T *ChanOutCtxt3;

    struct pi_device i2s;

    // Configure PDM for 4 inputs
    if (open_i2s_PDM(&i2s, SAI_ITF, 3072000, 3, 0)) return -1;
}

```

(continues on next page)

(continued from previous page)

```

StartSFU(0, 1);

    ChanOutCtx0 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer0 = pi_l2_malloc(BUFF_SIZE);
if (buffer0 == NULL) return -1;

    ChanOutCtx1 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer1 = pi_l2_malloc(BUFF_SIZE);
if (buffer1 == NULL) return -1;

    ChanOutCtx2 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer2 = pi_l2_malloc(BUFF_SIZE);
if (buffer2 == NULL) return -1;

    ChanOutCtx3 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer3 = pi_l2_malloc(BUFF_SIZE);
if (buffer3 == NULL) return -1;

// Get in and out uDMA channels
SFU_Allocate_uDMA_Channel(ChanOutCtx0, 0, &SFU_RTD(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtx0, buffer0, BUFF_SIZE);

SFU_Allocate_uDMA_Channel(ChanOutCtx1, 0, &SFU_RTD(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtx1, buffer1, BUFF_SIZE);

SFU_Allocate_uDMA_Channel(ChanOutCtx2, 0, &SFU_RTD(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtx2, buffer2, BUFF_SIZE);

SFU_Allocate_uDMA_Channel(ChanOutCtx3, 0, &SFU_RTD(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtx3, buffer3, BUFF_SIZE);

// Connect Channels to SFU
SFU_GraphConnectIO(SFU_Name(Graph), Out0), ChanOutCtx0->ChannelId, 0, &SFU_
RTD(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In0), SAI_ITF, MIC_A_R1, &SFU_RTD(Graph));

SFU_GraphConnectIO(SFU_Name(Graph), Out1), ChanOutCtx1->ChannelId, 0, &SFU_
RTD(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In1), SAI_ITF, MIC_A_L1, &SFU_RTD(Graph));

SFU_GraphConnectIO(SFU_Name(Graph), Out2), ChanOutCtx2->ChannelId, 0, &SFU_
RTD(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In2), SAI_ITF, MIC_B_R1, &SFU_RTD(Graph));

SFU_GraphConnectIO(SFU_Name(Graph), Out3), ChanOutCtx3->ChannelId, 0, &SFU_
RTD(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In3), SAI_ITF, MIC_B_L1, &SFU_RTD(Graph));

// Let the microphone start (20ms in datasheet) before capturing
pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL);
pi_time_wait_us(30000);

```

(continues on next page)

(continued from previous page)

```
// Load and start Graph
SFU_StartGraph(&SFU_RTD(Graph));

printf("Starting\n");
pi_time_wait_us(DURATION_US);
pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_STOP, NULL);
printf("Stopped\n");

char path[50];
printf("Saving MIC_A_R1 output to file...\n");
sprintf(path, "../.../out_file_MIC_A_R1.wav");
dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 1, BUFF_SIZE);
dump_wav_write(buffer0, BUFF_SIZE);
dump_wav_close();

printf("Saving MIC_A_L1 output to file...\n");
sprintf(path, "../.../out_file_MIC_A_L1.wav");
dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 1, BUFF_SIZE);
dump_wav_write(buffer1, BUFF_SIZE);
dump_wav_close();

printf("Saving MIC_B_R1 output to file...\n");
sprintf(path, "../.../out_file_MIC_B_R1.wav");
dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 1, BUFF_SIZE);
dump_wav_write(buffer2, BUFF_SIZE);
dump_wav_close();

printf("Saving MIC_B_L1 output to file...\n");
sprintf(path, "../.../out_file_MIC_B_L1.wav");
dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 1, BUFF_SIZE);
dump_wav_write(buffer3, BUFF_SIZE);
dump_wav_close();

return 0;
}
```

Makefile

```
APP = Test

include $(RULES_DIR)/pmsis_defs.mk

APP_SRCS +=
APP_SRCS += test.c wav_out.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(SFU_RUNTIME)/SFU_RT.
←c
APP_CFLAGS += -Os -g -I$(TARGET_BUILD_DIR) -I$(SFU_RUNTIME)/include
APP_LDFLAGS += -Os -g

CONFIG_I2S=1
```

(continues on next page)

(continued from previous page)

```

CONFIG_TESTBENCH=1

$(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.h: $(CURDIR)/
└─Graph/src
    mkdir -p $(TARGET_BUILD_DIR)
    cd $(TARGET_BUILD_DIR) && SFU -i $(CURDIR)/Graph/src -C

graph: $(TARGET_BUILD_DIR)/Graph_L2_Descr.c

build:: graph

clean::
    rm -f out_file_MIC_A_R1.wav out_file_MIC_A_L1.wav out_file_MIC_B_R1.wav out_
└─file_MIC_B_L1.wav

include $(RULES_DIR)/pmsis_rules.mk

```

## PDM 4 mics input to 2 stereo files on PC

### Requirements

This example requires the Audio Add-On board.

### Description

This example shows how to use I2S and SFU to retrieve data from the four TDK T3902 microphones onto the Audio Add-on boards. The TDK microphones input PDM data to SFU which converts to 48 KHz PCM and save onto 2 different stereo files.

### Code

C

```

/*
 * Copyright (C) 2022 GreenWaves Technologies
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.

```

(continues on next page)

(continued from previous page)

```

/*
#include "pmsis.h"

#include "Graph_L2_Descr.h"
#include "SFU_RT.h"
#include "Gap.h"
#include "wav_out.h"
//#include "gaplib/wavIO.h"
//#include "testbench.h"

#define SAMPLING_RATE (48000)
#define WORD_SIZE      32
#define SAI_ITF        (1)

#define SEC_TO_REC     1
#define NUM_CHANS     2
#define BYTE_SAMPLE    (WORD_SIZE/8)
#define DURATION_US   (SEC_TO_REC * 1000000)
#define BUFF_SIZE      (SAMPLING_RATE * SEC_TO_REC * NUM_CHANS *BYTE_SAMPLE)

#define SAI_ID          (48)
#define SAI_SCK(itf)    (48+(itf*4)+0)
#define SAI_WS(itf)     (48+(itf*4)+1)
#define SAI_SDI(itf)    (48+(itf*4)+2)
#define SAI_SDO(itf)    (48+(itf*4)+3)

#define MIC_A_R1 0
#define MIC_A_L1 1
#define MIC_B_R1 2
#define MIC_B_L1 3

PI_L2 static pi_evt_t proc_task;

/*
* On the Audio Add-On board there are 4 mics:
*****
*           *           *
*   MicA - L1       MicA - R1       *
*   SFU Channel 1   SFU Channel 0   *
*           *           *
*           *           *
*   MicB - L1       MicB - R1       *
*   SFU Channel 3   SFU Channel 2   *
*           *           *
***** */

static int open_i2s_PDM(struct pi_device *i2s, unsigned int SAIn, unsigned int Frequency,
→ unsigned int Polarity, unsigned int Diff)

```

(continues on next page)

(continued from previous page)

```

{
    struct pi_i2s_conf i2s_conf;
    pi_i2s_conf_init(&i2s_conf);

    // polarity: b0: SDI: slave/master, b1:SDO: slave/master      1:RX, 0:TX
    i2s_conf.options |= PI_I2S_OPT_REF_CLK_FAST;
    i2s_conf.frame_clk_freq = Frequency;                         // In pdm mode, the
    ↵frame_clk_freq = i2s_clk                                     // Which sai interface
    i2s_conf.itf = SAIn;                                         // Choose PDM mode
    i2s_conf.format |= PI_I2S_FMT_DATA_FORMAT_PDM;               // 2b'11 slave on both SDI
    i2s_conf.pdm_polarity = Polarity;                            // 2b'11 slave on both SDI
    ↵and SDO (SDO under test)                                    // Set differential mode on
    i2s_conf.pdm_diff = Diff;                                    // Put I2S CLK in input mode
    ↵pairs (TX only)                                           // for safety

    // i2s_conf.options |= PI_I2S_OPT_EXT_CLK;                  // Put I2S CLK in input mode
    ↵for safety

    pi_open_from_conf(i2s, &i2s_conf);

    if (pi_i2s_open(i2s))
        return -1;

    pi_pad_set_function(SAI_SCK(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SDI(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SDO(SAI_ITF), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_WS(SAI_ITF), PI_PAD_FUNC0);

    return 0;
}

static void handle_mem_out_end(void *arg)
{
    printf("Recording ended in handler\n");
    pi_evt_push(&proc_task);
}

int main(void)
{
    SFU_uDMA_Channel_T *ChanOutCtxt0;
    SFU_uDMA_Channel_T *ChanOutCtxt1;
    SFU_uDMA_Channel_T *ChanOutCtxt2;
    SFU_uDMA_Channel_T *ChanOutCtxt3;

    struct pi_device i2s;
    pi_evt_sig_init(&proc_task);

    pi_freq_set(PI_FREQ_DOMAIN_FC,      370*1000*1000);
    pi_freq_set(PI_FREQ_DOMAIN_PERIPH, 370*1000*1000);
    pi_freq_set(PI_FREQ_DOMAIN_SFU,    370*1000*1000);
    // Configure PDM for 4 inputs
    if (open_i2s_PDM(&i2s, SAI_ITF, 3072000, 3, 0)) return -1;
}

```

(continues on next page)

(continued from previous page)

```

StartSFU(0, 1);

int buffer_size = BUFF_SIZE;
if(BUFF_SIZE > 1024*1024){
    printf("Buffer too big limiting to 20 bits, 1MB");
    buffer_size = 1024*1024 -1;
}

ChanOutCtxt0 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer0 = pi_l2_malloc(buffer_size);
if (buffer0 == NULL){
    printf("Allocation Error\n");
    return -1;
}
int32_t* p = buffer0;

ChanOutCtxt1 = (SFU_uDMA_Channel_T *) pi_l2_malloc(sizeof(SFU_uDMA_Channel_T));
void *buffer1 = pi_l2_malloc(buffer_size);
if (buffer1 == NULL){
    printf("Allocation Error\n");
    return -1;
}
// Get in and out uDMA channels
SFU_Allocate_uDMA_Channel(ChanOutCtxt0, 0, &SFU_RTID(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtxt0, (void*)buffer0, (uint32_t)buffer_size);
SFU_uDMA_Channel_Callback(ChanOutCtxt0, handle_mem_out_end, ChanOutCtxt0);

SFU_Allocate_uDMA_Channel(ChanOutCtxt1, 0, &SFU_RTID(Graph));
SFU_Enqueue_uDMA_Channel(ChanOutCtxt1, buffer1, buffer_size);

// Connect Channels to SFU
SFU_GraphConnectIO(SFU_Name(Graph), In0, SAI_ITF, MIC_A_L1, &SFU_RTID(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In1, SAI_ITF, MIC_A_R1, &SFU_RTID(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), Out0, ChanOutCtxt0->ChannelId, 0, &SFU_RTID(Graph));

SFU_GraphConnectIO(SFU_Name(Graph), In2, SAI_ITF, MIC_B_L1, &SFU_RTID(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), In3, SAI_ITF, MIC_B_R1, &SFU_RTID(Graph));
SFU_GraphConnectIO(SFU_Name(Graph), Out1, ChanOutCtxt1->ChannelId, 0, &SFU_RTID(Graph));

// Let the microphone start (20ms in datasheet) before capturing
pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_START, NULL);
printf("Start recording...\n");
pi_time_wait_us(20000);

// Load and start Graph
SFU_StartGraph(&SFU_RTID(Graph));

//Waiting for the copy completion

```

(continues on next page)

(continued from previous page)

```

pi_evt_wait(&proc_task);

printf("Stopped\n");
pi_i2s_ioctl(&i2s, PI_I2S_IOCTL_STOP, NULL);
SFU_StopGraph(&SFU_RTD(Graph));
StopSFU(1);

char path[50];
printf("Saving MIC_A_L1 and MIC_A_R1 to stereo output file...\n");
sprintf(path,"../../../../../out_file_MIC_A_L1_R1.wav");

dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 2, buffer_size);
dump_wav_write(buffer0, buffer_size);
dump_wav_close();

printf("Saving MIC_B_L1 and MIC_B_R1 to stereo output file...\n");
sprintf(path,"../../../../../out_file_MIC_B_L1_R1.wav");

//WriteWavToFile(path, 16, 48000, 2, buffer0, buffer_size/2);
dump_wav_open(path, WORD_SIZE, SAMPLING_RATE, 2, buffer_size);
dump_wav_write(buffer1, buffer_size);
dump_wav_close();

return 0;
}

```

Makefile

```

APP = Test

include $(RULES_DIR)/pmsis_defs.mk

APP_SRCS += wav_out.c
APP_SRCS += test.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(SFU_RUNTIME)/SFU_RT.c
##$(GAP_LIB_PATH)/wav_io/wavIO.c
APP_CFLAGS += -Os -g -I$(TARGET_BUILD_DIR) -I$(SFU_RUNTIME)/include -I$(GAP_LIB_PATH) \
    -I$(CURDIR)
APP_LDFLAGS += -Os -g

CONFIG_I2S=1
#CONFIG_TESTBENCH=1

$(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.h: $(CURDIR) \
    -I$(TARGET_BUILD_DIR)/Graph \
        mkdir -p $(TARGET_BUILD_DIR) \
            cd $(TARGET_BUILD_DIR) && SFU -i $(CURDIR)/Graph/src -C

graph: $(TARGET_BUILD_DIR)/Graph_L2_Descr.c

build:: graph

```

(continues on next page)

(continued from previous page)

```
clean:::  
    rm -f out_file_MIC_A_R1.wav out_file_MIC_A_L1.wav  out_file_MIC_B_R1.wav out_  
    ↪file_MIC_B_L1.wav  
  
include $(RULES_DIR)/pmsis_rules.mk
```

## Input

### Requirements

1. 4 PDM input on SAI1 (CN5), which correspond to the 4 PDM Mics on audio add-on board.
2. TDM-4 output on the SAI2 (CN6), configured to 44100 Sample rate with 32 bits word size.
3. For having a better audio quality, we are using an external clock of  $44100 * 512$  Hz as GAP9's fast reference clock, which is connected on the CN0 pin2.

### Description

This example shows you how to use SFU for audio streaming from 4 PDM mics -> TDM-4

### Code

C

```
/*  
 * Copyright (C) 2022 GreenWaves Technologies  
 *  
 * Licensed under the Apache License, Version 2.0 (the "License");  
 * you may not use this file except in compliance with the License.  
 * You may obtain a copy of the License at  
 *  
 *     http://www.apache.org/licenses/LICENSE-2.0  
 *  
 * Unless required by applicable law or agreed to in writing, software  
 * distributed under the License is distributed on an "AS IS" BASIS,  
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 * See the License for the specific language governing permissions and  
 * limitations under the License.  
 */  
  
#include "pmsis.h"  
#include "Graph_L2_Descr.h"  
#include "sfu_pmsis_runtime.h"  
  
#define FREQ_PDM_BIT      ( 3072000 )  
#define SAMPLE_RATE        ( 44100 )  
#define WORD_SIZE          ( 32 )
```

(continues on next page)

(continued from previous page)

```

#define PDM_IN_ITF      ( 1 )
#define TDM_OUT_ITF     ( 2 )

#define NUM_CHANS        ( 4 )          // Prepare 8 TDM Channels, since MCHStreamer
// support only TDM8
#define NB_ACTIVE_CHANS ( 4 )

#define SAI_RX           ( 1 )          // 4 pdm mics on Audio addon board has been
// connected on SAI1
#define SAI_TX           ( 2 )

#define SAI_ID            (48)
#define SAI_SCK(itf)      (48+(itf*4)+0)
#define SAI_WS(itf)       (48+(itf*4)+1)
#define SAI_SD1(itf)      (48+(itf*4)+2)
#define SAI_SDO(itf)      (48+(itf*4)+3)

#define MIC_A_R1 0
#define MIC_A_L1 1
#define MIC_B_R1 2
#define MIC_B_L1 3

// SFU
static pi_sfu_graph_t *sfu_graph;

// SAI used for receiving and sending PDM
static pi_device_t sai_dev_rx;
static pi_device_t sai_dev_tx;

PI_L2 static pi_evt_t proc_task;

/*
* On the Audio Add-On board there are 4 mics:
*****
*      MicA - L1          MicA - R1          *
*      SFU Channel 1       SFU Channel 0       *
*                                         *          *
*                                         *          *
*      MicB - L1          MicB - R1          *
*      SFU Channel 3       SFU Channel 2       *
*                                         *          *
*****
*/
static void sai_pad_init( uint8_t sai_id )
{
    pi_pad_set_function(SAI_SCK(sai_id), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SD1(sai_id), PI_PAD_FUNC0);
    pi_pad_set_function(SAI_SDO(sai_id), PI_PAD_FUNC0);
}

```

(continues on next page)

(continued from previous page)

```

    pi_pad_set_function(SAI_WS(sai_id), PI_PAD_FUNC0);
}

static int open_pdm_for_rx()
{
    int err = 0;

    sai_pad_init(SAI_RX);

    struct pi_i2s_conf i2s_conf;
    pi_i2s_conf_init(&i2s_conf);

    // polarity: b0: SDI: slave/master, b1:SDO: slave/master      1:RX, 0:TX
    i2s_conf.options = PI_I2S_OPT_INT_CLK | PI_I2S_OPT_REF_CLK_FAST;
    i2s_conf.frame_clk_freq = FREQ_PDM_BIT;
    i2s_conf.itf = SAI_RX;                                     // Which sai interface
    i2s_conf.format |= PI_I2S_FMT_DATA_FORMAT_PDM;           // Choose PDM mode
    i2s_conf.pdm_polarity = 0b11;                             // 2b'11 slave on both SDI
    ↪and SDO (SDO under test)
    i2s_conf.pdm_diff = 0b00;                                 // Set differential mode on_
    ↪pairs (TX only)

    pi_open_from_conf(&sai_dev_rx, &i2s_conf);

    if (pi_i2s_open(&sai_dev_rx))
        return -1;

    // Connect to SFU
    pi_sfu_pdm_itf_id_t itf_id[4] ={
        {SAI_RX, 0, 0},
        {SAI_RX, 1, 0},
        {SAI_RX, 2, 0},
        {SAI_RX, 3, 0},
    };
    err += pi_sfu_graph_bind_pdm(sfu_graph, SFU_Name(Graph, In0), &itf_id[0]);
    err += pi_sfu_graph_bind_pdm(sfu_graph, SFU_Name(Graph, In1), &itf_id[1]);
    err += pi_sfu_graph_bind_pdm(sfu_graph, SFU_Name(Graph, In2), &itf_id[2]);
    err += pi_sfu_graph_bind_pdm(sfu_graph, SFU_Name(Graph, In3), &itf_id[3]);
    if (err != 0)
        return -1;

    return 0;
}

static int open_tdm_for_tx()
{
    int err = 0;
    int stream_ch[4];
    struct pi_i2s_conf tdm_conf;
    pi_i2s_conf_init(&tdm_conf);

    sai_pad_init(SAI_TX);
}

```

(continues on next page)

(continued from previous page)

```

tdm_conf.itf = SAI_TX;
tdm_conf.frame_clk_freq = SAMPLE_RATE;
tdm_conf.word_size = WORD_SIZE;
tdm_conf.channels = NUM_CHANS;
tdm_conf.options = PI_I2S_OPT_TDM |           // TDM mode
                  PI_I2S_OPT_EXT_CLK |     // Use external clk
                  PI_I2S_OPT_EXT_WS |     // Use external WS
                  PI_I2S_OPT_FULL_DUPLEX ;

pi_open_from_conf(&sai_dev_tx, &tdm_conf);
if (pi_i2s_open(&sai_dev_tx)
    printf("Failed to open SAI %d in I2S mode\n", SAI_TX);

// Tx slot
pi_sfu_i2s_itf_id_t itf_id = {SAI_TX, 1};
err += pi_sfu_graph_bind_i2s(sfu_graph, SFU_Name(Graph, Out0), &itf_id, &stream_
ch[0]);
err += pi_sfu_graph_bind_i2s(sfu_graph, SFU_Name(Graph, Out1), &itf_id, &stream_
ch[1]);
err += pi_sfu_graph_bind_i2s(sfu_graph, SFU_Name(Graph, Out2), &itf_id, &stream_
ch[2]);
err += pi_sfu_graph_bind_i2s(sfu_graph, SFU_Name(Graph, Out3), &itf_id, &stream_
ch[3]);
if (err != 0)
{
    printf("Unable to bind I2S(SAI: %d) to SFU STREAM block\n", SAI_TX);
    return -1;
}

struct pi_i2s_channel_conf i2s_slot_conf;
pi_i2s_channel_conf_init(&i2s_slot_conf);

for (uint8_t i=0; i<(uint8_t) NB_ACTIVE_CHANS; i++)
{
    i2s_slot_conf.options = PI_I2S_OPT_IS_TX | PI_I2S_OPT_ENABLED;
    i2s_slot_conf.word_size = WORD_SIZE;
    i2s_slot_conf.format = PI_I2S_CH_FMT_DATA_ORDER_MSB | PI_I2S_CH_FMT_DATA_ALIGN_
LEFT | PI_I2S_CH_FMT_DATA_SIGN_NO_EXTEND;
    i2s_slot_conf.stream_id = stream_ch[i];

    if (pi_i2s_channel_conf_set(&sai_dev_tx, i, &i2s_slot_conf))
        return -1;
}

return 0;
}

int main(void)
{
    int err = 0;

```

(continues on next page)

(continued from previous page)

```

// Open SFU device with default frequency
pi_sfu_conf_t conf = { .sfu_frequency=0 };
if (pi_sfu_open(&conf))
    printf("SFU device open failed\n");
printf("SFU activated\n");

// Open graph
sfu_graph = pi_sfu_graph_open(&SFU_RTD(Graph));
if (sfu_graph == NULL)
    printf("SFU graph open failed\n");
printf("Graph opened\n");

// Configure interfaces
err += open_pdm_for_rx();
if (err != 0)
    printf("PDM interface init failed\n");
printf("PDM Rx interface configured\n");

err += open_tdm_for_tx();
if (err != 0)
    printf("TDM interface init failed\n");
printf("TDM tx interface configured\n");

pi_sfu_graph_load(sfu_graph);

pi_i2s_ioctl(&sai_dev_rx, PI_I2S_IOCTL_START, NULL);
pi_i2s_ioctl(&sai_dev_tx, PI_I2S_IOCTL_START, NULL);

while(1)
{
    pi_time_wait_us(5000);
}

pi_i2s_ioctl(&sai_dev_tx, PI_I2S_IOCTL_STOP, NULL);
pi_i2s_ioctl(&sai_dev_rx, PI_I2S_IOCTL_STOP, NULL);

pi_sfu_graph_unload(sfu_graph);

pi_sfu_graph_close(sfu_graph);

return err;
}

```

Makefile

```

APP = Test

include $(RULES_DIR)/pmsis_defs.mk

APP_SRCS += test.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(SFU_RUNTIME)/sfu_pmsis_runtime.
            c

```

(continues on next page)

(continued from previous page)

```

APP_CFLAGS += -Os -g -I$(TARGET_BUILD_DIR) -I$(SFU_RUNTIME)/include -I../../Utils
    ↵$(FLAGS)
APP_LDFLAGS += -Os -g

CONFIG_I2S=1
#CONFIG_TESTBENCH=1

CONFIG_FAST_OSC_FREQUENCY = 22579200
CONFIG_USE_FAST_REF_CLK_PAD=1

$(TARGET_BUILD_DIR)/Graph_L2_Descr.c $(TARGET_BUILD_DIR)/Graph_L2_Descr.h: $(CURDIR)/
    ↵Graph.src
        mkdir -p $(TARGET_BUILD_DIR)
        cd $(TARGET_BUILD_DIR) && SFU -i $(CURDIR)/Graph.src -C

graph: $(TARGET_BUILD_DIR)/Graph_L2_Descr.c

build:: graph

clean::
    rm -f out_file_MIC_A_R1.wav out_file_MIC_A_L1.wav out_file_MIC_B_R1.wav out_
    ↵file_MIC_B_L1.wav

include $(RULES_DIR)/pmsis_rules.mk

```

## PDM INPUT-OUTPUT Example

### Description

This example is meant to run on the GAP9 EVK with the Audio Add-On board.

It takes input samples from one of the PDM microphones on the Audio Add-On board, downsample it in the SFU to 48 or 16 KHz (this can be selected in the Makefile) and store them in L2 memory within a ring buffer. Then the output is then upsampled, converted in PDM and sent to the PDMAmp stereo minijack on the Audio Add-On board.

### Instructions

**Warning:** Pay attention to required jumper settings (see GAP9\_EVK User Manual)

### Exercised hardware

- Audio Add-on board

## SPI

Here are various examples using the PMSIS [SPI](#).

### Multi SPI dummy clock cycle test

#### Description

This test highlights the dummy clock cycle functionality of the GAP9 SPI devices. This function ensures the correct synchronization of the clock signal between the master and the slave during a SPI transfer. This test uses two SPI peripherals of a single GAP9. It initiates a data transfer from a first SPI device “master” to the second “slave” in half duplex and checks the content of the received data. A valid received data buffer makes the test succeed and confirm that dummy clock cycle parameters are correct.

#### Setup

This test has been developed with GAP9 EVK board with FreeRTOS.

#### Gap9 EVK board configuration

SPI1 and SPI2 signals are available on the following GAP9 EVK connectors :

**SPI1:** SCK: Connector CN2, pin 5 (B2/GPIO33) SDI: Connector CN3, pin 5 (E3/GPIO39) SDO: Connector CN2, pin 1 (A1/GPIO38) CS1: Connector CN2, pin 6 (E5/GPIO67)

**SPI2:** SCK: Connector CN6, pin 3 (H1/GPIO56) SDI: Connector CN6, pin 6 (H3/GPIO58) SDO: Connector CN6, pin 2 (H4/GPIO59) CS0: Connector CN6, pin 7 (H2/GPIO57)

For this test, SPI1 is the master and SPI2 is the slave.

Connect the two SPI peripherals this way: SPI1\_SCK <-> SPI2\_SCK SPI1\_SDO <-> SPI2\_SD SPI1\_SD <-> SPI2\_SDO SPI1\_CS1 <-> SPI2\_CS0

To avoid any interferences from the level shifters (U12 to U16) on SPI1 signals, unplug J8 connector to power these shifters down.

#### Warnings

GAP9\_EVK also provide a Mikro-E connector including SPI1 signals access. Here, SPI signals have a voltage shifted by level shifters (SN74AUP2G241) to 1,8V or 3,3V depending on J8 jumper configuration. These shifters are activated by the SPI1 CS0 GPIO (GPIO34). It means that if this test send dummy clock cycle before setting the SPI1\_CS0 to a low logic state, it will not be seen on the Mikro-E connector.

## Panel Control

In multi\_spi.c file, you can modify the dummy clock cycle configuration in two ways:

1. Setting the macro SPI\_MASTER\_DUMMY\_CYCLE will change the number of dummy clock cycle sent. This macro can be set between 2 and 31.
2. Setting the macro SPI\_MASTER\_DUMMY\_CYCLE\_MODE allows you to choose if these clock cycles are sent before the Chip select toggling or after.

## Test results

This test succeeded in the following configurations :

1. Mode : Dummy cycles before chip select. Number : 3 and above.
2. Mode : Dummy cycles after chip select. Number : 8.

## Code

C

```
#include "pmsis.h"

// Spi Master configuration
#define SPI_MASTER_BAUDRATE          20000000
#define SPI_MASTER_WORDSIZE          PI_SPI_WORDSIZE_8
#define SPI_MASTER_ENDIANESS         1
#define SPI_MASTER_POLARITY          PI_SPI_POLARITY_0
#define SPI_MASTER_PHASE              PI_SPI_PHASE_0
#define SPI_MASTER_CS                  1 // CS0_
#define SPI_MASTER_ITF                1 // SPI1_
// peripheral is set as master
#define SPI_MASTER_DUMMY_CYCLE        3
#define SPI_MASTER_DUMMY_CYCLE_MODE   PI_SPI_DUMMY_CLK_CYCLE_BEFORE_CS

// Spi Master pad configuration
#define SPI_MASTER_PAD_SCK            PI_PAD_033 // Ball B2      > CN2 connector,_
// pin 5
#define SPI_MASTER_PAD_CS0             PI_PAD_034 // Ball E4      > CN2 connector,_
// pin 4
#define SPI_MASTER_PAD_SDO             PI_PAD_038 // Ball A1 (MOSI) > CN2 connector,_
// pin 1
#define SPI_MASTER_PAD_SDI             PI_PAD_039 // Ball E3 (MISO) > CN3 connector,_
// pin 5
#define SPI_MASTER_PAD_CS1             PI_PAD_067 // Ball E4      > CN2 connector,_
// pin 4
#define SPI_MASTER_PAD_FUNC            PI_PAD_FUNC0
#define SPI_MASTER_IS_SLAVE           0

// SPI Slave Configuration
#define SPI_SLAVE_BAUDRATE           20000000
```

(continues on next page)

(continued from previous page)

```

#define SPI_SLAVE_WORDSIZE          PI_SPI_WORDSIZE_8
#define SPI_SLAVE_ENDIANESS         1
#define SPI_SLAVE_POLARITY          PI_SPI_POLARITY_0
#define SPI_SLAVE_PHASE              PI_SPI_PHASE_0
#define SPI_SLAVE_CS                  0                                // (GPIO57) Use SPI2 CS0 on_
// GAP9EVK's CN6(7) connector
#define SPI_SLAVE_ITF                2                                // SPI2 peripheral is set as_
// slave
#define SPI_SLAVE_IS_SLAVE           1

// Spi slave pad configuration
#define SPI_SLAVE_PAD_SCK            PI_PAD_056 // Ball H1 > GAP9 EVK CN6 connector,
// pin 5
#define SPI_SLAVE_PAD_CS0             PI_PAD_057 // Ball H2 > GAP9 EVK CN6 connector,
// pin 7
#define SPI_SLAVE_PAD_CS1             PI_PAD_053 // Ball H7 > GAP9 EVK CN5 connector,
// pin 10
#define SPI_SLAVE_PAD_SDO              PI_PAD_059 // Ball H4 > GAP9 EVK CN6 connector,
// pin 2
#define SPI_SLAVE_PAD_SDIN             PI_PAD_058 // Ball H3 > GAP9 EVK CN6 connector,
// pin 6
#define SPI_SLAVE_PAD_FUNC             PI_PAD_FUNC2

#define SPI_NO_OPTION                 0 // No option applied for SPI transfer

// Data size to send
#define CHUNK_SIZE ( 2048 )
#define CHUNKS      ( 100 )
#define CHUNK_SIZE_BITS (CHUNK_SIZE << 3) // Get buffer size expressed in bits
#define BUFFER_SIZE (CHUNK_SIZE * CHUNKS )

static uint8_t debug = 0;
static uint8_t *spi_master_tx_buf;
static uint8_t *spi_master_rx_buf;
static uint8_t *spi_slave_rx_buf;

// GPIO Initialization
// ****
static void spi_master_pad_init()
{
    pi_pad_set_function(SPI_MASTER_PAD_SCK, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_SDO, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_SDIN, SPI_MASTER_PAD_FUNC);
    pi_pad_set_mux_group(SPI_MASTER_PAD_CS1, PI_PAD_MUX_GROUP_SPI1_CS1);
    pi_pad_set_function(SPI_MASTER_PAD_CS1, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_CS0, SPI_MASTER_PAD_FUNC);
}

static void spi_slave_pad_init()
{
    pi_pad_set_function(SPI_SLAVE_PAD_SCK, SPI_SLAVE_PAD_FUNC);
    pi_pad_set_function(SPI_SLAVE_PAD_CS0, SPI_SLAVE_PAD_FUNC);
}

```

(continues on next page)

(continued from previous page)

```

pi_pad_set_function(SPI_SLAVE_PAD_SDO, SPI_SLAVE_PAD_FUNC);
pi_pad_set_function(SPI_SLAVE_PAD_SDI, SPI_SLAVE_PAD_FUNC);
}

// SPI Initialization
*****  

static void spi_master_init(pi_device_t* spi_master, struct pi_spi_conf* spi_master_conf)
{
    pi_assert(spi_master);
    pi_assert(spi_master_conf);

    pi_spi_conf_init(spi_master_conf);
    spi_master_conf->wordsize = SPI_MASTER_WORDSIZE;
    spi_master_conf->big_endian = SPI_MASTER_ENDIANESS;
    spi_master_conf->max_baudrate = SPI_MASTER_BAUDRATE;
    spi_master_conf->polarity = SPI_MASTER_POLARITY;
    spi_master_conf->phase = SPI_MASTER_PHASE;
    spi_master_conf->itf = SPI_MASTER_ITF;
    spi_master_conf->cs = SPI_MASTER_CS;
    spi_master_conf->dummy_clk_cycle = SPI_MASTER_DUMMY_CYCLE;
    spi_master_conf->dummy_clk_cycle_mode = SPI_MASTER_DUMMY_CYCLE_MODE;
    spi_master_conf->is_slave = SPI_MASTER_IS_SLAVE;
    pi_open_from_conf(spi_master, spi_master_conf);

    printf("INFO: Dummy clock cycle number : %d\n", spi_master_conf->dummy_clk_cycle);
    printf("INFO: Dummy clock cycle mode : %d\n", spi_master_conf->dummy_clk_cycle_
mode);
}

static void spi_slave_init(pi_device_t* spi_slave, struct pi_spi_conf* spi_slave_conf)
{
    pi_assert(spi_slave);
    pi_assert(spi_slave_conf);

    pi_spi_conf_init(spi_slave_conf);
    spi_slave_conf->wordsize = SPI_SLAVE_WORDSIZE;
    spi_slave_conf->big_endian = SPI_SLAVE_ENDIANESS;
    spi_slave_conf->max_baudrate = SPI_SLAVE_BAUDRATE;
    spi_slave_conf->polarity = SPI_SLAVE_POLARITY;
    spi_slave_conf->phase = SPI_SLAVE_PHASE;
    spi_slave_conf->itf = SPI_SLAVE_ITF;
    spi_slave_conf->cs = SPI_SLAVE_CS;
    spi_slave_conf->dummy_clk_cycle = SPI_MASTER_DUMMY_CYCLE;
    spi_slave_conf->dummy_clk_cycle_mode = SPI_MASTER_DUMMY_CYCLE_MODE;
    spi_slave_conf->is_slave = SPI_SLAVE_IS_SLAVE;
    pi_open_from_conf(spi_slave, spi_slave_conf);
}

// Test
*****  


```

(continues on next page)

(continued from previous page)

```

/**
 * @brief Slave SPI callback on receiving data.
 *        The context is a notification that block the execution
 *        of the test until the SPI slave receive step is over.
 * @param context End notification.
 */
static void receive_callback(void* context)
{
    if(debug) printf("INFO: Slave SPI receive is finished\n");
    pi_evt_push((pi_evt_t*)context);
}

static void send_callback(void* context)
{
    if(debug) printf("INFO: Master SPI send is finished\n");
}

int main(void)
{
    printf("\n\t *** PMSIS Multi SPI dummy clock cycles example ***\n\n");
    int retval = 0;
    pi_device_t spi_master, spi_slave;
    struct pi_spi_conf spi_master_conf, spi_slave_conf;
    pi_evt_t send_task, receive_task, end_task;

    // Initialize Gap9 EVK GPIOs
    spi_master_pad_init();
    spi_slave_pad_init();

    // Initialize SPIs
    spi_master_init(&spi_master, &spi_master_conf);
    spi_slave_init(&spi_slave, &spi_slave_conf);
    if (pi_spi_open(&spi_master) || pi_spi_open(&spi_slave))
    {
        printf("ERROR: Failed to open SPI peripheral\n");
        retval = -1;
    }
    else
    {
        spi_master_tx_buf = pi_l2_malloc(BUFFER_SIZE);
        spi_master_rx_buf = pi_l2_malloc(BUFFER_SIZE);
        spi_slave_rx_buf = pi_l2_malloc(BUFFER_SIZE);

        // Prepare data buffer to send
        for (int i = 0; i < BUFFER_SIZE; i++)
        {
            spi_master_tx_buf[i] = BUFFER_SIZE - i;
            spi_master_rx_buf[i] = 0;
            spi_slave_rx_buf[i] = 0;
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

for (int i = 0; i < CHUNKS; i++)
{
    // Initialize a notification event to block this
    // process until spi master send and spi slave receive
    // operation are performed.
    pi_evt_sig_init(&end_task);

    // Initiate SPI slave receive
    pi_spi_receive_async(&spi_slave, (spi_slave_rx_buf + (i*CHUNK_SIZE)), CHUNK_
    SIZE_BITS, SPI_NO_OPTION, pi_evt_callback_irq_init(&receive_task, &receive_callback,_
    (void*)&end_task));

    // Initiate SPI master send
    pi_spi_send_async(&spi_master, (spi_master_tx_buf + (i*CHUNK_SIZE)), CHUNK_
    SIZE_BITS, SPI_NO_OPTION, pi_evt_callback_irq_init(&send_task, &send_callback, NULL));

    // Blocking the execution until send and receive steps are done.
    pi_evt_wait(&end_task);
}

printf("master -> slave test done\n");

for (int i = 0; i < CHUNKS; i++)
{
    // Initialize a notification event to block this
    // process until spi master receive and spi slave sent
    // operation are performed.
    pi_evt_sig_init(&end_task);

    // Initiate SPI slave send
    pi_spi_send_async(&spi_slave, (spi_slave_rx_buf + (i*CHUNK_SIZE)), CHUNK_
    SIZE_BITS, SPI_NO_OPTION, pi_evt_callback_irq_init(&send_task, &send_callback, NULL));

    // Initiate SPI master receive
    pi_spi_receive_async(&spi_master, (spi_master_rx_buf + (i*CHUNK_SIZE)),_
    CHUNK_SIZE_BITS, SPI_NO_OPTION, pi_evt_callback_irq_init(&receive_task, &receive_
    callback, (void*)&end_task));

    // Blocking the execution until send and receive steps are done.
    pi_evt_wait(&end_task);
}
printf("slave -> master test done\n");

// Verify received data
for(int i = 0; i < BUFFER_SIZE; i++)
{
    if(spi_master_tx_buf[i] != spi_master_rx_buf[i])
    {
        printf("ERROR: index (%ld), expected (0x%lx), got M:(0x%lx), S:(0x%lx)\n"
        ,
               i, spi_master_tx_buf[i], spi_master_rx_buf[i], spi_slave_rx_buf[i]);
    }
}

```

(continues on next page)

(continued from previous page)

```

        retval = -1;
        break;
    }

    // Testing data shift
    if(retval)
    {
        if(spi_master_tx_buf[1] == spi_slave_rx_buf[0])
        {
            printf("ERROR: First byte not read. All data are shifted by 1 byte\n");
            printf("INFO: Wrong dummy clock cycle configuration\n");
        }
    }
    else
    {
        printf("INFO: Data received successfully\n");
    }
}
printf("\n\t *** End of test ***\n\n");
return retval;
}

```

Makefile

```

APP      = spim_spis_loopback
APP_SRCS = multi_spi.c
APP_CFLAGS =
APP_LDFLAGS =

include $(RULES_DIR)/pmsis_rules.mk

```

## Multi SPI (full duplex) dummy clock cycle test

### Description

This example show you how to handle a full duplex SPI transfer on GAP9 with both SPI Master and SPI Slave. This example use SPI1 as master and SPI2 as slave, with loopback, to communicate each others in full duplex mode.

## Setup

This test has been developed with GAP9 EVK board with FreeRTOS.

### Gap9 EVK board configuration

SPI1 and SPI2 signals are available on the following GAP9 EVK connectors :

**SPI1:** SCK: Connector CN2, pin 5 (B2/GPIO33) SDI: Connector CN3, pin 5 (E3/GPIO39) SDO: Connector CN2, pin 1 (A1/GPIO38) CS1: Connector CN2, pin 6 (E5/GPIO67)

**SPI2:** SCK: Connector CN6, pin 3 (H1/GPIO56) SDI: Connector CN6, pin 6 (H3/GPIO58) SDO: Connector CN6, pin 2 (H4/GPIO59) CS0: Connector CN6, pin 7 (H2/GPIO57)

For this test, SPI1 is the master and SPI2 is the slave.

Connect the two SPI peripherals this way: SPI1\_SCK <-> SPI2\_SCK SPI1\_SDO <-> SPI2\_SD SPI1\_SD <-> SPI2\_SDO SPI1\_CS1 <-> SPI2\_CS0

To avoid any interferences from the level shifters (U12 to U16) on SPI1 signals, unplug J8 connector to power these shifters down.

## Warnings

GAP9\_EVK also provide a Mikro-E connector including SPI1 signals access. Here, SPI signals have a voltage shifted by level shifters (SN74AUP2G241) to 1,8V or 3,3V depending on J8 jumper configuration. These shifters are activated by the SPI1 CS0 GPIO (GPIO34). It means that if this test send dummy clock cycle before setting the SPI1\_CS0 to a low logic state, it will not be seen on the Mikro-E connector.

## Panel Control

In multi\_spi.c file, you can modify the dummy clock cycle configuration in two ways:

1. Setting the macro SPI\_MASTER\_DUMMY\_CYCLE will change the number of dummy clock cycle sent. This macro can be set between 2 and 31.
2. Setting the macro SPI\_MASTER\_DUMMY\_CYCLE\_MODE allows you to choose if these clock cycles are sent before the Chip select toggling or after.

## Code

C

```
#include "pmsis.h"

// Spi Master configuration
#define SPI_MASTER_BAUDRATE          20000000
#define SPI_MASTER_WORDSIZE           PI_SPI_WORDSIZE_8
#define SPI_MASTER_ENDIANESS          1
#define SPI_MASTER_POLARITY           PI_SPI_POLARITY_0
#define SPI_MASTER_PHASE              PI_SPI_PHASE_0
#define SPI_MASTER_CS                  1                                     // CS0
→(GPIO34)
```

(continues on next page)

(continued from previous page)

```

#define SPI_MASTER_ITF           1                                // SPI1_
  ↵peripheral is set as master
#define SPI_MASTER_DUMMY_CYCLE   3
#define SPI_MASTER_DUMMY_CYCLE_MODE PI_SPI_DUMMY_CLK_CYCLE_BEFORE_CS

// Spi Master pad configuration
#define SPI_MASTER_PAD_SCK        PI_PAD_033 // Ball B2          > CN2 connector,_
  ↵pin 5
#define SPI_MASTER_PAD_CS0        PI_PAD_034 // Ball E4          > CN2 connector,_
  ↵pin 4
#define SPI_MASTER_PAD_SDO        PI_PAD_038 // Ball A1 (MOSI) > CN2 connector,_
  ↵pin 1
#define SPI_MASTER_PAD_SDI        PI_PAD_039 // Ball E3 (MISO) > CN3 connector,_
  ↵pin 5
#define SPI_MASTER_PAD_CS1        PI_PAD_067 // Ball E4          > CN2 connector,_
  ↵pin 4
#define SPI_MASTER_PAD_FUNC       PI_PAD_FUNC0
#define SPI_MASTER_IS_SLAVE       0

// SPI Slave Configuration
#define SPI_SLAVE_BAUDRATE       20000000
#define SPI_SLAVE_WORDSIZE        PI_SPI_WORDSIZE_8
#define SPI_SLAVE_ENDIANESS       1
#define SPI_SLAVE_POLARITY        PI_SPI_POLARITY_0
#define SPI_SLAVE_PHASE           PI_SPI_PHASE_0
#define SPI_SLAVE_CS              0                                // (GPIO57) Use SPI2 CS0 on_
  ↵GAP9EVK's CN6(7) connector
#define SPI_SLAVE_ITF             2                                // SPI2 peripheral is set as_
  ↵slave
#define SPI_SLAVE_IS_SLAVE        1

// Spi slave pad configuration
#define SPI_SLAVE_PAD_SCK         PI_PAD_056 // Ball H1 > GAP9 EVK CN6 connector,
  ↵ pin 5
#define SPI_SLAVE_PAD_CS0         PI_PAD_057 // Ball H2 > GAP9 EVK CN6 connector,
  ↵ pin 7
#define SPI_SLAVE_PAD_CS1         PI_PAD_053 // Ball H7 > GAP9 EVK CN5 connector,
  ↵ pin 10
#define SPI_SLAVE_PAD_SDO         PI_PAD_059 // Ball H4 > GAP9 EVK CN6 connector,
  ↵ pin 2
#define SPI_SLAVE_PAD_SDI         PI_PAD_058 // Ball H3 > GAP9 EVK CN6 connector,
  ↵ pin 6
#define SPI_SLAVE_PAD_FUNC        PI_PAD_FUNC2

#define SPI_NO_OPTION             0 // No option applied for SPI transfer
// Data size to send
#define CHUNK_SIZE ( 2048 )
#define CHUNKS      ( 100 )
#define CHUNK_SIZE_BITS (CHUNK_SIZE << 3) // Get buffer size expressed in bits
#define BUFFER_SIZE ( CHUNK_SIZE * CHUNKS )

static uint8_t debug = 0;

```

(continues on next page)

(continued from previous page)

```

static uint8_t *spi_master_tx_buf;
static uint8_t *spi_master_rx_buf;
static uint8_t *spi_slave_tx_buf;
static uint8_t *spi_slave_rx_buf;

// GPIO Initialization
//********************************************************************

static void spi_master_pad_init()
{
    pi_pad_set_function(SPI_MASTER_PAD_SCK, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_SDO, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_SDI, SPI_MASTER_PAD_FUNC);
    pi_pad_set_mux_group(SPI_MASTER_PAD_CS1, PI_PAD_MUX_GROUP_SPI1_CS1);
    pi_pad_set_function(SPI_MASTER_PAD_CS1, SPI_MASTER_PAD_FUNC);
    pi_pad_set_function(SPI_MASTER_PAD_CS0, SPI_MASTER_PAD_FUNC);
}

static void spi_slave_pad_init()
{
    pi_pad_set_function(SPI_SLAVE_PAD_SCK, SPI_SLAVE_PAD_FUNC);
    pi_pad_set_function(SPI_SLAVE_PAD_CS0, SPI_SLAVE_PAD_FUNC);
    pi_pad_set_function(SPI_SLAVE_PAD_SDO, SPI_SLAVE_PAD_FUNC);
    pi_pad_set_function(SPI_SLAVE_PAD_SDI, SPI_SLAVE_PAD_FUNC);
}

// SPI Initialization
//********************************************************************

static void spi_master_init(pi_device_t* spi_master, struct pi_spi_conf* spi_master_conf)
{
    pi_assert(spi_master);
    pi_assert(spi_master_conf);

    pi_spi_conf_init(spi_master_conf);
    spi_master_conf->wordsize = SPI_MASTER_WORDSIZE;
    spi_master_conf->big_endian = SPI_MASTER_ENDIANESS;
    spi_master_conf->max_baudrate = SPI_MASTER_BAUDRATE;
    spi_master_conf->polarity = SPI_MASTER_POLARITY;
    spi_master_conf->phase = SPI_MASTER_PHASE;
    spi_master_conf->itf = SPI_MASTER_ITF;
    spi_master_conf->cs = SPI_MASTER_CS;
    spi_master_conf->dummy_clk_cycle = SPI_MASTER_DUMMY_CYCLE;
    spi_master_conf->dummy_clk_cycle_mode = SPI_MASTER_DUMMY_CYCLE_MODE;
    spi_master_conf->is_slave = SPI_MASTER_IS_SLAVE;
    pi_open_from_conf(spi_master, spi_master_conf);

    printf("INFO: Dummy clock cycle number : %d\n", spi_master_conf->dummy_clk_cycle);
    printf("INFO: Dummy clock cycle mode : %d\n", spi_master_conf->dummy_clk_cycle_
mode);
}

```

(continues on next page)

(continued from previous page)

```

static void spi_slave_init(pi_device_t* spi_slave, struct pi_spi_conf* spi_slave_conf)
{
    pi_assert(spi_slave);
    pi_assert(spi_slave_conf);

    pi_spi_conf_init(spi_slave_conf);
    spi_slave_conf->wordsize = SPI_SLAVE_WORDSIZE;
    spi_slave_conf->big_endian = SPI_SLAVE_ENDIANESS;
    spi_slave_conf->max_baudrate = SPI_SLAVE_BAUDRATE;
    spi_slave_conf->polarity = SPI_SLAVE_POLARITY;
    spi_slave_conf->phase = SPI_SLAVE_PHASE;
    spi_slave_conf->itf = SPI_SLAVE_ITF;
    spi_slave_conf->cs = SPI_SLAVE_CS;
    spi_slave_conf->dummy_clk_cycle = SPI_MASTER_DUMMY_CYCLE;
    spi_slave_conf->dummy_clk_cycle_mode = SPI_MASTER_DUMMY_CYCLE_MODE;
    spi_slave_conf->is_slave = SPI_SLAVE_IS_SLAVE;
    pi_open_from_conf(spi_slave, spi_slave_conf);
}

// Test
*****  

*****

/**
 * @brief Slave SPI callback on receiving data.
 *        The context is a notification that block the execution
 *        of the test until the SPI slave receive step is over.
 * @param context End notification.
 */
static void slave_callback(void* context)
{

    if(debug) printf("INFO: Slave SPI RX & TX is finished\n");
    pi_evt_push((pi_evt_t*)context);
}

static void master_callback(void* context)
{
    if(debug) printf("INFO: Master SPI RX & TX is finished\n");
}

int main(void)
{
    printf("\n\t *** PMSIS Multi SPI dummy clock cycles example ***\n\n");
    int retval = 0;
    pi_device_t spi_master, spi_slave;
    struct pi_spi_conf spi_master_conf, spi_slave_conf;
    pi_evt_t master_task, slave_task, end_task;

    // Initialize Gap9 EVK GPIOs
    spi_master_pad_init();
    spi_slave_pad_init();
}

```

(continues on next page)

(continued from previous page)

```

// Initialize SPIs
spi_master_init(&spi_master, &spi_master_conf);
spi_slave_init(&spi_slave, &spi_slave_conf);
if (pi_spi_open(&spi_master) || pi_spi_open(&spi_slave))
{
    printf("ERROR: Failed to open SPI peripheral\n");
    retval = -1;
}
else
{
    spi_master_tx_buf = pi_l2_malloc(BUFFER_SIZE);
    spi_master_rx_buf = pi_l2_malloc(BUFFER_SIZE);
    spi_slave_tx_buf = pi_l2_malloc(BUFFER_SIZE);
    spi_slave_rx_buf = pi_l2_malloc(BUFFER_SIZE);

// Prepare data buffer to send
for (int i = 0; i < BUFFER_SIZE; i++)
{
    spi_master_tx_buf[i] = BUFFER_SIZE - i;
    spi_master_rx_buf[i] = 0;
    spi_slave_tx_buf[i] = i;
    spi_slave_rx_buf[i] = 0;
}

for (int i = 0; i < CHUNKS; i++)
{
    // Initialize a notification event to block this
    // process until spi master send&receive and spi slave send&receive
    // operation are performed.
    pi_evt_sig_init(&end_task);

    // Initiate SPI slave send & receive
    pi_spi_transfer_async(&spi_slave, (spi_slave_tx_buf + (i*CHUNK_SIZE)), (spi_
    ↵slave_rx_buf + (i*CHUNK_SIZE)), CHUNK_SIZE_BITS, PI_SPI_CS_AUTO, pi_evt_callback_irq_
    ↵init(&slave_task, &slave_callback, (void*)&end_task));

    // Initiate SPI master send & receive
    pi_spi_transfer_async(&spi_master, (spi_master_tx_buf + (i*CHUNK_SIZE)),_
    ↵(spi_master_rx_buf + (i*CHUNK_SIZE)), CHUNK_SIZE_BITS, SPI_NO_OPTION, pi_evt_callback_
    ↵irq_init(&master_task, &master_callback, NULL));

    // Blocking the execution until send and receive steps are done.
    pi_evt_wait(&end_task);
}

printf("master -> slave test done\n");
// Verify received data
for(int i = 0; i < BUFFER_SIZE; i++)
{
    if(spi_master_tx_buf[i] != spi_slave_rx_buf[i] || spi_master_rx_buf[i] !=_
    ↵spi_slave_tx_buf[i])
}

```

(continues on next page)

(continued from previous page)

```

    {
        printf("ERROR: index (%ld), expected M: (0x%lx), S: (0x%lx), got M:(0x
        ↵%lx), S:(0x%lx)\n",
               i, spi_master_tx_buf[i], spi_slave_tx_buf[i],spi_master_rx_buf[i],_
        ↵spi_slave_rx_buf[i]);
        retval = -1;
        break;
    }
}

// Testing data shift
if(!retval)
{
    printf("INFO: Data received successfully\n");
}
printf("\n\t *** End of test ***\n\n");
return retval;
}

```

## Makefile

```

APP      = spim_spis_loopback
APP_SRCS = multi_spi.c
APP_CFLAGS =
APP_LDFLAGS =

include $(RULES_DIR)/pmsis_rules.mk

```

## UART

Here are some examples using *UART*.

### Input

### Requirements

TODO

### Description

This example shows how to use the PMSIS UART API to receive data from a peripheral. It uses both synchronous and asynchronous variants.

## Code

C

```

/* PMSIS includes */
#include "pmsis.h"

/* Variables used. */
#define BUFFER_SIZE      ( 10 )

char l2_in[BUFFER_SIZE];
struct pi_device uart;
volatile uint8_t done = 0;
char *sentence = "Type your name(10 letters) :\n";

void uart_rx_cb(void *arg);
void uart_tx_cb(void *arg);

void uart_rx_cb(void *arg)
{
    pi_evt_t *task = (pi_evt_t *) arg;
    pi_evt_callback_no_irq_init(task, (void *) uart_tx_cb, NULL);
    sentence = (char *) pi_l2_malloc(sizeof(char) * (BUFFER_SIZE + 8));
    sprintf(sentence, "Coucou %s\n", l2_in);
    pi_uart_write_async(&uart, sentence, strlen(sentence), task);
}

void uart_tx_cb(void *arg)
{
    pi_evt_t *task = (pi_evt_t *) arg;
    done++;
    switch (done)
    {
        case 1 :
            pi_evt_callback_no_irq_init(task, (void *) uart_rx_cb, task);
            pi_uart_read_async(&uart, l2_in, BUFFER_SIZE, task);
            break;
        default :
            break;
    }
}

int main(void)
{
    struct pi_uart_conf conf;
    printf("\n\n\t *** PMSIS Uart Input Test ***\n\n");

    /* Init & open uart. */
    pi_uart_conf_init(&conf);
    conf.enable_tx = 1;
    conf.enable_rx = 1;
    conf.baudrate_bps = 115200;
    pi_open_from_conf(&uart, &conf);
}

```

(continues on next page)

(continued from previous page)

```

if (pi_uart_open(&uart))
{
    printf("Uart open failed !\n");
    pmsis_exit(-1);
}

#if defined(UART_FLOW_CONTROL_EMU)
{
    pi_uart_ioctl(&uart, PI_UART_IOCTL_ENABLE_FLOW_CONTROL, NULL);
    printf("Flow control enable\n");
}
#endif /* UART_FLOW_CONTROL_EMU */

/* Write on uart then wait for data from uart. */
#if (ASYNC)
pi_evt_t task;
pi_evt_callback_no_irq_init(&task, (void *) uart_tx_cb, &task);
pi_uart_write_async(&uart, sentence, strlen(sentence), &task);
while (done != 2)
{
    pi_yield();
}
#else
pi_uart_write(&uart, sentence, strlen(sentence));
pi_uart_read(&uart, l2_in, BUFFER_SIZE);
sentence = (char *) pi_l2_malloc(sizeof(char) * (BUFFER_SIZE + 8));
sprintf(sentence, "Coucou %s\n", l2_in);
pi_uart_write(&uart, sentence, strlen(sentence));
#endif /* ASYNC */

pi_uart_close(&uart);

pmsis_exit(0);
}

```

Makefile

```

# User Test
#-----

APP          = test
APP_SRCS     += test_uart_input.c
APP_INC      +=
APP_CFLAGS   += #-DUART_FLOW_CONTROL_EMU

ifeq ($(ASYNC), 1)
APP_CFLAGS   += -DASYNC=1
endif

include $(GAP_SDK_HOME)/utils/rules/pmsis_rules.mk

```

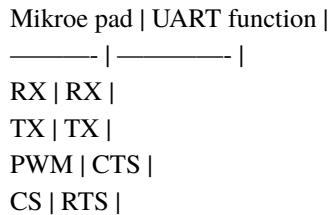
## Input with Timeout

### Requirements

Chip: GAP9

Loopback between RX/TX and CTS/RTS on user board.

Uses UART1 as below:



### Description

This example shows how to use the PMSIS UART API to receive data from a peripheral, with control flow and timeout support. The example will either try to finish the transfer, checking at each timeout if the transfer is actually progressing, or fail if transfer stops to progress.

### Code

C

```

/* PMSIS includes */
#include "pmsis.h"

/* Variables used. */
#define BUFFER_SIZE      ( 10 )

char l2_in[BUFFER_SIZE];
struct pi_device uart;
char *sentence = "HelloWorld !"; // A char* of size >= BUFFER_SIZE

static int8_t enable_3v3_periph()
{
    #define GPIO_3V3_PERIPH ( PI_PAD_035 )

    pi_device_t gpio_3v3_periph;
    struct pi_gpio_conf gpio_conf;

    pi_gpio_conf_init(&gpio_conf);
    gpio_conf.port = GPIO_3V3_PERIPH >> 5;
    pi_open_from_conf(&gpio_3v3_periph, &gpio_conf);
}

```

(continues on next page)

(continued from previous page)

```

if (pi_gpio_open(&gpio_3v3_periph)) return -1;

/* set pad to gpio mode */
pi_pad_set_function(GPIO_3V3_PERIPH, PI_PAD_FUNC1);

/* configure gpio output */
pi_gpio_pin_configure(&gpio_3v3_periph, GPIO_3V3_PERIPH, PI_GPIO_OUTPUT);

pi_gpio_pin_write(&gpio_3v3_periph, GPIO_3V3_PERIPH, 1);

/* wait some time to let voltage rise */
pi_time_wait_us(5000);

return 0;
}

static void setup_fc_pads()
{
    #define GPIO_UART1_RTS ( PI_PAD_034 ) // MIKROE CS
    #define GPIO_UART1_CTS ( PI_PAD_067 ) // MIKROE PWM

    pi_pad_set_function(GPIO_UART1_RTS, PI_PAD_FUNC0);
    pi_pad_set_function(GPIO_UART1_CTS, PI_PAD_FUNC0);

    pi_pad_set_mux_group(GPIO_UART1_RTS, PI_PAD_MUX_GROUP_UART1_RTS);
    pi_pad_set_mux_group(GPIO_UART1_CTS, PI_PAD_MUX_GROUP_UART1_CTS);
}

static void setup_pads()
{
    #define GPIO_UART1_RX ( PI_PAD_044 )
    #define GPIO_UART1_TX ( PI_PAD_045 )

    pi_pad_set_function(GPIO_UART1_RX, PI_PAD_FUNC0);
    pi_pad_set_function(GPIO_UART1_TX, PI_PAD_FUNC0);

    pi_pad_set_mux_group(GPIO_UART1_RX, PI_PAD_MUX_GROUP_UART1_RX);
    pi_pad_set_mux_group(GPIO_UART1_TX, PI_PAD_MUX_GROUP_UART1_TX);

    setup_fc_pads();
}

int main(void)
{
    printf("\n\n\t *** PMSIS Uart Input Timeout Test ***\n\n");
    printf("Enter test\n");

    printf("Enable 3V3 PERIPH\n");
    if (enable_3v3_periph())
    {
        printf("3V3 PERIPH enable failed\n");
        pmsis_exit(-1);
    }
}

```

(continues on next page)

(continued from previous page)

```

}

printf("Setup pads\n");
setup_pads();

printf("Launch test\n");

int32_t errors = 0;
struct pi_uart_conf conf;

/* Init & open uart. */
pi_uart_conf_init(&conf);
conf.uart_id      = 1;
conf.enable_tx    = 1;
conf.enable_rx    = 1;
conf.baudrate_bps = 9600;
conf.use_ctrl_flow = 1; // MANDATORY with timeout !
// Change rts_pad. By default it is set to be used with uart0.
conf.rts_pad      = PI_PAD_034; // MIKROE CS.
pi_open_from_conf(&uart, &conf);
if (pi_uart_open(&uart))
{
    printf("Uart open failed !\n");
    pmsis_exit(-1);
}

uint32_t timeout_us = 50000;
// Timeout RXTX ==> timeout is reset each time a byte pass on the link
int32_t timeout_id = pi_udma_timeout_alloc(PI_UDMA_TIMEOUT_MODE_RXTX);
if (timeout_id == -1)
{
    printf("Timeout alloc failed !\n");
    pmsis_exit(-2);
}
// Attach the hw timeout to uart channel
pi_uart_ioctl(&uart, PI_UART_IOCTL_ATTACH_TIMEOUT_RX, (void *) timeout_id);

// WRITE only few bytes and ask to read more to force timeout to occur
uint8_t first_send_size = 4;
pi_uart_write(&uart, sentence, first_send_size);

// -----
// READ, with timeout, use HW timeout + SW timeout in case no bytes ever come
pi_evt_t timeout_evt;

// ----- SYNCRHONE VERSION -----
// NO initialisation of the timeout_evt
// errors = pi_uart_read_timeout(&uart, l2_in, BUFFER_SIZE, timeout_us,
//                               &timeout_evt, PI_UART_TIMEOUT_FLAG_SW_HW);
// ----- END SYNCRHONE VERSION -----

```

(continues on next page)

(continued from previous page)

```

// ----- ASYNCHRONE VERSION -----
pi_evt_sig_init(&timeout_evt);
pi_evt_t sw_timeout_evt;
uart_sw_timeout_cb_arg_t sw_timeout_cb_args;
uart_sw_timeout_arg_t sw_timeout_args;
sw_timeout_args.sw_timeout_task = &sw_timeout_evt;
sw_timeout_args.cb_args = &sw_timeout_cb_args;
pi_uart_read_timeout_async(&uart, l2_in, BUFFER_SIZE, timeout_us, &timeout_evt,
                           PI_UART_TIMEOUT_FLAG_SW_HW, (void *) &sw_timeout_args);
pi_evt_wait(&timeout_evt);
pi_evt_cancel_delayed_us(&sw_timeout_evt); // Pay attention to this call. MANDATORY
errors = pi_evt_status_get(&timeout_evt);
// ----- END ASYNCHRONE VERSION -----
// -----



if (errors == PI_ERR_TIMEOUT)
{
    uint32_t prev_bytes_left = BUFFER_SIZE;
    // get bytes left at the moment of timeout irq
    uint32_t bytes_left = pi_uart_get_bytes_left(&timeout_evt);
    printf("Timeout, with bytes left at the moment of timeout = 0x%x\n", bytes_left);

    // The following will pause properly the transfer, ensuring no bytes is lost
    // No transfer will happen until a pi_uart_resume
    pi_uart_pause(&uart, RX_CHANNEL, &timeout_evt);
    bytes_left = pi_uart_get_bytes_left(&timeout_evt);
    printf("bytes left at the moment of pause: 0x%x\n", bytes_left);

    // Write bytes left and finish the read (without timeout)
    pi_uart_write(&uart, sentence+first_send_size, bytes_left);
    errors = pi_uart_resume(&uart, RX_CHANNEL, &timeout_evt);

    //
    // The following while loop shows how to handle "recursively" read/resume timeout
    //
    // while (errors == PI_ERR_TIMEOUT)
    // {
    //     prev_bytes_left = bytes_left;
    //     bytes_left = pi_uart_get_bytes_left(&timeout_evt);
    //     if (bytes_left < prev_bytes_left)
    //     {
    //         printf("Timeout, with bytes left at the moment of timeout = 0x%x\n", bytes_left);
    //
    //         pi_uart_pause(&uart, RX_CHANNEL, &timeout_evt);
    //         bytes_left = pi_uart_get_bytes_left(&timeout_evt);
    //         printf("bytes left at the moment of pause: %x\n", bytes_left);
    //
    //         errors = pi_uart_resume_timeout(&uart, RX_CHANNEL, &timeout_evt,
    //                                         1000, PI_UART_TIMEOUT_FLAG_SW_HW);
    //     }
    // }
}

```

(continues on next page)

(continued from previous page)

```

//      }
//      else
//      {
//          printf("transfer not progressing, aborting\n");
//          // The following will kill the current transfer, but no bytes
//          // will be lost. Next transfer, if it exists, will start right away.
//          pi_uart_abort(&uart, RX_CHANNEL, &timeout_evt);
//          errors = -1;
//          break;
//      }
//  }

if (errors)
{
    printf("Transfer ended with error: %d\n", errors);
    printf("FAILURE !\n");
}
else
{
    printf("Read %d bytes: %s\n", strlen(l2_in), l2_in);
    printf("SUCCESS !\n");
}
else if (errors)
{
    printf("UART read timeout failed with error: %d\n", errors);
    printf("FAILURE !\n");
}
else
{ /* This case should never happen since we write 4 bytes and ask to read
   BUFFER_SIZE bytes...
   timeout should be reached then (errors == PI_ERR_TIMEOUT). */
    printf("Timeout does not occurred. Unwanted event.\n");
    printf("FAILURE !\n");
    errors = -1;
}

pi_uart_close(&uart);

pmsis_exit(errors);
}

```

Makefile

```

# User Test
#-----

APP           = test
APP_SRCS      += test_uart_input.c
APP_INC       +=
APP_CFLAGS    +=

```

(continues on next page)

(continued from previous page)

```
ifeq ($(ASYNC), 1)
APP_CFLAGS      += -DASYNC=1
endif

include $(GAP_SDK_HOME)/utils/rules/pmsis_rules.mk
```

## UART loopback

### Description

This example shows how to use the PMSIS UART API to transmit and receive data from a peripheral in a loopback way, i.e. the same platform board sends and receive the data from the two TX and RX interfaces.

### Requirements

In this example we use the UART 0 interface, without flow control. Therefore you need to connect the pins 1 and 4 of CN7 together.

### Commands

```
# Create a build repository
cmake -B build

# Manage options
cmake --build build --target menuconfig

# Run the example
cmake --build build --target run
```

### Code

C

```
/* UART loopback example
*
* On a board, setting an interface of uart and communicating by sending
* data from the TX channel to the RX channel.
*/

#include <stdio.h>
#include <stdint.h>
#include <pmsis.h>
```

(continues on next page)

(continued from previous page)

```

#define BUFFER_SIZE 256

PI_L2 uint8_t tx_buffer[BUFFER_SIZE];
PI_L2 uint8_t rx_buffer[BUFFER_SIZE];

struct pi_device uart;

int main(void)
{
    printf("\n*****\nUART loopback example\n*****\n");

    /* Setting the uart interface (here the 0 one), with its TX and RX channels.
     * For this, we configure the pads related to the channels.
     */
    pi_pad_set_function(PI_PAD_060, PI_PAD_FUNC0);
    pi_pad_set_function(PI_PAD_061, PI_PAD_FUNC0);

    /* Creating a uart configuration to configure our uart device.
     * Once, the init_conf function has been called, you can change
     * the defaults values by accessing the fields directly.
     * Then, call the open_from_conf to attach the configuration
     * to the device.
     */
    struct pi_uart_conf conf;
    pi_uart_conf_init(&conf);
    conf.enable_tx = 1;
    conf.enable_rx = 1;
    conf.uart_id = 0;
    conf.word_size = PI_UART_WORD_SIZE_8_BITS;
    conf.baudrate_bps = 115200;
    pi_open_from_conf(&uart, &conf);

    // Trying to open the uart device
    if (pi_uart_open(&uart))
    {
        /* An error occurred when opening the device.
         * It can be due to a wrong configuration,
         * or pads setup, mainly.
         */
        return -1;
    }

    /* Initialize some values */
    for (uint16_t i = 0; i < BUFFER_SIZE; i++)
    {
        tx_buffer[i] = i;
    }

    /* Declare an event as a blocking signal.
     * We will attach it to a transfer and wait for it.
     * When we will get the signal back, this will mean that the

```

(continues on next page)

(continued from previous page)

```

    * transfer has finished.
    */
pi_evt_t wait;
pi_evt_sig_init(&wait);

/*
 * We first enqueue a read copy so that we can receive what we will send.
 * This must be asynchronous as the transfer cannot finish before
 * we enqueue the write.
 * Attaching the 'wait' event to the transfer.
 */
pi_uart_read_async(&uart, rx_buffer, BUFFER_SIZE, &wait);

/* This one is not using any event and is thus blocking. */
pi_uart_write(&uart, tx_buffer, BUFFER_SIZE);

/* This will block execution until we have received the whole buffer. */
pi_evt_wait(&wait);

    /* Once the transfer has finished, we can access the data received by calling
the rx_buffer.
     * As an example, we will access these data and compare with the data send.
     */
int errors = 0;
for (uint16_t i = 0; i < BUFFER_SIZE; i++)
{
    //printf("%d: @%p: %x @%p: %x\n", i, &rx_buffer[i], rx_buffer[i], &tx_
buffer[i], tx_buffer[i]);
    if (rx_buffer[i] != tx_buffer[i])
    {
        printf("Error at index %d, expected 0x%x, got 0x%x\n", i, tx_
buffer[i], rx_buffer[i]);
        errors++;
    }
}

if (errors)
{
    printf("Transfer failed, we got %d errors in the data received.\n",_
errors);
}
else
{
    printf("Transfer succeed, all data are correct.\n");
}

return errors;
}

```

## Fast Reference Clock

### Requirements

TODO

### Description

This example shows how to use the PMSIS UART API to receive data from a peripheral. It uses both synchronous and asynchronous variants.

In this example, the UART interface is clocked by the FAST Clock.

### Code

C

```
/* PMSIS includes */
#include "pmsis.h"

/* Variables used. */
#define BUFFER_SIZE      ( 10 )

char l2_in[BUFFER_SIZE];
struct pi_device uart;
volatile uint8_t done = 0;
char *sentence = "Type your name(10 letters) :\n";

int main(void)
{
    struct pi_uart_conf conf;

    /* Init & open uart. */
    pi_uart_conf_init(&conf);
    conf uart_id = 0;
    conf.enable_tx = 1;
    conf.enable_rx = 1;
    conf.baudrate_bps = 115200;
    conf.use_fast_clk = 1;           // Enable the fast clk for uart
    conf.use_ctrl_flow = 1;          // Enable the HW Flow Control
    pi_open_from_conf(&uart, &conf);
    if (pi_uart_open(&uart))
    {
        printf("Uart open failed !\n");
        pmsis_exit(-1);
    }

    /* Write on uart then wait for data from uart. */
    pi_uart_write(&uart, sentence, strlen(sentence));
    pi_uart_read(&uart, l2_in, BUFFER_SIZE);
```

(continues on next page)

(continued from previous page)

```
// Change the periph clock
// If the uart is based on ref clock fast
// uart should works and no need to update the clk div
pi_freq_set(PI_FREQ_DOMAIN_PERIPH, 200*1000*1000);

sentence = "Hello ";
pi_uart_write(&uart, sentence, strlen(sentence));
pi_uart_write(&uart, l2_in, BUFFER_SIZE);

pi_uart_close(&uart);

pmsis_exit(0);
}
```

Makefile

```
# User Test
#-----

APP           = uart_fast_ref_clock
APP_SRCS      += uart_fast_ref_clock.c
APP_INC       +=
APP_CFLAGS    +=

include $(GAP_SDK_HOME)/utils/rules/pmsis_rules.mk
```

## Memory Allocation

### Requirements

No specific requirement.

### Description

This example shows how to use memory allocation on GAP9.

### Code

C

```
/* PMSIS includes */
#include "pmsis.h"

/* Demo utilities includes. */
#include "bsp/bsp.h"
#include "bsp/ram.h"

/* Variables used. */
static uint32_t hyper_buff;
```

(continues on next page)

(continued from previous page)

```

static struct pi_device ram;
static struct pi_default_ram_conf conf;

void test_malloc_cluster(void *arg)
{
    uint32_t coreid = pi_core_id(), clusterid = pi_cluster_id(), args = (uint32_t) arg;
    uint32_t *p = NULL;

    /* Allocate in Cluster L1. */
    for (uint32_t i=0; i<1000; i++)
    {
        p = (uint32_t *) pi_l1_malloc(NULL, 2048);
        if (p == NULL)
        {
            break;
        }
        printf("[%ld, %ld] L1_Malloc : %p\n", clusterid, coreid, p);
    }

    /* Allocate in L2. */
    for (uint32_t i=0; i<1000; i++)
    {
        p = (uint32_t *) pi_l2_malloc(2048*8);
        if (p == NULL)
        {
            break;
        }
        printf("[%ld, %ld] L2_Malloc from Cluster : %p\n", clusterid, coreid, p);
    }

    /* Allocate in Ram. */
    pi_cl_ram_alloc_req_t alloc_req;
    for (uint32_t i=0; i<1000; i++)
    {
        pi_cl_ram_alloc(&ram, 2048*20, &alloc_req);
        pi_cl_ram_alloc_wait(&alloc_req, p);
        if ((*p == 0) || alloc_req.error)
        {
            break;
        }
        printf("[%ld, %ld] Hyperram_Malloc : %lx\n", clusterid, coreid, *p);
    }
}

void master_entry(void *arg)
{
    pi_cl_team_fork(1, test_malloc_cluster, arg);
}

int main(void)
{
    printf("\n\n\t *** PMSIS Malloc Test ***\n\n");
}

```

(continues on next page)

(continued from previous page)

```

printf("Entering main controller\n");
uint32_t coreid = pi_core_id(), clusterid = pi_cluster_id();
uint32_t *p = NULL;

pi_default_ram_conf_init(&conf);
pi_open_from_conf(&ram, &conf);
if (pi_ram_open(&ram))
{
    printf("Error ram open !\n");
    pmsis_exit(-1);
}

#if defined(PMSIS_DRIVERS)
/* Allocate in FC L1. */
for (uint32_t i=0; i<1000; i++)
{
    p = (uint32_t *) pi_fc_l1_malloc(2048);
    if (p == NULL)
    {
        break;
    }
    printf("[%d, %d] FC_Malloc : %p\n", clusterid, coreid, p);
}
#endif /* PMSIS_DRIVERS */

struct pi_device *cluster_dev = (struct pi_device *) pi_l2_malloc(sizeof(struct pi_device));
if (cluster_dev == NULL)
{
    printf("Cluster dev alloc failed !\n");
    pmsis_exit(-2);
}

struct pi_cluster_conf *conf = (struct pi_cluster_conf *) pi_l2_malloc(sizeof(struct pi_cluster_conf));
if (cluster_dev == NULL)
{
    printf("Cluster conf alloc failed !\n");
    pi_l2_free(cluster_dev, sizeof(struct pi_device));
    pmsis_exit(-3);
}
pi_cluster_conf_init(conf);
conf->id = 0;
pi_open_from_conf(cluster_dev, conf);
if (pi_cluster_open(cluster_dev))
{
    printf("Cluster open failed !\n");
    pi_l2_free(cluster_dev, sizeof(struct pi_device));
    pi_l2_free(conf, sizeof(struct pi_cluster_conf));
    pmsis_exit(-4);
}

```

(continues on next page)

(continued from previous page)

```

struct pi_cluster_task *task = (struct pi_cluster_task *) pi_l2_malloc(sizeof(struct pi_cluster_task));
if (task == NULL)
{
    printf("Cluster task alloc failed !\n");
    pi_l2_free(cluster_dev, sizeof(struct pi_device));
    pi_l2_free(conf, sizeof(struct pi_cluster_conf));
    pmsis_exit(-5);
}

pi_cluster_task(task, master_entry, NULL);

/* Sending task to cluster. */
pi_cluster_send_task_to_cl(cluster_dev, task);
pi_l2_free(task, sizeof(struct pi_cluster_task));
/* shut down cluster after task completion. */
pi_cluster_close(cluster_dev);
pi_l2_free(conf, sizeof(struct pi_cluster_conf));
pi_l2_free(cluster_dev, sizeof(struct pi_device));

pi_ram_close(&ram);

printf("Test success !\n");
pmsis_exit(0);
}

```

Makefile

```

# User Test
#-----

APP          = test
APP_SRCS     += test_malloc.c
APP_INC      +=
APP_CFLAGS   +=

include $(RULES_DIR)/pmsis_rules.mk

```

## Multi Thread vs Multi Task

### Requirements

No specific requirement.

## Description

This example shows how to run cluster tasks from the FC with pre/post processing in the FC on GAP9 using both multi-threading (one cluster task per FC thread) and multi-tasking (FC scheduling cluster tasks one after the other using irqs). For each thread/task the application does:

- fc\_preprocess: allocate a buffer in L1
- cluster\_fn: set the buffer values using multiplier/offset
- fc\_postprocess: sum the elements of the buffer and deallocate the buffer

Use the Makefile variable MULTI\_THREAD to select the test to run.

## Code

### Multi Threading

```
/* PMSIS includes */
#include "pmsis.h"

#define SILENT
#ifndef SILENT
#define PRINTF(...) ((void) 0)
#else
#define PRINTF printf
#endif

#define NB_THREADS 8
#define N_ELEM 10

int stacks_size;
struct pi_device cluster_dev;
float results[NB_THREADS];
int cyc_counts[NB_THREADS];

struct test_thread_argument_s
{
    pi_evt_t end_task;
    uint32_t arr_elements;
    uint32_t thread_id;
};

typedef struct
{
    float *a;
    float *b;
    int n_elem;
    int thread_id;
} cl_arg_t;

void cl_fn(cl_arg_t *arg)
{
    PRINTF("Thread %d: Cl Fn\n", arg->thread_id);
```

(continues on next page)

(continued from previous page)

```

for (int i=0; i<arg->n_elem; i++)
    arg->a[i] = arg->a[i] * arg->b[i];
}

void fc_preprocess(float* a, float *b, int n_elem, float m, float o) {
    PRINTF("Thread %d: Preprocess (N=%d M=%f O=%f)\n", (int)o, n_elem, m, o);
    for (int i=0; i<n_elem; i++) {
        a[i] = i*m + o;
        b[i] = (i+o)*m;
    }
}

float fc_postprocess(float *arr, int n_elem) {
    float sum = 0.0;
    for (int i=0; i<n_elem; i++)
        sum += arr[i];
    return sum;
}

void test_thread_entry( void *parameters )
{
    char *taskname = pi_thread_get_name( NULL );
    PRINTF("Starting %s: %d\n", taskname, pi_time_get_us());
    uint32_t pid = 0, cid = 0;
    struct test_thread_argument_s *arg = (struct test_thread_argument_s*) parameters;

    int n_elem = arg->arr_elements;
    float multiplier = arg->thread_id*0.2;
    float offset = arg->thread_id;

    float *a_arr = (float *) pi_ll_malloc(0, sizeof(float) * n_elem);
    float *b_arr = (float *) pi_ll_malloc(0, sizeof(float) * n_elem);

    fc_preprocess(a_arr, b_arr, n_elem, multiplier, offset);

    PRINTF("Thread %d: Preprocessed\n", arg->thread_id);

    cl_arg_t cl_arg = {a_arr, b_arr, n_elem, arg->thread_id};
    struct pi_cluster_task task;
    pi_cluster_task(&task, (void (*)(void *))cl_fn, (void *) &cl_arg);
    void *stacks = pi_cl_ll_scratch_alloc(&cluster_dev, &task, stacks_size);
    pi_cluster_task_stacks(&task, stacks, 1024);
    pi_cluster_send_task(&cluster_dev, &task);
    PRINTF("Thread %d: Finished Cluster Fn\n");

    float sum = fc_postprocess(a_arr, n_elem);

    results[arg->thread_id] = sum;
    PRINTF("Finished %s: %d -> sum = %.2f (suspending...)\n", taskname, pi_time_get_us(),
    ↵ sum);
    pi_task_release(&arg->end_task);
    pi_thread_suspend( NULL );
}

```

(continues on next page)

(continued from previous page)

```

}

int main(void)
{
    printf("\n\n\t *** Multi Thread Test ***\n\n");
    printf("Entering main controller\n");

    /* Configure And open cluster. */
    struct pi_cluster_conf cl_conf;

    stacks_size = 512 * pi_cl_cluster_nb_pe_cores();
    pi_cluster_conf_init(&cl_conf);
    cl_conf.id = 0;
    cl_conf.scratch_size = stacks_size + 0x8000;
    pi_open_from_conf(&cluster_dev, (void *) &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        printf("Cluster open failed !\n");
        pmsis_exit(-7);
    }

    /* Initialize perf counters */
    pi_perf_conf(1<<PI_PERF_CYCLES | 1<<PI_PERF_ACTIVE_CYCLES);
    pi_perf_fc_start();
    int start_act_cyc = pi_perf_fc_read(PI_PERF_ACTIVE_CYCLES);
    int start_cyc = pi_perf_fc_read(PI_PERF_CYCLES);

    void *task_handle[NB_THREADS];
    char task_name[NB_THREADS][24];
    struct test_thread_argument_s test_thread_arg[NB_THREADS];
    for (uint32_t i = 0; i < NB_THREADS; i++)
    {
        sprintf(task_name[i], "Thread%d", i);
        cyc_counts[i] = pi_perf_fc_read(PI_PERF_CYCLES);
        pi_evt_sig_init(&test_thread_arg[i].end_task);
        test_thread_arg[i].arr_elements = N_ELEM;
        test_thread_arg[i].thread_id = i;
        task_handle[i] = pi_thread_create(test_thread_entry,
            (void *) &test_thread_arg[i],
            task_name[i], configMINIMAL_STACK_SIZE*4, tskIDLE_PRIORITY +1);
        if( NULL == task_handle )
        {
            printf("%s is NULL !\n", task_name[i]);
            pmsis_exit(0 - i);
        }
    }

    for(uint32_t i=0; i < NB_THREADS; i++)
    {
        pi_evt_wait(&test_thread_arg[i].end_task);
    }
}

```

(continues on next page)

(continued from previous page)

```

    pi_thread_delete(task_handle[i]);
    cyc_counts[i] = pi_perf_fc_read(PI_PERF_CYCLES) - cyc_counts[i];
}
int total_cyc = pi_perf_fc_read(PI_PERF_CYCLES) - start_cyc;
int total_act_cyc = pi_perf_fc_read(PI_PERF_ACTIVE_CYCLES) - start_act_cyc;

for(uint32_t i=0; i < NB_THREADS; i++)
    printf("Thread%d: -> sum: %.2f (%d Cycles)\n", i, results[i], cyc_counts[i]);
printf("Test success (Total Cycles: %d, Active Cycles: %d)!\n", total_cyc, total_act_
cyc);

pmsis_exit(0);
}

```

## Multi Tasking

```

/* PMSIS includes */
#include "pmsis.h"

#define SILENT
#ifndef SILENT
#define PRINTF(...) ((void) 0)
#else
#define PRINTF printf
#endif

#define NB_THREADS 8
#define N_ELEM 10

typedef struct
{
    float *a;
    float *b;
    int n_elem;
    int task_id;
} cl_arg_t;

static int stacks_size;
static struct pi_device cluster_dev;
static pi_evt_t end_task;
static float results[NB_THREADS];
static cl_arg_t cl_arg[NB_THREADS];
static int cyc_counts[NB_THREADS];
static float *a_arr[NB_THREADS];
static float *b_arr[NB_THREADS];

void cl_fn(cl_arg_t *arg)
{
    PRINTF("Task %d: Cl Fn\n", arg->task_id);
    for (int i=0; i<arg->n_elem; i++)
        arg->a[i] = arg->a[i] * arg->b[i];
}

```

(continues on next page)

(continued from previous page)

```

void fc_preprocess(float* a, float *b, int n_elem, float m, float o) {
    for (int i=0; i<n_elem; i++) {
        a[i] = i*m + o;
        b[i] = (i+o)*m;
    }
}

float fc_postprocess(float *arr, int n_elem) {
    float sum = 0.0;
    for (int i=0; i<n_elem; i++)
        sum += arr[i];
    return sum;
}

void callback_fn(cl_arg_t *arg) {

    int task_id = arg->task_id;
    float sum = fc_postprocess(arg->a, arg->n_elem);
    cyc_counts[task_id] = pi_perf_fc_read(PI_PERF_CYCLES) - cyc_counts[task_id];
    results[task_id] = sum;
    PRINTF("Finished Task%d: %d > sum = %.2f\n", task_id, pi_time_get_us(), sum);
    if (task_id == (NB_THREADS-1)) {
        pi_evt_push(&end_task);
    }
}

int main(void)
{
    printf("\n\n\t *** Multi Tasks Test ***\n\n");
    printf("Entering main controller\n");

    /* Configure And open cluster. */
    struct pi_cluster_conf cl_conf;

    stacks_size = 512 * pi_cl_cluster_nb_pe_cores();
    pi_cluster_conf_init(&cl_conf);
    cl_conf.id = 0;
    cl_conf.scratch_size = stacks_size + 0x8000;
    pi_open_from_conf(&cluster_dev, (void *) &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        printf("Cluster open failed !\n");
        pmsis_exit(-7);
    }

    /* Initialize perf counters */
    pi_perf_conf(1<<PI_PERF_CYCLES | 1<<PI_PERF_ACTIVE_CYCLES);
    pi_perf_fc_start();
    int start_act_cyc = pi_perf_fc_read(PI_PERF_ACTIVE_CYCLES);
}

```

(continues on next page)

(continued from previous page)

```

int start_cyc = pi_perf_fc_read(PI_PERF_CYCLES);

struct pi_cluster_task task[NB_THREADS];
pi_evt_t callback_evt[NB_THREADS];

int n_elem = N_ELEM;
for(uint32_t i=0; i < NB_THREADS; i++) {
    a_arr[i] = (float *) pi_ll_malloc(0, sizeof(float) * n_elem);
    b_arr[i] = (float *) pi_ll_malloc(0, sizeof(float) * n_elem);
}

pi_evt_sig_init(&end_task);

for (int task_id=0; task_id<NB_THREADS; task_id++) {
    PRINTF("Starting Task%d: %d\n", task_id, pi_time_get_us());
    cyc_counts[task_id] = pi_perf_fc_read(PI_PERF_CYCLES);

    float multiplier = task_id*0.2;
    float offset = task_id;
    fc_preprocess(a_arr[task_id], b_arr[task_id], n_elem, multiplier, offset);

    cl_arg[task_id].a = a_arr[task_id];
    cl_arg[task_id].b = b_arr[task_id];
    cl_arg[task_id].n_elem = n_elem;
    cl_arg[task_id].task_id = task_id;

    pi_cluster_task(&task[task_id], (void (*)(void *))cl_fn, (void *) &cl_arg[task_id]);
    void *stacks = pi_cl_ll_scratch_alloc(&cluster_dev, &task[task_id], stacks_size);
    pi_cluster_task_stacks(&task[task_id], stacks, 1024);

    pi_cluster_send_task_async(&cluster_dev, &task[task_id], pi_evt_callback_irq_
->init(&callback_evt[task_id], (void (*)(void *))callback_fn, (void *) &cl_arg[task_id]));
}
pi_evt_wait(&end_task);
int total_cyc = pi_perf_fc_read(PI_PERF_CYCLES) - start_cyc;
int total_act_cyc = pi_perf_fc_read(PI_PERF_ACTIVE_CYCLES) - start_act_cyc;

printf("Finished all\n");
for(uint32_t i=0; i < NB_THREADS; i++)
    printf("Task%d: -> sum: %.2f (%d Cycles)\n", i, results[i], cyc_counts[i]);
printf("Test success (Total Cycles: %d, Active Cycles: %d)!\n", total_cyc, total_act_cyc);

pmsis_exit(0);
}

```

Makefile

```

# User Test
#-----

```

(continues on next page)

(continued from previous page)

```

APP           = test
ifeq ($(MULTI_THREAD), 1)
APP_SRCS      += multi_thread.c
else
ifeq ($(SEQ), 1)
APP_SRCS      += multi_task_seq.c
else
APP_SRCS      += multi_task.c
endif
endif
APP_INC       +=
APP_CFLAGS    += -O3

# Uncomment to disable preemptive scheduling
# NO_PREEMPTION = true

PMSIS_OS = freertos

include $(GAP_SDK_HOME)/utils/rules/pmsis_rules.mk

```

## Performance Counters

### Requirements

None

### Description

This example shows how to use performance counters.

Performance counters can be used to measure the efficiency of your application. Documentation can be found [here](#).

### Code

C

```

/* PMSIS includes */
#include "pmsis.h"

/* Variables used. */
struct pi_device uart;
PI_L2 uint32_t perf_values[ARCHI_CLUSTER_NB_PE];

/* Task executed by cluster cores. */
void cluster_helloworld(void *arg)
{
    pi_perf_conf(1 << PI_PERF_ACTIVE_CYCLES);
    pi_perf_start();
}

```

(continues on next page)

(continued from previous page)

```

uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
printf("[%d %d] Hello World!\n", cluster_id, core_id);

pi_perf_stop();
perf_values[core_id] = pi_perf_read(PI_PERF_ACTIVE_CYCLES);
}

/* Cluster main entry, executed by core 0. */
void cluster_delegate(void *arg)
{
    printf("Cluster master core entry\n");
    /* Task dispatch to cluster cores. */
    pi_cl_team_fork(pi_cl_cluster_nb_cores(), cluster_helloworld, arg);
    printf("Cluster master core exit\n");
}

int main(void)
{
    printf("\n\n\t *** PMSIS HelloWorld ***\n\n");
    pi_perf_conf(1 << PI_PERF_CYCLES | 1 << PI_PERF_ACTIVE_CYCLES);
    pi_perf_start();
    printf("Entering main controller\n");
    uint32_t errors = 0;
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    printf("[%d %d] Hello World!\n", cluster_id, core_id);
    uint32_t fc_perf = pi_perf_read(PI_PERF_ACTIVE_CYCLES);

    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;

    /* Init cluster configuration structure. */
    pi_cluster_conf_init(&cl_conf);
    cl_conf.id = 0; /* Set cluster ID. */
    /* Configure & open cluster. */
    pi_open_from_conf(&cluster_dev, &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        printf("Cluster open failed !\n");
        pmsis_exit(-2);
    }

    /* Prepare cluster task and send it to cluster. */
    struct pi_cluster_task cl_task;

    pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_delegate, NULL));
}

pi_cluster_close(&cluster_dev);

pi_perf_stop();
uint32_t cycles = pi_perf_read(PI_PERF_ACTIVE_CYCLES);

```

(continues on next page)

(continued from previous page)

```

uint32_t tim_cycles = pi_perf_read(PI_PERF_CYCLES);
printf("Perf : %d cycles Timer : %d cycles\n", cycles, tim_cycles);
printf("[%d %d] Perf : %d cycles\n", cluster_id, core_id, fc_perf);
for (uint32_t i = 0; i < (uint32_t) ARCHI_CLUSTER_NB_PE; i++)
{
    printf("[%d %d] Perf : %d cycles\n", 0, i, perf_values[i]);
}

printf("Test success !\n");

pmsis_exit(errors);
}

```

**Makefile**

```

# User Test
#-----

APP          = test
APP_SRCS     += perf.c
APP_INC      +=
APP_CFLAGS   +=

include $(RULES_DIR)/pmsis_rules.mk

```

**Power management**

These examples show how to use important features of GAP processors in order to improve power consumption. To have more information GAP9 low-power modes, look at the [Low-power guide](#).

**Idle Mode****Requirements**

In order to run this example, you must have:

- Chip: GAP9
- Platform: Any board (not yet supported on GVSOC)
- OS: PulpOS (not yet supported on FreeRTOS)

## Description

This example shows how to use the Idle mode. Idle mode gates the clocks of GAP9 for a short amount of time to save power.

The function `pi_soc_idle_us` puts the chip in idle mode for the required amount of time.

## Code

C

```
#include "pmsis.h"

int main(void)
{
    printf("Idle Mode Example\n");

    for (int i = 0; i < 10; i++)
    {
        /* Go to idle mode for some time */
        /* During this mode, GAP9 is clock-gated i.e. halts completely */
        /* When the idle mode finishes, everything continues as before */
        pi_soc_idle_us(10000);
    }

    pmsis_exit(0);
}
```

Makefile

```
# User Test
#-----
APP           = idle_mode
# App sources
APP_SRCS      = idle_mode.c
# App includes
APP_INC        =
# Compiler flags
APP_CFLAGS     =
# Linker flags
APP_LDFLAGS    =
include $(RULES_DIR)/pmsis_rules.mk
```

### Deep Sleep

#### Requirements

In order to run this example, you must have:

- Chip: GAP9
- Platform: GAP9\_EVK board
- OS: Any OS

#### Description

This example shows how to use the Deep Sleep mode in order to save power. It sets up the wakeup source using `pi_pmu_wakeup_control` as RTC. The RTC is set to trigger after a few seconds. The chip then goes to deep sleep using `pi_pmu_domain_state_change`. To have more details about these functions you can refer to the [PMU API documentation](#).

When the RTC triggers, the chip wakes up and blinks the LED for 10 times. It then sets up the deep sleep and RTC wakeup again.

The memory banks are not saved during the deep sleep mode. The code needs to be reloaded when the chip wakes up. Hence it needs to be flashed.

To run this example, follow the [Flashing](#) guide.

For more information about low-power modes, please refer to the [Using low power modes](#) guide.

#### Code

C

```
#include "pmsis.h"

#define PAD_GPIO_LED2      (PI_PAD_086)
#define BLINK_DELAY_US    (500 * 1000)
#define BLINK_ITERATIONS  (10)
#define TIMER_COUNTER_SECONDS (5)

static int blink_led()
{
    pi_device_t gpio_led;
    struct pi_gpio_conf gpio_conf;

    pi_gpio_conf_init(&gpio_conf);

    gpio_conf.port = PAD_GPIO_LED2 / 32;

    pi_open_from_conf(&gpio_led, &gpio_conf);

    if (pi_gpio_open(&gpio_led))
    {
        return -1;
    }
}
```

(continues on next page)

(continued from previous page)

```

}

/* set pad to gpio mode */
pi_pad_set_function(PAD_GPIO_LED2, PI_PAD_FUNC1);

/* configure gpio output */
pi_gpio_flags_e flags = PI_GPIO_OUTPUT;
pi_gpio_pin_configure(&gpio_led, PAD_GPIO_LED2, flags);

/* blink the LED */
for (int i = 0; i < BLINK_ITERATIONS; i++)
{
    pi_gpio_pin_write(&gpio_led, PAD_GPIO_LED2, 1);
    pi_time_wait_us(BLINK_DELAY_US);
    pi_gpio_pin_write(&gpio_led, PAD_GPIO_LED2, 0);
    pi_time_wait_us(BLINK_DELAY_US);
}

return 0;
}

static void go_to_sleep(void)
{
    /* force outputs during sleep to avoid random communications */
    pi_pad_sleep_cfg_force(1);

    /* set wakeup source */
    pi_pmu_wakeup_control(PI_PMU_WAKEUP_RTC, 0);

    /* configure RTC */
    {
        struct pi_rtc_conf conf;
        pi_device_t rtc;
        pi_rtc_conf_init(&conf);
        conf.mode = PI_RTC_MODE_TIMER;
        conf.counter = TIMER_COUNTER_SECONDS;
        pi_open_from_conf(&rtc, &conf);

        if (pi_rtc_open(&rtc))
        {
            pmsis_exit(-1);
        }

        uint32_t repeat = 1;
        pi_rtc_ioctl(&rtc, PI_RTC_TIMER_START, (void*) repeat);
    }

    /* go to deep sleep */
    pi_pmu_domain_state_change(PI_PMU_DOMAIN_CHIP, PI_PMU_DOMAIN_STATE_DEEP_SLEEP, 0);
}

int main(void)

```

(continues on next page)

(continued from previous page)

```
{
    /* Release the outputs that we forced in case we come back from deep sleep */
    pi_pad_sleep_cfg_force(0);

    /* Get boot source */
    switch(pi_pmu_boot_state_get())
    {
        case PI_PMU_DOMAIN_STATE_DEEP_SLEEP:
            {
                blink_led();
            }
            break;
        default:
            break;
    }

    /* enter the deep sleep mode */
    go_to_sleep();

    pmsis_exit(0);
}
```

Makefile

```
# User Test
#-----
APP          = deep_sleep
# App sources
APP_SRCS     = deep_sleep.c
# App includes
APP_INC      =
# Compiler flags
APP_CFLAGS   =
# Linker flags
APP_LDFLAGS  =
# RTC uses the 32k clock
CONFIG_SLOW_OSC=1

include $(RULES_DIR)/pmsis_rules.mk
```

## L2 Banks Management

### Requirements

No specific requirement.

### Description

GAP9's shared L2 memory is consisted with 24 shared banks (65536 Bytes each). Each bank can be powered on/off dynamically in your application, which is controlled by malloc and free.

### Code

C

```
/* PMSIS includes */
#include "pmsis.h"

#define L2_SHARED_BANK_SIZE      (1<<17)
#define NB_SHARED_BANK          (12)

static int InitGPIOProbing(struct pi_gpio_conf *Gpio_conf, struct pi_device *Gpio_port, u
                           unsigned int Gpio)

{
#ifndef __PLATFORM_GVSOC__
    pi_gpio_conf_init(Gpio_conf);
    pi_open_from_conf(Gpio_port, Gpio_conf);
    Gpio_conf->port = (Gpio & PI_GPIO_NUM_MASK) / 32;
    if (pi_gpio_open(Gpio_port)) return -1;
    pi_gpio_pin_configure(Gpio_port, Gpio, PI_GPIO_OUTPUT);
    pi_gpio_pin_write(Gpio_port, Gpio, 0);
    pi_pad_set_function(Gpio, 1);
#endif
    return 0;
}

int main(void)
{
    printf("\n\n\t *** PMSIS Malloc Test ***\n\n");
    printf("Entering main controller\n");

    /* Init a GPIO for power measurement */
#ifndef __PLATFORM_GVSOC__
    struct pi_gpio_conf GpioConf;
    struct pi_device GpioPort;
    unsigned int GPIO = 89, V=1, Vdd = 650;
    pi_pmu_voltage_set(PI_PMU_VOLTAGE_DOMAIN_CHIP, Vdd);

```

(continues on next page)

(continued from previous page)

```

if (InitGPIOProbing(&GpioConf, &GpioPort, GPIO)) pmsis_exit(-1);
#endif

uint8_t *p[NB_SHARED_BANK];

printf("In the beginning, all the banks unused should be off\n");
printf("Dump the L2 free block:\n");
pi_l2_malloc_dump();
pi_time_wait_us(10 * 1000);

printf("Test shared banks\n");
for(int i=0; i<NB_SHARED_BANK; i++)
{
    // Allocate 1 shared bank size, this should turn the next bank on
    printf("Allocate bank %d:\n", i);
    p[i] = pi_l2_malloc(L2_SHARED_BANK_SIZE);
    if (p[i] == NULL)
    {
        printf("Allocation failed, last free chunk is smaller than 128KB. All the_
        banks are powered on\n");
        pi_l2_malloc_dump();
        break;
    }
    printf("allocated %d at 0x%p\n", L2_SHARED_BANK_SIZE, p[i]);

    printf("Dump the L2 free block:\n");
    pi_l2_malloc_dump();
#ifndef __PLATFORM_GVSOC__
    pi_gpio_pin_write(GpioPort, GPIO, V);
    V = V^1;
#endif
    pi_time_wait_us(10 * 1000);
}

printf("Test success !\n");
pmsis_exit(0);
}

```

Makefile

```

# User Test
#-----

APP          = test
APP_SRCS     += l2_banks_management.c
APP_INC      +=
APP_CFLAGS   +=

include $(RULES_DIR)/pmsis_rules.mk

```

## SFU EXAMPLES

### Basic SFU Examples

#### SFU PMSIS example: Dynamic graph cloning

##### Overview

The example shows how to use API `pi_sfu_graph_dynamic_clone`, `pi_sfu_graph_get_gfu_filter_refs`, and `pi_sfu_graph_apply_dynamic_clone_patch` to create cloned graphs at runtime and execute them one after the other.

A cloned graph created by `pi_sfu_graph_dynamic_clone` keeps reference to the L2 configuration of the original graph, but allocates additional memory for its specific elements (GFU filter coefficients and states). For a filter node to be clonable, it must be exposed to the runtime by SfuCfgGen tool. This can be ensured by setting `.Visible` property to 1 in .src file.

`pi_sfu_graph_dynamic_clone` will traverse the graph and collect relevant references to GFU filters/states in a local structure. These elements can be retrieved by the application by `pi_sfu_graph_get_gfu_filter_refs` so that it can modify filter coefficients or states of the clones.

Since all clones still refer to the same base L2 configuration, `pi_sfu_graph_apply_dynamic_clone_patch` must be called for a cloned graph before loading it. This will update relevant fields in base L2 config (pointers to GFU filter coefficients and states).

##### Usage

```
make graph make all -j make run
```

#### SFU PMSIS example: Static graph cloning

##### Overview

The example shows how to create cloned graphs with help of SfuCfgGen and switch between them at runtime.

Prior to build, a SFU\_CloneGraph() sections must be used in .src file to describe which elements of the original graph are to be cloned. SfuCfgGen will then generate SFU\_ClonedRunTimeDescr\_T structures that contain references to cloned filter's coefficients and states.

At runtime, `pi_sfu_graph_clone` is used to create cloned graph and register information provided in SFU\_ClonedRunTimeDescr\_T structures.

Since all clones still refer to the same base L2 configuration, `pi_sfu_graph_apply_clone_patch` must be called for a cloned graph before loading it. This will update relevant fields in base L2 config (pointers to GFU filter coefficients and states) based on references generated by SfuCfgGen.

## Usage

```
make graph make all -j make run
```

### Concurrent graphs described in multiple .src files

The example shows how to configure multiple concurrent graphs that can run in parallel by using several .src files (one file for each graph).

For two or more graphs to run in parallel, it must be ensured that:

- [1] They don't use the same SFU resources (blocks instances, contexts, logical clocks, ...)
- [2] They don't use the same uDMA resources (channel or stream IDs)

To satisfy [1] when using multiple .src files, additional section `Protects` must be defined in each .src file other than the first. This section shall contain first instance of a block/clock that may be used by the graph described in this file. If this is not indicated in `Protects` section, some graphs may be mapped to the same SFU resource, in which case they won't be able to run concurrently. See `graph_2.src` and `graph_3.src` for examples.

[2] is managed by runtime API with the help of the application: the application must indicate which graphs are allowed to share resources. This can be done via `rgroup_id` argument of the `pi_sfu_graph_open()` call (see its documentation for more details).

### Concurrent graphs described in single .src file

The example shows how to configure multiple concurrent graphs that can run in parallel by using one signle .src file.

For two or more graphs to run in parallel, it must be ensured that:  
\* [1] They don't use the same SFU resources (blocks instances, contexts, logical clocks, ...)  
\* [2] They don't use the same uDMA resources (channel or stream IDs)

[1] is automatically satisfied when describing multiple graphs in the same .src file (i.e. having a single run of SFU generator application).

[2] is managed by runtime API with the help of the application: the application must indicate which graphs are allowed to share resources. This can be done via `rgroup_id` argument of the `pi_sfu_graph_open()` call (see its documentation for more details).

### Concurrent graphs described in single .src file

The example shows how to run two sets of graphs on SFU:

- Permanent set: Set of 2 graphs that are started once and run continuously until the end of application
- Switchable set: Set of 3 graphs that run one by one in one-shot manner, each processing one buffer of data on each memory port

Graphs from permanent set are defined in `permanent_graphs.src`. They use non-overlapping SFU resources. In application they are opened in separate resource groups 0 and 1 so that they don't share uDMA resources either.

Graphs from switchable set are defined in `switchable_graph_<x>.src` files. To avoid resource overlap with permanent graphs, the “`Protects`” section must be specified at the top of the file to define starting block instances (or logical clock) that may be used by switchable graphs. In application, graphs from switchable set can share resources between themselves but not with graphs from permanent set, so they are opened in another resource group 2.

## Concurrent graphs described in single .src file

The example shows how to run two sets of graphs on SFU:

- Permanent set: Set of 2 graphs that are started once and run continuously until the end of application
- Switchable set: Set of 3 graphs that run one by one in one-shot manner, each processing one buffer of data on each memory port

Graphs from permanent set are defined in permanent\_graphs.src. They use non-overlapping SFU resources. In application they are opened in separate resource groups 0 and 1 so that they don't share uDMA resources either.

Graphs from switchable set are defined in switchable\_graph\_<x>.src files. To avoid resource overlap with permanent graphs, the "Protects" section must be specified at the top of the file to define starting block instances (or logical clock) that may be used by switchable graphs. In application, graphs from switchable set can share resources between themselves but not with graphs from permanent set, so they are opened in another resource group 2.

## SFU PMSIS example: PDM via FC/L2 to I2S

Graph topology is as follows:

SAI0.PDM.Rx -> SFU.PDM\_IN -> SFU.MEM\_OUT -> L2 -> (FC processing) -> L2 -> SFU.MEM\_IN -> SFU.STREAM\_OUT -> SAI1.I2S.Tx

For this test the continuous memory transfer must be included in build with -DCONFIG\_AUDIO\_SW\_QUEUES.

### Usage

Generate SFU configuration from Graph.src:

`make graph`

Build:

`make all -j`

Run

`make run`

## SFU PMSIS example: PDM to I2S streaming

Audio flow is as follows:

SAI0.PDM.Rx -> SFU.PDM\_IN -> SFU.STREAM\_OUT -> (uDMA stream) -> SAI1.I2S.Tx

### Usage

Generate SFU configuration from Graph.src:

`make graph`

Build:

`make all -j`

Run

```
make run
```

### SFU PMSIS example: ASRC stream to memory

The example shows how to receive incoming stream at some rate (here 48skHz), convert to another rate (here 44.1kHz) with ASRC in automatic mode and feed output to memory buffer.

Input rate is set by configuring SAI0 in I2S mode with frame rate of 48000Hz. Output rate is obtained from FAST\_REF\_CLOCK by SFU's programmable clock divider CLK\_AUD0.

For this configuration to work, a physical source of ASRC output must be explicitly in Graph.src:

```
Asrc1.ClockOut = CLK_AUD0;
```

Frequency divider for CLK\_AUD0 is set in application with line:

```
err = pi_sfu_audio_clock_set(CLK_AUD0, 1, F_OUT);
```

### SFU PMSIS example: PDM to PDM

Audio flow is as follows:

SAI0.PDM.Rx -> SFU.PDM\_IN -> SFU.PDM\_OUT -> SAI1.PDM.Tx

### Usage

Generate SFU configuraion from Graph.src:

```
make graph
```

Build:

```
make all -j
```

Run (without testbench):

```
make run
```

Run (with testbench enabled and testbench client process in another terminal):

In 1st terminal: USE\_GVSOC\_TESTBENCH=1 make run In 2nd terminal: USE\_GVSOC\_TESTBENCH=1 make proxy

Run (with testbench enabled and testbench client process in the same terminal):

```
make run_dual RUN_CMD="make run USE_GVSOC_TESTBENCH=1" PROXY_CMD="make proxy"
```

Check output against reference output given by utils/golden.raw:

```
make check
```

## SFU PMSIS example: GFU reconfiguration

### Overview

The example shows how to switch at runtime between two sets of coefficients of a FIR filter. Two sets of coefficients are predefined in Graph.src file (fir\_filter\_a and fir\_filter\_b).

To synchronize reconfiguration with audio flow, we use splitter to send input audio to a SFU MEM\_OUT port. Then we enqueue uDMA transfers on this port and use callback to be called when desired number of samples have been received. This MEM\_OUT port is used only for timing, so the data received on it is just ignored. CONFIG\_AUDIO\_SW\_QUEUES compilation flag is required here to continuously receive data from this MEM\_OUT port.

### Usage

Generate C-code SFU grpah configuration from Graph.src:

```
make graph
```

Build application:

```
make all -j
```

Run:

```
make run
```

Before building, optionally adjust TRACE\_... macros for verbosity.

## SFU PMSIS example: Save/restore (graph time-sharing)

### Overview

The example shows how to run several independent SFU graphs in time-sharing by saving state of a graph to L2 before switching to the next one.

To save SFU graph state to L2, SFU SAVE command is used.

Since graphs use SFU memory interface, allocation of uDMA channels is required. SFU API manages uDMA channel allocation internally on graph open/close. uDMA channels are not released on save/unload, instead, they are kept to be reused by the next graph to be loaded.

### Usage

Generate C-code SFU grpah configuration from Graph.src:

```
make graph
```

Build application:

```
make all -j
```

Run:

```
make run
```

### TDM8-input ASRC

#### Requirements

Running on board only:

1. a GAP9 EVK

#### Description

This example shows you how to use mem in -> de-interleave with resampler -> asrc -> mem out using SFU. With provided code, input is 24 bits audio at 44100Hz, output is 24 bits at 48000Hz. A script *mix\_input.pl* is provided to generate interleaved inputs at different sampling rate or bitwidth. Be careful that if going above 24 bits, a limiter will have to be placed before ASRC.

The whole code is fully reactive, that is, end of input buffers or output buffers will trigger a callback to reenqueue, until input is fully consumed.

#### Code

#### Stacks

These examples show how to use stack related APIs

#### CMake how-to

#### Test Description:

Run an helloworld on FC and print the current stack usage at the point of Hello

#### Few command examples for:

- gap9\_v2
- freertos

Generate build files (Ninja generator is optional - requires installation of ninja-build package):

```
# without cluster:  
CMAKE_GENERATOR=Ninja cmake -S . -B build
```

Build

```
# Optional: configure the build (io, platform etc)  
cmake --build build --target menuconfig  
# Build the actual binary  
cmake --build build
```

Run

```
# Without traces
cmake --build build --target run
# with fc traces (for other traces, read gvsoc documentation)
runner_args="--trace=fc/insn" cmake --build . --target run
```

Cleanup

```
rm -rf build
```

## WATCHDOG

### Description

A watchdog is kind of a timer that can reset the chip when it gets stuck. This example show how to use it.

#### Pay attention to the following points:

- A limit for the timer exists, and depends on the ref clock you are using.
- If the watchdog timer is triggered, your chip will reset.

Also, we don't have support for watchdog on gvsoc. Therefore, this example can be execute on board only.

### Commands

```
# Create a build repository
cmake -B build

# Manage options
cmake --build build --target menuconfig

# Run the example
cmake --build build --target run
```

### Code

C

```
/*
 * Copyright (C) 2022 GreenWaves Technologies
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms
 * of the BSD license. See the LICENSE file for details.
 */

/* WatchDog example
 *
 * Set a watchdog timer properly and use it.
 */
```

(continues on next page)

(continued from previous page)

```

/* Note:
 * Timer for watchdog is based on RefClk = 24 MHz (Fast) if exist
 *          or      = 32 kHz (Slow) else
 * When using the fast clock, it can be divided. For now,
 * the clock value is 6 MHz.
 */

#include "pmsis.h"

int main()
{
    printf("\n*****\nWhatchDog example\n*****\n");

    /* Setting the timer (in us)
     * And check if it has been set properly (== 0)
     */
    if (pi_watchdog_timer_set(250))
    {
        printf("Error when setting the watchdog timer.");
        /* It can happen if the timer value is too high.
         * Remind that regarding the clock the limit will be different.
         */
        return -1;
    }

    /* Starting the countdown
     * Executing a task, faster than the timer's limit.
     * WatchDog should not be triggered.
     */
    printf("Executing first task. No reset should append.\n");
    pi_watchdog_start();
    pi_time_wait_us(200);
    pi_watchdog_stop();
    printf("Task faster than the watchdog's timer. Done without reset.\n");

    /* In case you try to execute a task that last more than the timer,
     * the latter will be triggered and the watchdog will reset the chip.
     */
    printf("Executing a second task. No reset should append.\n");
    /* To execute a second task that have no attach to the first one (in term of
     * execution time)
     * you have to rearm the timer. You can start it again then.
     * If you do not rearm it the counter will continue where it stopped.
     */
    pi_watchdog_timer_rearm();
    pi_watchdog_start();
    pi_time_wait_us(300);
    pi_watchdog_stop();
    printf("Task faster than the watchdog's timer. Done without reset.\n");

    return 0;
}

```

(continues on next page)

(continued from previous page)

}

## 5.1.2 DSP

These examples show how to use GAP9 to do various Digital Signal Processing including image and sound.

### 5.1.3 Libraries

#### Jpeg Encoder

This is an example of JPEG encoding. It reads the input PGM image from the flash, encodes it to JPEG, and saves it to the workstation using semi-hosting.

You can run this example with CMake. Note: it can be take some time to run. Once it has been run, the output JPEG image can be opened in the build folder with

```
xview imgTest.jpg
```

#### Options

Via menuconfig you can choose:

- the monochrome (default) or the color example;
- to run the encoder on the Gap Cluster (default) or Fabric Controller;
- the JPEG quality (10, 20, 50 default, 90, 95).

#### Wav IO Example

#### Requirements

No specific requirement. This example should run without issue on all chips/boards/OSes.

#### Description

This example reads a WAV file from PC and save it back to a new file.

You can run this on GVSOC or your board:

```
# Run on GVSOC
make clean all run platform=gvsoc

# Run on real board
make clean all run platform=board
```

### 5.1.4 Neural Networks

These examples show how to use GAP9 features and SDK tools to run neural networks on the chip.

#### MNist using NNTool's Python APIs

In this example it is shown how to train a Neural Network on PyTorch for MNist (Training.ipynb).

The trained model is then exported to ONNX and processed in NNTool for deployment on GAP devices (NNTool.ipynb). This notebook quantizes the graph and test the accuracy in a subset of the original testing dataset. It also contains automatic code generation for a template project that is run from NNTool's Python APIs to collect on-device performance.

The same code generation procedure mentioned before has been used to generate the gap9\_project folder. That template has then been expanded to automate the Autotiler Model code generation (nntool\_generate\_model.py and CMakeLists.txt), read images from files and check the predicted class (mnist.c).

NOTE: if targeting GAP8 processor, remember to disable the use\_ne16 option in the quantization of the graph

**TOOLS**

## 6.1 Audio Framework

### 6.1.1 Overview

The Audio Framework (AF) is a collection of tools and libraries that allow efficient development of audio applications on GAP9 platform. It consists of:

- Audio Mapping Tool (AMT): Host tool that takes input graph in .graph (JSON) format and converts it to a fully resolved graph that is fed to Runtime Generator (RTG).
- Runtime Generator (RTG): Host tool that takes fully resolved graph produced by AMT and produces set of source and configuration files that are to be build for GAP9 target.
- Toolchain (TC): A short name for AMT followed by RTG.
- Runtime Library (RNT): Target API that enables user applications to load and run graphs produced by the toolchain.

High-level input graph file can be designed by one of three front ends:

- Graph Designer (GD): Main GUI front end.
- Python API
- C++ API

Graph Designer is principal tool for designing audio graphs, filters and other components. To facilitate testing and automation, graphs can also be created programmaticaly by Python or C++ API provided by component library.

Graph processing flow of the Toolchain is straighforward: input graph created by GD or Python/C++ API is treated by AMT, then RTG.

### 6.1.2 Graph description format

All tools in the Audio Framework use common representation of the graph consisting of nodes, ports, edges, parameters, descriptors, etc. JSON format is used to store input graph and varius intermediate graphs that occur during tool flow. Generally, input graph contains just high-level description with references to black box library components. Then it is elaborated as it passes through the toolchain into fully expanded low level graph.

Following conventions are adopted for file extensions: - Input graph (that is the output of GD or Python/C++ graph generator API) files have .graph extension - Intermediate files (i.e. AMT output or debug dumps) have .json extension - Library component templates have .comp extension

In addition, Graph Designer project file is using extension .gdg to save GUI level graphs. However, note that .gdg graph format is not compatible with AF format presented here. User must use GD command “Publish” to export the graph to AF format.

Applications can complete audio graph data and add their information in a specific object.

### General format

Topological elements of the graph are nodes, ports, edges and links.

- *Node* may represent a simple node (e.g. a filter), a subgraph (composite node) or entire top level graph (root node). In below text all three variants are implied under one term *Node*, unless otherwise specified.
- Node may have one or more input or output *Ports*.
- *Edge* can connect one output port with one input port on the same hierarchical level (horizontally).
- *Link* connects port of a node (graph) with a port of a child node. In other words, it enables vertical connections across the hierarchy.

### Nodes

Basic structure and properties of node, graph and subgraph elements are the same. Some properties (e.g. name) are present in all nodes, while presence of others depends on node type.

```
{  
    "name": "node_name",  
    "template": "relative/path/to/template",  
    "nodes": [],  
    "edges": [],  
    "inputs": [  
        {  
            "name": "input_name",  
            "link": {  
                "name": "node_name",  
                "input": "input_name"  
            }  
        }  
    ],  
    "outputs": [  
        {  
            "name": "output_name"  
        }  
    ],  
    "parameters": [  
        {  
            "name": "parameter_name",  
            "value": 3  
        }  
    ],  
}
```

- **"name"**

Name of the node/graph. Used in tool traces to refer to a node.

- Mandatory: Yes  
Data type: String
- **"template"**  
Relative path to library component file (.comp) that contains full description of this node. If present, AMT will inline referenced component and override its parameters and descriptors with parameters and descriptors given here. Parameters and descriptors that are not known by the component (not present in .comp file) will be dropped.  
Mandatory: No  
Data type: String
- **"inputs"**  
Array of input ports of this node. An input port can contain properties of its own.  
Mandatory: No  
Data type: Array of objects
- **"output"**  
Array of output ports of this node. An output port can contain properties of its own.  
Mandatory: No  
Data type: Array of objects
- **"nodes"**  
Array of immediate child nodes of this node.  
Mandatory: No  
Data type: Array of objects
- **"edges"**  
Array of edges between ports of immediate child nodes of this node.  
Mandatory: No  
Data type: Array of objects
- **"parameters"**  
Array of parameters of a node. A parameter is usually an object containing two attribute-value pairs:  
- "name" : <parameter\_name\_string>  
- "value": <parameter\_value>  
A parameter name must exist in list of supported parameter of this node (component), otherwise the parameter will be dropped by the AMT.  
Normally, parameters are oriented to functional aspects of a node, and all implementations should support the same set of parameters.  
Mandatory: No  
Data type: Array of objects
- **"implementations"**  
Array containing a list of available implementations a node. An implementation object details about implementation (e.g. name, location, descriptors, source code paths).
- **"selected\_implementation"**  
Name of the component implementation to use for this node. List of available implementations of a component is given in component's .comp file.  
Mandatory: No.  
Composite components may not need implementation.

For single components that require implementation, if selected\_implementation is not specified (empty string or keyword “AUTOMATIC”), AMT will pick first from the list.

Data type: String

- **“descriptors”**

Array of descriptors of a node. In structure, descriptors is similar as parameters, with difference that descriptors are more implementation specific properties.

Mandatory: No

Data type: Array of objects

Here is an example of a simple graph named “example” containing one FIR filter node named “fir1” with one input and one output.

```
{
  "inputs": [
    {
      "descriptors": [
        {
          "name": "binding_type",
          "value": "UDMA_STREAM"
        },
        {
          "name": "audio_data_type",
          "value": "INT32"
        },
        {
          "name": "q_precision",
          "value": "Q1.23"
        }
      ],
      "link": {
        "input": "processing",
        "name": "fir1"
      },
      "name": "input",
      "type": "audio"
    }
  ],
  "name": "example",
  "nodes": [
    {
      "descriptors": [
        {
          "name": "instance",
          "value": 0
        },
        {
          "name": "context",
          "value": 0
        }
      ],
      "name": "fir1",
      "parameters": [
        ...
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
{
    "name": "filter_length",
    "value": [
        22
    ]
},
{
    "name": "filter_coefficients",
    "value": "generators/fir_1_filter_coefficients_lowpass83325131.dfl"
},
{
    "name": "input_shift",
    "value": [
        0
    ]
],
"selected_implementation": "sfu_fir",
"template": "filters/fir/fir.comp"
}
],
"outputs": [
{
    "descriptors": [
        {
            "name": "binding_type",
            "value": "UDMA_STREAM"
        },
        {
            "name": "audio_data_type",
            "value": "INT32"
        },
        {
            "name": "q_precision",
            "value": "Q1.23"
        }
    ],
    "link": {
        "name": "fir1",
        "output": "push"
    },
    "name": "output",
    "type": "audio"
}
]
}
```

## Ports

Ports are endpoints that can be connected by edges. There is distinction between input and output ports (only the opposite types may have an edge between them).

A simple port is represented just by its name:

```
{ "name": "port_name" }
```

A port also may contain its own properties and descriptors, for example:

```
{
  "name": "port_name",
  "type": "audio",
  "descriptors": [
    {
      "name": "binding_type",
      "value": "UDMA_STREAM"
    }
  ]
}
```

## Edges

An edge is used to connect a source node output to a destination node input. Source and destination nodes must have the same immediate parent (direct edges are not possible across hierarchy boundaries).

```
{
  "source": {
    "name": "node_name",
    "output": "output_name"
  },
  "destination": {
    "name": "node_name",
    "input": "input_name"
  }
}
```

## Links

Links enable vertical connections across hierarchy. A link can be placed between a node port (input or output) and port of a child node with the same input/output direction.

In following example, a parent input port “input” is linked to input port “child\_port” of a child node “child\_node”. Child node object itself doesn’t keep information about which parent port it is linked to; this is deduced by the toolchain.

```
{
  "inputs": [
    {
      "name": "input",
      "type": "audio",
      "link": {
        "port": "child_port",
        "node": "child_node"
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "name": "child_node",
        "input": "child_port"
    }
}
}
}

```

## Implementations

An implementation object provides details about particular implementation of a component. On GAP9 platform, nodes may be mapped on several physical domains:

- SFU (Smart Filtering Unit)
- Fabric controller core (FC)
- Cluster multi-core block (CL)
- other miscellaneous hardware modules (e.g. Float-fix converter (FFC))

Composite components may also be mapped across multiple domains (although no such examples currently exist in component library).

Depending on the domain, implementation object may contain various information specific to the domain. For example, SFU implementation objects may contain “block” and “instance” descriptors, while software implementations (FC or CL) may contain list of source files, hints about required CPU or memory resources, etc.

AMT and RTG shall use this information to optimize graph resource allocation and performance.

Normally, available implementations shall be provided in component template (.comp) of library components, but also they can be given directly in main graph file.

A primitive node must have at least one implementation so that the toolchain would be able to map it to the GAP9 platform. Primitive node is a leaf node that doesn’t contain any subnodes and cannot be further expanded by loading templates.

Composite nodes don’t require implementation if all their subnodes can be resolved down to primitive nodes that each have at least one implementation.

## Main implementation properties

- **“name”**

Implementation name is a custom string used by “selected\_implementation” property of a node to select preferred implementation.

Data type: String

- **“location” descriptor**

A descriptor commonly present in implementations is “location”, aka domain. Simple components can be mapped to exactly one location, and they must provide value for “location” as one of keywords “sfu”, “fc”, “cluster” or “ffc”. Composite components may be mapped across multiple locations and they don’t require “location” descriptor at the top-level.

Data type: String

- **“parameters”**

Usually contains parameters obtained from common component parameters by some kind of conversion to adapt them to specific implementation. For example, floating-point filter coefficients need to be converted to fixed-point coefficients for SFU.

## Component specific properties

Apart from common properties, library components may have various specific properties. For comprehensive list of properties per component, see component's template file (.comp) and related documentation page.

### SFU implementations

SFU component implementations require some common descriptors related to resource allocation in SFU.

- **"block"**

Type of SFU block (e.g. GFU, MEM\_IN, MEM\_OUT, ...).

Data type: String

- **"instance"**

Physical instance ID of the block of type "block".

Data type: Integer

- **"context"**

Context within the "instance" reserved for this node. Concept of context is used by SFU to manage time sharing of physical instances by multiple logical nodes (e.g. filters). Contexts of the same instance reuse same computational resources while keeping separate states in memory. SFU blocks with no context support are treated as single-context blocks and "context" is optional (assumed to be 0).

Data type: Integer

- **"filter"**

For a GFU block, "filter" indicates which of several filters supported by GFU is to be used for this component.

Data type: String

- **"rt\_check"**

All clocked SFU blocks (with exception of MEM\_OUT) on GAP9 provide option to control triggering behavior. In SFU L2 mapping documentation this is denoted as RT\_EN configuration bit, accompanied by CLK\_SEL bit. Clocked SFU blocks in GAP9 are: MEM\_IN, STREAM\_IN, PDM\_IN, MEM\_OUT, STREAM\_OUT, PDM\_OUT, PIPE, RESAMPLER, POLYPHASE, FIFO and ASRC. "Clocked" here means that they produce an output only if a clock selected by CLK\_SEL is triggered. Other SFU blocks (e.g. GFU) are not explicitly clocked, but they push output to the target block as soon as they have calculated it.

In addition to clocked category, another category of blocks is defined: clock target blocks. Clock target blocks are subset of clocked blocks: MEM\_OUT, STREAM\_OUT, PDM\_OUT, PIPE, RESAMPLER, POLYPHASE, ASRC. If RT\_EN is enabled on them, they participate in RT violation detection of clock domain whose check-mask they are part of.

- **false**: On clock, the block will push next sample if it is available. If not available, the block will push it as soon as it becomes available. Target blocks ignore RT violations and do not raise RT violation condition.

- **true**: Block behavior depends on its role in a clock domain where RT violation occurs. *Target block*: On clock, if a target block did not receive an input sample to work with, it will raise RT violation condition on the clock domains in which they participate in check-mask. Also, since new sample is not available, it will use value of the previously received sample to proceed with its processing iteration, so that downstream blocks and clock domains remain undisturbed. *Clocked block*: On clock, if a violation condition is raised on this clock domain, it will drop its next output sample since downstream blocks are not able to accept it.

For more low-level details about GAP9 SFU RT behavior, see SFU\_RT.h source file.

Data type: Boolean

Currently, instance/context allocation is not automated and user must specify correct nonconflicting allocation for entire graph.

### FC and CL implementations

Some typical properties of software implementations are given below.

- **"c\_sources"**

List of .c source files for GAP target that implement the component on FC or CL.

Data type: Array of strings

- **"h\_sources"**

List of .h source files for GAP target that export relevant symbols to be used by runtime library and application.

Data type: Array of strings

### 6.1.3 Tools

#### Frontends

Frontends simplify the design of audio graphs by abstracting the use of the Audio Toolchain and by providing simple and intuitive ways to interact with graphs.

Main front end is Graph Designer (a GUI application).

In addition, graphs can also be created programmatically by either Python or C++ API provided by AF component library.

Default component library is located at:

```
<sdk>/tools/audio-framework/components
```

Each component in the library should provide a .hpp and .py file at its root directory. User application can then use provided component classes to instantiate nodes, set parameters and make connections between nodes.

Finally, all three ways to create graph shall produce a single file with graph description in JSON format of the AF Toolchain (.graph file) which is the only point of exchange between a frontend and the toolchain.

## Graph Designer

Graph Designer (GD) is a principal tool for designing audio graphs, filters and other components. It is maintained separately from AF and detailed documentation can be found dedicated Graph Designer repository.

Graph Designer maintains its own component library and internal JSON data format that is not compatible with format of the AF toolchain. It provides “Publish” command to export its graphs to AF compatible JSON format (by convention, a file with .graph extension).

## Python frontend API

Python API consist of core module `gap_audio_framework` and component classes available in the library.

`gap_audio_framework` module contains common methods to create graph, nodes, set properties and make connections between nodes. It provides base class `Node` from which all component classes should inherit from.

## Usage example

As an example, we take a look at one of provided examples located at `<sdk>/tools/audio-framework/examples/filters/fir/generators/graph.py`

The script imports relevant modules:

```
import gap_audio_framework as af  
from filters.fir.fir import Fir
```

Component module filters.fir.fir path is relative to component library root path <sdk>/tools/audio-framework/components.

Then, a top-level graph named “example” is created:

```
example = af.Node("example")
```

Then we create nodes and include them in the graph:

```
fir = Fir("fir", location=sfu.location, filter_length=22, coefficients_file='generators/  
→fir_1_filter_coefficients_lowpass83325131.dfl')  
example.add_node(fir)
```

This graph contains just a single FIR node where its ports are exported to the graph boundary. This is done by first creating graph boundary ports:

```
input = example.add_input('input', data_type='Q9.23')
output = example.add_output('output', data_type='Q9.23')
```

`data_type='Q9.23'` here indicates that this port expects audio data in Q9.23 fixed-point precision.

Then, the boundary ports are linked to ports of the FIR node:

```
example.bind(example, 'input', fir, 'processing')
example.bind(fir, 'push', example, 'output')
```

Finally, a base class method `gen` is called to export graph to a JSON file named `example.graph`.

## C++ frontend API

C++ frontend API consists of set of classes provided by component library that enable instantiation of nodes and creating connections between them. All component classes are derived from base class Graph which is the same class used by the tools AMT and RTG to manipulate graphs. This base class is defined in file

```
<sdk>/tools/audio-framework/toolchain/common/json_graph/src/graph.hpp
```

### Usage example

As an example, we take a look at one of applications located at `<sdk>/tools/audio-framework/examples/filters/fir/generators/graph.cpp`

The application includes core framework headers:

```
#include "json_parser.hpp"
#include "graph.hpp"
#include "af_utils/trace.hpp"
```

and header of the “fir” component:

```
#include "filters/fir/fir.hpp"
```

Header path `filters/fir/fir.hpp` is relative to component library root path `<sdk>/tools/audio-framework/components`.

Then, a top-level graph named “example” is created:

```
Graph top("example");
```

The we create nodes, set few properties and include them in the graph:

```
Fir *fir1 = new Fir("fir1", 22, "generators/fir_1_filter_coefficients_lowpass83325131.dfl
˓→", 0);
fir1->set_selected_implementation(std::string(argv[1]) + "_fir");
fir1->set_descriptor("instance", 0);
fir1->set_descriptor("context", 0);
top.add_node(*fir1);
```

This graph contains just a single FIR node where its ports are exported to the graph boundary. This is done by creating graph boundary ports as links to ports of the internal FIR node:

```
Input top_in1 = Input("input", fir1->name, "processing");
Output top_out1 = Output("output", fir1->name, "push");
```

and then adding them to the graph:

```
top.add_input(top_in1);
top.add_output(top_out1);
```

Finally, a framework library function `dump_to_file` is called to export graph to a JSON file named `example.graph`:

```
std::string output_filename("example.graph");
JsonGraph::Parser::dump_to_file(top, output_filename);
```

### Toolchain

The Audio Toolchain is a set of tools intended to simplify the design of audio graphs on GAP9.

It acts as a compiler to transform an high-level audio graph made by one of frontends into executable code and graph description understandable by runtime library. It abstracts the timing, hardware and software constraints to allow audio graph designers to focus on the graph design and not spend time on GAP9 platform technical details.

### Usage

Normally it is sufficient to use a wrapper script that calls individual tools of the toolchain:

```
gap-audio-toolchain --input <source_graph.graph> --output-dir <output_directory>
```

For list of options:

```
gap-audio-toolchain --help
```

Tools can be also called individually. To get full list of options:

```
audio_mapping_tool --help  
runtime_generator --help
```

## Tools

### Audio Mapping Tool

AMT takes high-level functional graph description provided by frontend applicatons and produces an elaborated graph that contains all details needed so that it can be deployed on GAP platform.

Currently it performs following functions:

- loads library components
- generates dynamic components
- inserts intermediate utility and interfacing nodes

Functions that are being developed:

- automatize mapping of the graph on GAP9 resources (mainly FC core, Cluser core and SFU module) to achieve optimal performance and resource usage
- calculate optimal buffer types and sizes between nodes
- calculate optimal scheduling

The Mapping Tool will optimize the mapping according to multiple constraints:

- real-time constraints
- latency
- power consumption
- hardware constraints (for example the number of hardware blocks)
- software constraints
- user-defined constraints

## Operation

AMT processes the graph in several steps.

Firstly, black-box nodes that reference library components (by “template” property) are expanded and replaced with full description that includes internal topology of the node, ports and parameters.

If a component has “generator” property, AMT will load referenced dynamic link library (.so) that contains various generators for that component.

If a component has “dynamic” property set, AMT will invoke “create” handler from component .so with user parameters to generate topology of the node. Example of this component in the library is mixer, which can be created with various number of input ports. Otherwise, if “dynamic”=false, component is considered static and fully described by its .comp file.

Then AMT overrides default parameter and descriptor values with ones from the input graph. This also includes overriding of properties of individual ports of the node.

If a component supports multiple implementations, AMT takes one (selected by the user) and drops all others.

With resolved node-level parameters, AMT then invokes “get\_implement\_params” handler from component .so with full set of functional parameters in order to convert functional parameters to parameters specific for selected implementation. For example, it may convert floating-point filter coefficients to fixed-point and bit-shift amounts relevant for SFU filters. Obtained converted parameters are then placed in “implementations” section of the node object.

When a node is fully resolved, AMT descends into subnodes and recursively repeats the process until all nodes of the graph are resolved.

Finally a .json file is produced containing expanded graph with all parameters converted to implementation-specific format. This file serves as input for RTG (Runtime Generator) tool.

## Usage

For usage details, try command:

```
audio_mapping_tool -h
```

## Audio Runtime Generator

This tool transforms a mapped graph into a representation that can be loaded and executed on a GAP9 processor.

The output of the tool is composed of:

- A list of C audio graph files that need to be compiled
- An audio runtime data representation of the graph

## Runtime library

### Audio Runtime

The audio runtime manages the audio graph running on GAP9. It schedules tasks according to their priorities to ensure that the graph is running properly on GAP9.

### 6.1.4 Audio libraries

Audio libraries contain premade components that can be used as building blocks to speed-up design process of audio graphs. Default library provided by GWT is located at `<sd>/tools/audio-framework/components`.

#### Component structure

A *component* is a module that can be instantiated in higher-level graph by referring to its template path (.comp file) and choosing values for minimal subset of parameters. Components complexity may range from a single filter to audio graphs like echo filtering or active noise cancellation.

Each component must contain following elements:

- JSON descriptor (.comp) placed in component root directory
- One of wrappers to allow its instantiation (either Python or C++, preferably both) by front end applications. Public interface (.hpp and .py) should be placed in component root directory, while C++ implementation is normally in gg subdirectory.

Optionally, a component may contain generator directory with C++ methods invoked by AMT during elaboration of the graph to execute custom functionality such as to generate custom parametrized topologies or convert parameters to specific implementation format.

Components that have software implementations may also have directories such as cluster or fc that contain related source code to be build for the target (GAP9) application.

Some examples of components with several hardware and software implementations:

- `<sdk>/components/filters/fir`
- `<sdk>/components/filters/biquad_cascade`

#### generator directory

This directory of a component contains C/C++ that is compiled as a dynamic link library for host machine and later loaded by AMT during graph elaboration if it encounters a node that instantiates this component. AMT will check existence of expected function symbols and (if they exist) call them at appropriate stages of node elaboration. Three methods are recognized by AMT graph parser and should be exported by the library as symbols “create”, “destroy” and “get\_implem\_params”.

- **create**

This method is called by AMT if component property `dynamic` is set to `true`. Single parameter for the method is a pointer to `std::vector<Parameter>` which contains list of parameters to be used for component creation. For example, `<sdk>/components/mixer` component is providing `create` method to create a mixer with custom number of input ports, where port number is given as a parameter “`nb_ports`” within parameters vector.

Parameters: `* std\:\:vector<Parameter> * parameters`

Return: Pointer to newly created Graph object.

Mandatory: No

- **destroy**

A clean-up method that should reverse allocations potentially made in `create`.

Parameters: `* Graph * p` : Pointer to Graph object to be deallocated.

- **get\_implement\_params**

Method to convert generic parameters to implementation-specific parameters. For example, it is used by SFU filter implementations to convert floating-point coefficients to fixed-point coefficients and internal arithmetic precision adjustments (bit shifts). Converted parameters are placed in implementation sub-structure of the node object, with parameter names that are expected by RTG.

Parameters: `* Graph * graph` : Node object `* char * implem_name` : Implementation name `* std\:\:\vector<Parameters> * parameters` : List of input parameters

Return: Error code

Mandatory: No

## JSON component descriptor (.comp)

This is the first element loaded by the AMT when a node that instantiates this component is encountered. Common properties expected to be found here are described below.

- **name**

Default name of a node that instantiates this component. Normally, the name should be customized by the user with node property of the same name "name".

Mandatory: Yes

Data type: String

- **component**

Name of the component. It should be unique across all libraries.

Mandatory: Yes

Data type: String

- **generator**

Relative path to dynamic link library (.so) that contains C++ methods for customized component elaboration by AMT. This library is obtained by compilation of source code in generator directory.

Mandatory: No

Data type: String

- **dynamic**

If after loads a template AMT finds `dynamic=true`, it will attempt to call `create` method from .so library of this component in order to generate the node programmatically. If the call succeeds, AMT will apply remaining user properties to just generated node and finally replace initial node with generated one. This allows scalable generation of complex nodes (subgraphs) by providing just few required parameters to `create` method.

Mandatory: No

Data type: Boolean

- **inputs**

List of input ports with their default properties.

Mandatory: No

Data type: List of input port objects

- **outputs**

List of output ports with their default properties.

Mandatory: No

Data type: List of output port objects

- **parameters**

List of parameters to apply over default component parameters. During node elaboration by AMT, this list is also given to .so methods `create` and `get_implem_params`.

Mandatory: No

Data type: List of parameter objects

- **implementations**

List of supported implementations of a component. Each implementaton object shall contain list of parameters and descriptors specific for that implementation. Software nodes shall also contain list of source files to be compiled for the target. Implementations of the same component may differ in internal topology and may map on different domains of GAP9. Implementations may refer to other components from the library. For more details about implementations section, see [Implementations](#).

Mandatory: Yes

Data type: List of implementation objects.

## Component reference

### Filters

#### Biquad cascade

Biquad cascade filter component. This component encapsulates both DF1 and DF2 biquad topologies.

For filter block diagrams, see SFU documentation.

#### Base parameters

- **nb\_stages**

Number of biquad stages in the cascade (N).

**Type:** Integer

**Range:** For “sfu\_df1” and “sfu\_df2” implementations, range for N is [1, 42].

- **MA\_coefficients**

List of MA coefficients (3 per stage) in following order: C0.B0, C0.B1, C0.B2, C1.B0, C1.B1, C1.B2, ..., C[N-1].B0, C[N-1].B1, C[N-1].B2 where Ci represents stages.

**Type:** Array of float of size 3 \* N.

- **AR\_coefficients**

List of AR coefficients (2 per stage) in following order: C0.A1, C0.A2, C1.A1, C1.A2, ..., C[N-1].A1, C[N-1].A2. A0 coefficients are assumed to be 1.0 and therefore omitted.

**Type:** Array of float of size 2 \* N.

- **stage\_gain**

List of stage gain multipliers (one per stage and one final multiplier) in following order: C<sub>0</sub>.G, C<sub>1</sub>.G, ..., C[N-1].G, G[final]

**Type:** Array of float of size N + 1.

- **input\_shift**

User-specified input shift.

**Type:** Integer

- **output\_shift**

User-specified output shift.

**Type:** Integer

### “sfu\_df1” implementation parameters

- **coefficients**

Fixed-point coefficients in following order:

C <sub>0</sub> .B <sub>0</sub> , C <sub>0</sub> .B <sub>1</sub> , C <sub>0</sub> .B <sub>2</sub> , C <sub>0</sub> .A <sub>1</sub> , C <sub>0</sub> .A <sub>2</sub> ,
C <sub>1</sub> .B <sub>0</sub> , C <sub>1</sub> .B <sub>1</sub> , C <sub>1</sub> .B <sub>2</sub> , C <sub>1</sub> .A <sub>1</sub> , C <sub>1</sub> .A <sub>2</sub> ,
...
C[N-1].B <sub>0</sub> , C[N-1].B <sub>1</sub> , C[N-1].B <sub>2</sub> , C[N-1].A <sub>1</sub> , C[N-1].A <sub>2</sub>

**Type:** Array of integers (signed, 32-bit)

- **shifts**

Bit shift values (according to SFU Biquad cascade DF1 block diagram):

C <sub>0</sub> .S <sub>0</sub> , C <sub>0</sub> .S <sub>1</sub> , C <sub>0</sub> .S <sub>2</sub> , C <sub>0</sub> .S <sub>3</sub> ,
C <sub>1</sub> .S <sub>0</sub> , C <sub>1</sub> .S <sub>1</sub> , C <sub>1</sub> .S <sub>2</sub> , C <sub>1</sub> .S <sub>3</sub> ,
...
C[N-1].S <sub>0</sub> , C[N-1].S <sub>1</sub> , C[N-1].S <sub>2</sub> , C[N-1].S <sub>3</sub>

### “sfu\_df2” implementation parameters

- **coefficients**

Fixed-point coefficients in following order:

C <sub>0</sub> .B <sub>0</sub> , C <sub>0</sub> .B <sub>1</sub> , C <sub>0</sub> .B <sub>2</sub> , C <sub>0</sub> .A <sub>1</sub> , C <sub>0</sub> .A <sub>2</sub> ,
C <sub>1</sub> .B <sub>0</sub> , C <sub>1</sub> .B <sub>1</sub> , C <sub>1</sub> .B <sub>2</sub> , C <sub>1</sub> .A <sub>1</sub> , C <sub>1</sub> .A <sub>2</sub> ,
...
C[N-1].B <sub>0</sub> , C[N-1].B <sub>1</sub> , C[N-1].B <sub>2</sub> , C[N-1].A <sub>1</sub> , C[N-1].A <sub>2</sub>

**Type:** Array of integers (signed, 32-bit)

- **shifts**

Bit shift values (according to SFU Biquad cascade DF1 block diagram):

```
C0.S0, C0.S1  
C1.S0, C1.S1  
...  
C[N-1].S0, C[N-1].S1
```

### “fc\_df1” implementation parameters

- **coefficients**

Floating-point coefficients made by combining stage coefficients with stage gains in following order:

```
C0.B0, C0.B1, C0.B2, C0.A1, C0.A2,  
C1.B0, C1.B1, C1.B2, C1.A1, C1.A2,  
...  
C[N-1].B0, C[N-1].B1, C[N-1].B2, C[N-1].A1, C[N-1].A2
```

**Type:** Array of floats

### “fc\_df2” implementation parameters

- **coefficients**

Floating-point coefficients made by combining stage coefficients with stage gains in following order:

```
C0.B0, C0.B1, C0.B2, C0.A1, C0.A2,  
C1.B0, C1.B1, C1.B2, C1.A1, C1.A2,  
...  
C[N-1].B0, C[N-1].B1, C[N-1].B2, C[N-1].A1, C[N-1].A2
```

**Type:** Array of floats

## FIR filter

FIR filter component. For filter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **filter\_length**

Number of delay elements in the filter (N). Number of taps is therefore N + 1.

**Type:** Integer

**Range:** [1, 253]

- **filter\_coefficients**

FIR filter coefficients in following order:  $B_0, B_1, B_2, \dots, B[N]$ .

**Type:** Array of doubles.

**Range:** [1, 253]

- **input\_shift**

User-defined input shift.

**Type:** Integer

**Range:** [-63, 32]

- **output\_shift**

User-defined output shift.

**Type:** Integer

**Range:** [-63, 32]

## Parameters of “sfu\_fir” implementation

- **coefficients**

Filter coefficients in fixed-point format. Arithmetic precision is calculated by component generator function `convert_double_to_sfu_fir` and depends on maximum absolute value of coefficients. Normally, So shift should be set to negative value of coefficients arithmetic precision.

**Type:** Array of integers (32-bit, signed) of size  $(N + 1)$ .

- **shifts**

Shifts  $S_i$  and  $S_o$ .

**Type:** Array of integers of size 2.

**Range:** [-63, 32]

## Parameters of “fc\_fir” and “cluster\_fir” implementations

These implementations have no additional parameters.

## IIR filter

IIR filter component. For filter block diagrams, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **filter\_length**

Number of delay elements in A or B branch ( $N$ ). Number of coefficients is therefore  $N + 1$ .

**Type:** Integer

**Range:** For “sfu\_iir\_df1” implementation, range is [1, 42].

- **MA\_coefficients**

Numerator coefficients  $MA(B)$  in following order:  $B_0, B_1, \dots, B[N]$ .

**Type:** Array of doubles of size  $N + 1$ .

- **AR\_coefficients**

Denominator coefficients  $AR$  ( $A$ ) in following order:  $A_1, A_2, \dots, A[N]$ .  $A_0$  is assumed to be 1 and thus is not configurable.

**Type:** Array of doubles of size N.

- **input\_shift**

User-defined input shift.

**Type:** Integer

- **output\_shift**

User-defined output shift.

**Type:** Integer

## Parameters of “sfu\_iir\_df1” implementation

- **coefficients**

Filter coefficients in fixed-point format arranged in single list as follows:

$B_0, B_1, A_1, B_2, A_2, \dots, B_N, A_N$

Arithmetic precision is calculated by component generator function `convert_to_sfu_iir` and it depends on maximum absolute value of coefficients.

**Type:** Array of integers (32-bit, signed) of size  $(N + 1)$ .

- **shifts**

Shifts  $S_i, S_0, S_1, S_2, S_o$  (see block diagram):

- $S_i$ : Input shift
- $S_0$ : Shift after multiplications by  $B[]$
- $S_1$ : Shift after multiplications by  $A[]$
- $S_2$ : Shift before entering delay elements of the A branch
- $S_o$ : Final shift

**Type:** Array of integers of length 5

**Range:** [-63, 32]

## Warped FIR

Warped FIR (WFIR) filter component. For filter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Number of stages/blocks (aka N) in the cascade of WFIR cells.

**Type:** Integer

**Range:** For “sfu\_wfir” implementation, range for N is [1, 169].

- **filter\_coefficients**

Coefficients  $C_0, C_1, \dots, C_N$  for final scalar product ( $N + 1$  coefficients total).

**Type:** Array of double floats

- **lambda**

Warping factor in floating-point representation.

**Type:** Double

- **q\_lambda**

Q (arithmetic precision) for lambda.

**Type:** Integer

**Range:** [0 : 31]

- **input\_shift**

User-specified input shift.

**Type:** Integer

- **output\_shift**

(Additional) user-specified output shift. It should not contain implicit output shift calculated by the generator used to return to the original Q format after multiplication with C[i]. Final output shift is thus a combination of the user “output\_shift” and generator’s shift.

**Type:** Integer

## Parameters of “sfu\_wfir” implementation

- **coefficients**

C[i] coefficients in Q1.31 fixed-point representation.

**Type:** List of integers

Layout: C[0], C[1], ..., C[N]

- **shifts**

Shifts calculated by the generator to regulate Q-precision along the filter while trying to maximize arithmetic precision by using as much accumulator bits as possible.

**Type:** List of integers

Layout:

- Si : input shift
- S0[i] : S0 shift of stage [i], i in range [0 : N-1]
- S1[i] : S1 shift of stage [i], i in range [0 : N-1]
- So: output shift

**Range:** [-63 : 31] is range for all shifts

- **lambda\_fix**

Lambda converted to fixed-point from base parameters “lambda” and “q\_lambda”.

**Type:** Integer

## Lattice ladder filter

Lattice ladder filter component. For filter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Number of lattice stages in the cascade (aka N).

**Type:** integer

**Range:** For “sfu\_lattice\_ladder” implementation, range for N is [1, 127].

- **filter\_k\_coefficients**

K[i] coefficients in floating point format. i in range [1:N]

**Type:** Array of double

- **filter\_v\_coefficients**

V[i] coefficients in floating point format. i in range [0:N]

**Type:** Array of double

### Parameters of “sfu\_lattice\_ladder” implementation

- **k\_coefficients**

K[i] coefficients in fixed-point representation with precision determined by Sk

**Layout:** K[1], K[2], ..., K[N]

**Type:** Array of integers (32-bit, signed)

- **v\_coefficients**

V[i] coefficients in fixed-point representation with precision determined by Sv

**Layout:** V[0], V[1], ..., V[N]

**Type:** Array of integers (32-bit, signed)

- **shifts**

Sk and Sv shifts are calculated by the generator to regulate Q-precision along the filter while trying to maximize arithmetic precision by using as much accumulator bits as possible.

**Layout:**

- Si : input shift

- Sk : shift after multiplication with K[i] (the same shift is used for all K[i])

- Sv : shift after multiplication with V[i] (the same shift is used for all V[i])

- So : output shift

**Type:** Array of integers

## Lattice FIR filter

Lattice FIR filter component. For filter block diagram, see SFU documentation.

### Base parameters

- **nb\_stages**

Description: Number of lattice FIR stages (aka N) in the cascade.

**Type:** Integer

**Range:** For “sfu\_lattice\_fir” implementation, range for N is [1, 255].

- **filter\_coefficients**

K[i] coefficients in floating point format. i in range [1:N]

**Type:** Array of floats

### Parameters of “sfu\_lattice\_fir” implementation

- **coefficients**

K[i] coefficients in Q1.31 fixed-point representation.

**Type:** Array of integers

Layout: K[0], K[1], ..., K[N-1]

- **shifts**

Shifts calculated by the generator to regulate Q-precision along the filter while trying to maximize arithmetic precision by using as much accumulator bits as possible.

Layout:

- Si : input shift

- Sk : shift after multiplication with K[i] (the same shift is used for all K[i])

- So : output shift

**Type:** Array of integers of size 3

## Lattice all-pole filter

Lattice all-pole filter component. For filter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Number of lattice stages in the cascade (aka N).

**Type:** integer

**Range:** For “sfu\_lattice\_ladder” implementation, range for N is [1, 255].

- **filter\_coefficients**

K[i] coefficients in floating point format. i in range [1:N]

**Type:** Array of double

## Parameters of “sfu\_lattice\_allpole” implementation

- **coefficients**

K[i] coefficients in Q1.31 fixed-point representation.

**Type:** List of integers (32-bit, signed)

**Layout:** K[0], K[1], ..., K[N-1]

- **shifts**

Shifts calculated by the generator to regulate Q-precision along the filter while trying to maximize arithmetic precision by using as much accumulator bits as possible.

**Type:** List of integers

**Layout:**

- Si : input shift

- Sk : shift after multiplication with K[i] (the same shift is used for all K[i])

- So : output shift

## Farrow

Farrow filter component. For filter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Number of FIR filter blocks (stages). Corresponds to M + 1 in block diagrams.

**Type:** Integer

**Range:** For “sfu\_farrow” implementation, range is [1, 8].

- **fir\_length**

Number of delay elements in one FIR block. Number of FIR coefficients in one block is therefore fir\_length + 1. Corresponds to N in block diagram.

**Type:** Integer

**Range:** For “sfu\_farrow” implementation, range is [1, 16].

- **filter\_coefficients**

Coefficients of all FIR blocks, arranged in ascending order along M and N dimensions according to the block diagram:

(m=0, n=0), (m=0, n=1), ..., (m=0, n=N),

(m=1, n=0), (m=1, n=1), ..., (m=1, n=N),

...

(m=M, n=0), (m=M, n=1), ..., (m=M, n=N)

**Type:** Double

**Range:** For “sfu\_farrow” implementation, range is not limited. Coefficients are normalized to (-1.0, +1.0) range and then converted to Q1.31 fixed point. Denormalization shift is placed in So.

- **nb\_mu\_sequences**

Number of micro-interval sequences.

**Type:** Integer

**Range:** For “sfu\_farrow” implementation, range is [1, 64].

- **nb\_mus\_per\_sequence**

Number of micro-intervals per sequence.

**Type:** Array of integers

**Range:** Array size must be the same as nb\_mu\_sequences. Array values must be in range [1, 16].

- **mu\_intervals**

Micro-intervals of all sequences arranged in 1-D list. Let’s denote number of micro-interval sequences as NMUS and number of micro-intervals per sequence as NMUPERS. Then micro-intervals should be arranged as follows:

(seq=0, mu=0), (seq=0, mu=1), ..., (seq=0, mu=NMUPERS(0)-1)

(seq=1, mu=0), (seq=1, mu=1), ..., (seq=1, mu=NMUPERS(0)-1)

...

(NMUS-1, mu=0), (NMUS-1, mu=1), ..., (NMUS-1, mu=NMUPERS(NMUS-1)-1)

**Type:** Array of double float

**Range:** [0.0, 1.0] (positive only)

- **input\_shift**

User input shift that is applied to input sample before entering the filter.

**Type:** Integer

**Range:** [-63, 31]

- **output\_shift**

User output shift that is applied to the output value before exiting the filter.

**Type:** Integer

**Range:** [-63, 31]

## Parameters of “sfu\_farrow” implementation

- **fir\_coefficients**

Fixed-point coefficients obtained from ‘filter\_coefficients’ parameter.

**Type:** Integer

**Range:** Signed 32-bit

- **mu\_intervals**

Upper 16 bits of fixed-point micro-intervals obtained from ‘mu\_intervals’ parameter. In SFU, it is padded with 15 zeros and then used as unsigned 31-bit value.

**Type:** Array of integers

**Range:** Unsigned 16-bit

- **shifts**

4 shift values of SFU implementation: Si, Sa, Su, So.

\* Si: Input shift obtained directly from ‘input\_shift’ parameter

\* Sa: A shift to return to normal precision after multiplication with micro-interval value. Composed of negative MU precision and negative MU normalization shifts.

\* Su: Applied to MU before multiplication. Not used in this component (set to 0).

\* Sout: Shift applied just before final output. It consists of negative coefficient precision shift, negative coefficient normalization shift and user's 'output\_shift' parameter.

**Type:** Array of integers

**Range:** [-63, 31]

## Horner structure

Horner polynomial evaluation component. For block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Order N of the polynomial to be evaluated.

**Type:** Integer

**Range:** For "sfu\_horner" implementation, range is [1, 255].

- **filter\_coefficients**

Polynomial coefficients arranged as C[0], C[1], ..., C[N].

**Type:** Array of double

**Range:** Array size must be (nb\_stages + 1).

- **input\_shift**

User input shift that is applied to input audio sample before entering the component.

**Type:** Integer

**Range:** [-63, -31]

- **output\_shift**

User output shift that is applied to the output value before exiting the component.

**Type:** Integer

**Range:** [-63, -31]

### Parameters of "sfu\_horner" implementation

For details about calculation of coefficients and shifts for sfu\_horner, please see comments in generator/src/horner.cpp.

- **coefficients**

Fixed-point coefficients obtained from 'filter\_coefficients' parameter.

**Type:** Array of integers

**Range:** Signed 64-bit

- **shifts**

Two shift values of SFU implementation: Si, So.

\* Si: Input shift, calculated as: 8 + input\_shift

\* So: Output shift, calculated as: 23 - coeffs\_precision - coeffs\_norm\_shift + output\_shift, where coeffs\_precision and coeffs\_norm\_shift are evaluated from 'filter\_coefficients' parameter, and 'output\_shift' is directly taken from the parameter.

## Resamplers

### Polyphase cascade resampler

Polyphase cascade component (rational ratio resampler). For block diagram, see SFU documentation.

In polyphase\_cascade.comp description, this component is using # notation to indicate that there may be multiple instances of related parameter (for nb\_stages > 1).

### Base parameters

Base parameters are common to all implementations.

- **nb\_stages**

Number of polyphase interpolator blocks in the cascade.

**Type:** Integer

**Range:** For “sfu\_polyphase” implementation, range is [1, 5].

- **filter\_length\_<c>**

Total number of coefficients of one polyphase block. The parameter has multiple instances (nb\_stages), with possible values for ‘c’ in range [1, 5].

**Type:** Integer

**Range:** For “sfu\_polyphase” implementation range is [1, 2048], with following constraints:

- Total number of coefficients of all blocks of the cascade must not exceed 2048
- Number of coefficients must be divisible by (upsampling\_ratio \* downsampling\_ratio)

- **filter\_coefficients\_<c>**

Coefficients of one polyphase block.

**Type:** List of double, or a filename.

If list of double is given, its size must be equal to ‘filter\_length\_<c>’.

If filename is given, its size in bytes must be equal to (‘filter\_length\_<c>’ \* 8).

**Layout:** Coefficients are a 3-D array along N, L, M axes, where N coordinate is the most significant and M coordinate least significant, that is, for coordinates (n, l, m) the layout is:

```
(0, 0, 0), (0, 0, 1), ..., (0, 0, M - 1),
(0, 1, 0), (0, 1, 1), ..., (0, 1, M - 1),
...
(0, L - 1, 0), (0, L - 1, 1), ..., (0, L - 1, M - 1),
(1, 0, 0), (1, 0, 1), ..., (1, 0, M - 1),
(1, 1, 0), (1, 1, 1), ..., (1, 1, M - 1),
...
(1, L - 1, 0), (1, L - 1, 1), ..., (1, L - 1, M - 1),
...
...
(N - 1, 0, 0), (N - 1, 0, 1), ..., (N - 1, 0, M - 1),
(N - 1, 1, 0), (N - 1, 1, 1), ..., (N - 1, 1, M - 1),
...
(N - 1, L - 1, 0), (N - 1, L - 1, 1), ..., (N - 1, L - 1, M - 1)
```

- **upsampling\_ratio\_<c>**

Resampler ratio numerator (L).

**Type:** Integer

**Range:** For “sfu\_polyphase” implementation the range is [1, 16].

- **upsampling\_ratio\_<c>**

Resampler ratio denominator (M).

**Type:** Integer

**Range:** For “sfu\_polyphase” implementation the range is [1, 16].

- **coefficient\_frac\_Q\_<c>**

Fixed-point Q precision that will be used to convert ‘filter\_coefficients’ to fixed-point values.

**Type:** Integer

**Range:** [1, 31]

- **output\_delay\_phase\_<c>**

Phase of each of the M branches in the output stage delay line.

**Type:** List of integers.

**Range:** List size must be equal to ‘downsampling\_ratio\_<c>’ (M).

List element value ranges from 0 to (M + L - 1).

## Parameters of “sfu\_polyphase” implementation

- **coefficients\_<c>**

Fixed-point coefficients in arithmetic precision given by coefficient\_frac\_Q base parameter.

**Type:** Array of integers (32-bit, signed)

**Layout:** Coefficients are a 3-D array along N, L, M axes, where N coordinate is the most significant and M coordinate least significant, that is, for coordinates (n, l, m) the layout is:

```
(0, 0, 0), (0, 0, 1), ..., (0, 0, M - 1),
(0, 1, 0), (0, 1, 1), ..., (0, 1, M - 1),
...
(0, L - 1, 0), (0, L - 1, 1), ..., (0, L - 1, M - 1),
(1, 0, 0), (1, 0, 1), ..., (1, 0, M - 1),
(1, 1, 0), (1, 1, 1), ..., (1, 1, M - 1),
...
(1, L - 1, 0), (1, L - 1, 1), ..., (1, L - 1, M - 1),
...
...
(N - 1, 0, 0), (N - 1, 0, 1), ..., (N - 1, 0, M - 1),
(N - 1, 1, 0), (N - 1, 1, 1), ..., (N - 1, 1, M - 1),
...
(N - 1, L - 1, 0), (N - 1, L - 1, 1), ..., (N - 1, L - 1, M - 1)
```

## Interfaces

### PDM demodulator

PDM demodulator component. For implementation block diagram, see SFU documentation.

#### Base parameters

Base parameters are common to all implementations.

- **cic\_order\_n**

Length of cascaded integrator (N).

**Type:** Integer

**Range:** [1, 8]

- **cic\_depth\_m**

Depth of the comb filter (M).

**Type:** Integer

**Range:** [1, 2]

- **cic\_ratio\_r**

Decimation ratio (R).

**Type:** Integer

**Range:** [1, 256]

- **cic\_shift**

Arithmetic shift value applied to the 32-bit output (right-shift only for sfu\_pdm\_in implementation)

**Type:** Integer

**Range:** [-63, 0] for sfu\_pdm\_in implementation

#### Parameters of “sfu\_pdm\_in” implementation

No specific parameters for sfu\_pdm\_in implementation.

### PDM modulator component

PDM modulator component. For implementation block diagram, see SFU documentation.

#### Base parameters

Base parameters are common to all implementations.

- **interpolator\_ratio**

Interpolator ratio.

**Type:** Integer

**Range:** [4, 128] For sfu\_pdm\_out implementation, only discrete values are valid: {4, 8, 16, 32, 64, 128}

- **interpolator\_type**

Interpolator type:

- “linear” - Linear interpolator
- “iir” - Interpolator made of zero-insertion upsampler followed by low-pass filter implemented as biquad DF1 cascade.

**Type:** String enum

- **interpolator\_nb\_stages**

Number of biquads in IIR interpolator.

**Type:** Integer

For sfu\_pdm\_out implementation the only valid value is 5.

- **interpolator\_coeffs\_ma**

MA (B) coefficients of IIR biquads.

**Type:** Array of double of size (interpolator\_nb\_stages \* 3)

- **interpolator\_coeffs\_ar**

AR (A) coefficients of IIR biquads.

**Type:** Array of double of size (interpolator\_nb\_stages \* 2)

- **modulator\_forward\_coeffs**

B coefficients of CRFB modulator.

**Type:** Array of double of size 6.

- **modulator\_feedback\_coeffs**

A coefficients of CRFB modulator.

**Type:** Array of double of size 5.

- **modulator\_loopback\_gains**

Gain coefficients of CRFB modulator.

**Type:** Array of double of size 2.

- **modulator\_coeffs\_q**

Q fixed-point precision of modulator\_forward\_coeffs and modulator\_feedback\_coeffs. In sfu\_pdm\_out implementation, gain Q precision is fixed to 31.

**Type:** Integer

**Range:** [0, 31]

- **modulator\_quantization\_values**

Two quantization values (positive and negative).

**Type:** Array of integers of size 2.

## Parameters of “sfu\_pdm\_out” implementation

- **modulator\_shifts**

List of shift values.

- PRSH: Shift after multiplication with coefficients B or A. Normally it's set to -modulator\_coeffs\_q.
- FSBH: Shift after multiplication with gain. Normally it's set such that FSBH + PRSH = -31 (so that G0 and G1 may be represented in Q31).

**Type:** Array of integers of size 4 in order PRSH[0], PRSH[1], LBSH[0], LBSH[2]

**Range:** Element range [-63, 0]

- **modulator\_forward\_coeffs\_fix**

B coefficients of CRFB modulator in fixed-point with arithmetic precision defined by `modulator_coeffs_q`.

**Type:** Array of integers of size 6.

- **modulator\_feedback\_coeffs\_fix**

A coefficients of CRFB modulator in fixed-point with arithmetic precision defined by `modulator_coeffs_q`.

**Type:** Array of integers of size 5.

- **modulator\_loopback\_gains\_fix**

Gain coefficients of CRFB modulator in fixed-point with arithmetic precision fixed to Q31.

**Type:** Array of integers of size 2.

## Other

### Limiter component

Limiter component. For limiter block diagram, see SFU documentation.

### Base parameters

Base parameters are common to all implementations.

- **external\_gain\_enable**

Enable/disable external gain port.

- **false**: Gain is calculated automatically by estimating incoming signal level (envelope) and using knee curve to calculate gain to be applied to input samples
- **true**: A second input port is added to the limiter and the gain is taken directly from this port. Envelope estimation and knee curve are not used.

**Type:** Boolean

- **external\_gain\_smoothing**

Applicable only if `external_gain_enable = true`.

- **false**: External gain values are applied directly.
- **true**: External gain is first smoothed through an integrator with coefficients given by `gain_smooth_coeffs`.

**Type:** Boolean

- **decimation**

Ratio at which the gain is recomputed (computation rate is sample rate divided by ‘decimation’ parameter). Valid only in automatic mode (`external_gain_enable = false`).

0 - No decimation

1 - Evaluated every 2nd sample

2 - Evaluated every 3rd sample

...

**Type:** Integer

**Range:** [0, 255]

- **gain\_smooth\_coeffs**

Gain smoothing coefficients.

**Type:** Array of 2 elements of type double

- **env\_smooth\_up\_coeffs**

Envelope smoothing coefficients (when new local maximum is greater than current estimated level)

**Type:** Array of 2 elements of type double

- **env\_smooth\_down\_coeffs**

Envelope smoothing coefficients (when new local maximum is lower than current estimated level)

**Type:** Array of 2 elements of type double

- **knee\_lower\_threshold and knee\_upper\_threshold**

Thresholds along x-axis of knee curve for gain calculation. Lower threshold must be less than upper threshold.

- If input is lower than lower threshold, the gain is forced to 1.0 (transmit zone).
- If input is between lower and higher thresholds, the gain is calculated as polynomial with coefficients `knee_polynomial_coeffs` of envelope level as its input. This is limiter knee zone.
- If input is greater than upper threshold, the gain gain forced to  $(1.0 / \text{level})$ . This is saturation zone.

**Type:** Double

- **knee\_polynomial\_coeffs**

Coefficients of polynomial used to calculate gain if the envelope level is in knee zone (between lower and upper thresholds).

**Type:** Array of 6 doubles

### **Parameters of “sfu\_limiter” implementation**

- **volume\_select**

Selection of one of 32 volume control registers available in SFU to be assigned to the limiter.

**Type:** Integer

**Range:** [0, 31]

### **Stub components**

Stub components are used for modeling and testing of TC features.

### **LMS**

LMS component. Least-Mean-Squares algorithm is processed block by block. Implementation is only available on the cluster.

## Parameters

- **filter\_order**

Number of delay elements (N) in the adaptive filter (in the audio space) . Number of taps is therefore N + 1.

**Type:** Integer

**Range:** [1, 254]

- **downsampling\_ratio**

Decimation factor between High Sampling Rate domain and Audio domain where LMS is performed.

**Type:** Integer

**Range:** [1, 16]

- **compens\_filter\_order**

Number of delay elements (N) in the IIR compensation filter. Number of taps is therefore N + 1 for the numerator, and N for the denominator.

**Type:** Integer

**Range:** [1, 14]

- **compens\_filter\_coefficients**

IIR filter coefficients, stored in following order: B0, B1, ..., BN, A1, ..., AN.

**Type:** Array of floats of size 2 \* N + 1

- **lagrange\_order**

Order of polynom (N) for Lagrange interpolation of the Farrow structure.

**Type:** Integer

**Range:** [1, 7]

- **lagrange\_coefficients**

Coefficients of the transfer matrix of the Farrow structure, stored in the following order: C0.0, C0.1, ..., C0.N, C1.0, C1.1, ..., CN.N.

**Type:** Array of floats of size ( N + 1 ) \* ( N + 1 )

- **num\_of\_cores**

Number of cores for the parallelisation on the cluster.

**Type:** Integer

**Range:** [1, 8]

## ASRC

ASRC component (Asynchronous Sample Rate Converter). For filter block diagram, see SFU documentation.

## Parameters

- **nb\_channels**

Number of channels of the filter. In the current implementation there are 3 ASRC units and each ASRC unit can accept up to 4 inputs and 4 outputs.

**Type:** Integer

**Range:** [1, 4]

- **nb\_coefficients**

Number of filter coefficients (N) in following order: DC, C0, C1, ..., C[N-1].

**Type:** Array of doubles.

**Range:** [1, 2049]

- **filter\_coefficients**

Filter coefficients in fixed-point format. Arithmetic precision is calculated by component generator function `convert_double_to_sfu_asrc` and it depends on maximum absolute value of coefficients.

**Type:** Array of integers (32-bit, signed) of size (N).

## Descriptors

- **clock\_in**

Specifies input clock CLK\_SAI0, CLK\_SAI1, CLK\_SAI2, CLK\_PDM0, CLK\_PDM1, CLK\_PDM2, CLK\_PDM3, CLK\_PDM4, CLK\_PDM5, CLK\_PDM6, CLK\_PDM7, CLK\_PDM8, CLK\_PDM9, CLK\_PDM10, CLK\_PDM11, CLK\_PWM0, CLK\_PWM1, CLK\_PWM2, CLK\_PWM3, CLK\_AUD0, CLK\_AUD1, CLK\_AUD2, CLK\_AUD3, CLK\_VIRT, CLK\_UNSPEC.

**Type:** String enum.

- **clock\_out**

Specifies output clock CLK\_SAI0, CLK\_SAI1, CLK\_SAI2, CLK\_PDM0, CLK\_PDM1, CLK\_PDM2, CLK\_PDM3, CLK\_PDM4, CLK\_PDM5, CLK\_PDM6, CLK\_PDM7, CLK\_PDM8, CLK\_PDM9, CLK\_PDM10, CLK\_PDM11, CLK\_PWM0, CLK\_PWM1, CLK\_PWM2, CLK\_PWM3, CLK\_AUD0, CLK\_AUD1, CLK\_AUD2, CLK\_AUD3, CLK\_VIRT, CLK\_UNSPEC.

**Type:** String enum.

- **manual\_ratio**

Enables/disables manual ratio set.

- **false**: ASRC is in automatic mode. That assumes that the physical clocks on input and output streams are available.

- **true**: ASRC is in manual mode. SO the user should set the ratio during runtime. The ASRC block will use this ratio to perform conversion.

**Type:** Boolean.

- **locking\_window**

Sets how many ratio calculation cycles the error should be below threshold before releasing the lock.

**Type:** Integer.

**Range:** [0, 7]

- **wait\_lock\_on\_input**

Blocks input transfers when no lock.

- **false**: Allows input samples.

- true: Halts input samples.  
**Type:** Boolean.
- **wait\_lock\_on\_output**  
Doesn't send output samples when no lock.
  - false: Allows output samples.
  - true: Halts output samples.**Type:** Boolean.
- **drop\_inputs\_when\_no\_lock**  
Allows input transfers when no lock, but replace input values with zeros.
  - Type:** Boolean.
- **reset\_on\_unload**  
Does HW reset of the ASRC on graph unload.
  - Type:** Boolean.
- **disable\_coefficient\_loading**  
If set ASRC coefficients are not loaded on next graph load.
  - Type:** Boolean.

## 6.1.5 Runtime descriptor format

### Main graph descriptor

A graph is described by a wrapper structure which is generated by the toolchain and must be compiled with the application. It provides several elements:

- Reference to the binary graph descriptor (either a file or a pointer)
- List of descriptors of software kernels used by the graph (**descriptors**)
- Reference to runtime information to manipulate SFU graphs (**sfu**)

```
typedef struct pi_audio_graph_desc_s
{
    /* One of 'filepath' or 'array' must be different from NULL */

    /*
     * If not NULL, file path in flash of the binary graph descriptor.
     * Otherwise, unused.
     */
    const char *filepath;

    /*
     * If not NULL, pointer to graph descriptor embedded in application as a C array.
     * Otherwise, unused.
     */
    uint8_t *array;

    uint32_t desc_size; /* Size of the binary graph descriptor */
    void **descriptors; /* Pointer to list of descriptors of software kernels */
    pi_sfu_rt_desc_t *sfu; /* Pointer to top-level descriptor of SFU graphs */
}
```

(continues on next page)

(continued from previous page)

```
} pi_audio_graph_desc_t;
```

Once loaded, the file containing the graph descriptor first contains the graph descriptor header.

Binary descriptor contains hierarchical description of the complete graph. Binary descriptor always begins with fixed graph descriptor structure (`pi_audio_graph_t`), followed by variable part that depends on number of ports and nodes in the graph. Nodes and ports are referenced by relative offsets that are converted to pointers during loading and relocating of the graph by `pi_audio_graph_open()`. For software nodes, all their parameters are give in their node section within binary graph descriptor. For SFU nodes, binary graph descriptor contains just minimum number of fields and a reference to corresponding node in SFU subgraph.

Binary descriptor (both graph and node descriptor) may contain fields that are left undefined by the toolchain and populated at runtime during graph loading. In other words, graph is elaborated “in place” in order to save some memory. This is also indicated by union in graph, node and port structures where `tc` represents toolchain view, and `rt` represents runtime view of the same memory area.

### Code descriptor

Each kernel needed by the graph must provide a descriptor containing its entry functions, so that the runtime can call them when appropriate. Code descriptor is an array of pointers of type `void *`. When preparing binary graph descriptor, the toolchain puts together this array by collecting all unique kernels used by the software nodes of the graph. Also, toolchain puts index of a kernel in this list in software node descriptor so that runtime can find kernels that correspond to a particular node.

Each descriptor in the descriptors array is of this type:

```
typedef struct
{
    /* Called one time on graph opening. It should contain startup initialization of the
     * kernel */
    int (*open)(pi_audio_node_header_t *node, pi_audio_node_conf_t *conf);

    /* Called on each start/resume of the graph. */
    int (*start)(pi_audio_node_header_t *node);

    /* Called by runtime scheduler to check whether the node is ready for execution
     * cycle. */
    int (*check)(pi_audio_node_header_t *node, pi_audio_node_check_exec_info_t *info);

    /*
     * Called by runtime scheduler when sufficient amount of buffers is enqueued on
     * input and
     * output ports of the node so that it can safely process one block of data.
     */
    void (*exec)(pi_audio_node_header_t *node);

    /*
     * Called by the runtime when upstream node or application have requested control
     * action
     * on the node, such as reconfiguration (e.g. filter coefficients update).
     */
}
```

(continues on next page)

(continued from previous page)

```
    int (*control)(pi_audio_node_header_t *node, int port_id, void *params);
} pi_audio_node_desc_t;
```

## Graph descriptor header

The graph descriptor in flash begins with a header of fixed format that describes top-level graph (root).

All pointers are actually offsets in the flash descriptors and the runtime is reallocating them when graph is loaded from flash to L2, in order to have pointers instead of offsets.

The offsets are computed by the toolchain and are relative to start of the graph descriptor. This is true for all the pointers of the graph descriptor.

```
typedef struct pi_audio_graph_s
{
    uint32_t nb_nodes;           // Number of nodes at top level
    uint32_t nb_inputs;          // Number of graph inputs
    uint32_t nb_outputs;         // Number of graph outputs
    pi_audio_node_header_t **nodes; // Pointer to the array of nodes (relocated by runtime)
    pi_audio_graph_input_t *inputs; // Pointer to the array of graph inputs (relocated by runtime)
    pi_audio_graph_output_t *outputs; // Pointer to the array of graph outputs (relocated by runtime)
    void **descriptors; // Used by runtime to keep reference to kernel descriptor list
    struct pi_device * cluster_dev; // Used by runtime to keep reference to cluster device
} pi_audio_graph_t;
```

## Node descriptor

```
typedef struct pi_audio_node_header_s
{
    union
    {
        struct
        {
            // Verbatim section (copy as is from .tc)
            uint32_t nb_nodes;           // Number of subnodes
            uint32_t nb_inputs;          // Number of input ports of the node
            uint32_t nb_outputs;         // Number of output ports of the node
            uint32_t nb_params;          // Number of (or total size of) parameters of the node
            uint32_t scheduled;          // Used by runtime scheduler
            uint32_t location;           // Node location (SFU, FC, CL, ...)
            uint32_t type;               // Node type (within location)
            uint32_t sfu_graph_idx;       // N/A (use tc.sfu_graph_idx)

            // Section relocated at runtime
        } node;
    };
}
```

(continues on next page)

(continued from previous page)

```

    pi_audio_node_inout_header_t *first_inout; // Used by runtime scheduler
    pi_evt_t *task;
    pi_audio_graph_t *graph;      // SW nodes only: pointer to RT object of the
    ↵graph
                                // that contains this node
    void *descriptor;           // SFU nodes: N/A (use tc.descriptor)
                                // SW nodes only: tc.descriptor relocated to a
    ↵pointer
    pi_audio_node_input_t *inputs;     // Array of pointers to input ports
    pi_audio_node_output_t *outputs;   // Array of pointers to output ports
    void *state;      // Private state, to be maintained between exec() calls
    void *temp;        // Private scratch data, can be discarded after exec()
    void *coeffs;     // For FC nodes: may be retrieved directly from descriptor.
                        // For CL nodes: L1 heap copy of coefficients.
    void *params;      // First level node parameters as given by TC
                        // Internal structure of the parameters is private to the
    ↵node
                                // implementation.
    pi_audio_node_header_t **nodes; // Array of pointers to subnodes
} rt; // Runtime view (after relocation)
struct {
    // Verbatim section
    uint32_t nb_nodes;
    uint32_t nb_inputs;
    uint32_t nb_outputs;
    uint32_t params_size;
    uint32_t scheduled;
    uint32_t location;
    uint32_t type;
    uint32_t sfu_graph_idx; // SFU nodes only: graph index in L2_RT_Descr_Table

    // Section relocated at runtime
    uint32_t first_inout;
    uint32_t task;
    uint32_t graph;          // N/A (use rt.graph allocated at runtime)
    uint32_t descriptor;    // SFU nodes: node index in SFU_RunTimeDescr_T
                            // Software nodes: sw descriptor index in table of
    ↵pi_audio_node_desc_t
    uint32_t inputs;
    uint32_t outputs;
    uint32_t state;
    uint32_t temp;
    uint32_t coeffs;
    uint32_t params;
    uint32_t nodes;
} tc; // Toolchain view
struct {
} raw;
};

};

} pi_audio_node_header_t;

```

## Node input/output port descriptors

```
typedef struct
{
    union
    {
        struct {
            pi_audio_node_inout_header_t header;           // Port header
            pi_audio_node_output_t *output;                // Pointer to connected output port
            or linked input port
            uint32_t flags;                                // Port flags (see PI_AUDIO_PORT_
            FLAG macros)
            char binding_type;                            // Port binding type (see PI_AUDIO_
            BINDING_TYPE macros)
            } rt; // Runtime view after relocation
        struct {
            uint32_t output_offset;                      // Offset to connected output port or linked_
            input port
            uint32_t flags;                            // Port flags (see PI_AUDIO_PORT_FLAG macros)
            int sample_data_type;                     // Port payload data type (see PI_AUDIO_DATA_
            TYPE macros)
            int sample_q_precision;                  // Arithmetic precision (for fixed-point data_
            types)
            int binding_type;                        // Binding type (see PI_AUDIO_BINDING_TYPE_
            macros)
            int binding_buffer_size;                 // Buffer size for buffered bindings
            int binding_max_nb_buffers;             // Number of buffers (for buffered bindings)
            } tc; // Toolchain view
        struct {
            uint8_t padding[PI_AUDIO_PORT_RSVD_SIZE];
        } raw;
    };
} pi_audio_node_input_t;
```

```
typedef struct
{
    union
    {
        struct {
            pi_audio_node_inout_header_t header;           // Port header
            pi_audio_node_input_t *input;                 // Pointer to connected input port
            or linked output port
            uint32_t flags;                                // Port flags (see PI_AUDIO_PORT_
            FLAG macros)
            char binding_type;                            // Port binding type (see PI_AUDIO_
            BINDING_TYPE macros)
            } rt;
        struct {
            uint32_t input_offset;                      // Offset to connected input port or linked_
            output port
            uint32_t flags;                            // Port flags (see PI_AUDIO_PORT_FLAG macros)
            int sample_data_type;                     // Port payload data type (see PI_AUDIO_DATA_
            TYPE macros)
```

(continues on next page)

(continued from previous page)

```

    int sample_q_precision;           // Arithmetic precision (for fixed-point data)
→ types)                         // Binding type (see PI_AUDIO_BINDING_TYPE)
→ macros)                         int binding_buffer_size;   // Buffer size for buffered bindings
                                    int binding_max_nb_buffers; // Number of buffers (for buffered bindings)
} tc;
struct {
    uint8_t padding[PI_AUDIO_PORT_RSVD_SIZE];
} raw;
};

} pi_audio_node_output_t;

```

**Graph input/output port descriptors**

```

typedef struct
{
    union
    {
        struct {
            pi_audio_node_input_t *node_input; // Pointer to a child input port linked
→ to this port
                                         // (graph input must always be a link)

→ input
            char * node_input_name;          // Pointer to port name string
            char binding_type;             // Port binding type (see PI_AUDIO_

→ BINDING_TYPE macros)
            } rt;
        struct {
            uint32_t node_input_offset;     // Offset to linked internal input port
            uint32_t node_input_name_offset; // Port name string offset
            int sample_data_type;          // Port payload data type (see PI_AUDIO_

→ DATA_TYPE macros)
            int sample_q_precision;        // Arithmetic precision (for fixed-point_
                                         // data)

→ data types
            int binding_type;             // Binding type (see PI_AUDIO_BINDING_)

→ TYPE macros)
            int binding_buffer_size;       // Buffer size for buffered bindings
            int binding_max_nb_buffers;   // Number of buffers (for buffered_
                                         // bindings)

→ bindings)
            } tc;
        struct {
            uint8_t padding[PI_AUDIO_PORT_RSVD_SIZE];
        } raw;
    };
} pi_audio_graph_input_t;

```

```

typedef struct
{
    union

```

(continues on next page)

(continued from previous page)

```

{
    struct {
        pi_audio_node_output_t *node_output;           // Pointer to a child output port
        ↵linked to this port
        char * node_output_name;                      // Pointer to port name string
        char binding_type;                            // Port binding type (see PI_AUDIO_
        ↵BINDING_TYPE macros)
        } rt;
        struct {
            uint32_t node_output_offset;               // Offset to linked internal output
        ↵port
            uint32_t node_output_name_offset;          // Port name string offset
            int sample_data_type;                     // Port payload data type (see PI_
            ↵AUDIO_DATA_TYPE macros)
            int sample_q_precision;                  // Arithmetic precision (for fixed-
            ↵point data types)
            int binding_type;                        // Binding type (see PI_AUDIO_
            ↵BINDING_TYPE macros)
        } tc;
        struct {
            uint8_t padding[PI_AUDIO_PORT_RSVD_SIZE];
        } raw;
    };
} pi_audio_graph_output_t;

```

## 6.1.6 Runtime API

### Contents

#### Graph management

group **GRAPH**

## TypeDefs

```
typedef struct pi_sfu_conf pi_sfu_conf_t
typedef struct pi_audio_conf pi_audio_conf_t
```

## Functions

```
static inline void pi_sfu_conf_init(struct pi_sfu_conf *conf)
    Initialize an sfu configuration with default values.
```

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the sfu is opened.

### Parameters

- **conf** – [in] A pointer to the sfu configuration.

```
void pi_sfu_enable()
    Enable SFU device.
```

Enable SFU domain FLL, power on SFU, ungate SFU clock and deassert SFU reset.

```
void pi_sfu_disable()
    Disable SFU device.
```

Assert SFU reset, gate its clock and power it down. Keep FLL running.

```
void pi_sfu_switch(int power_en, int clock_en)
    Switch on/off SFU power or clock.
```

### Parameters

- **power\_en** – 0: Switch SFU voltage domain off. 1: Switch SFU voltage domain on.
- **clock\_en** – 0: Disable (gate) SFU domain clock. Keep FLL running. 1: Enable (ungate) SFU domain clock.

```
void pi_sfu_reset()
    \bref Do hardware reset of the SFU.
```

Do SFU reset without gating clock or powering down the SFU.

```
int pi_sfu_open(pi_sfu_conf_t *conf)
    Open the sfu.
```

This will allocate and initialize all the resources needed to use the sfu. Once this function is successfully called, other functions for managing graphs can be called.

---

**Note:** This function can not be called from an interrupt handler.

---

### Parameters

- **conf** – [in] A pointer to the sfu configuration. Can be NULL to take default values.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

---

```
void pi_sfu_close()
cloe the sfu.
```

This will free and deinitialize all the resources. Once this function is called, other functions for managing graphs can not be called anymore, and the sfu must be opened before it can be used again.

---

**Note:** This function can not be called from an interrupt handler.

```
pi_sfu_graph_t *pi_sfu_graph_open(pi_sfu_graph_desc_t *graph_desc)
Open an SFU graph (in new resource group).
```

This will open an SFU graph and allocate additional resources needed by the graph (e.g. uDMA channels). The function does not interact with SFU, rather it just prepares runtime structures. To actually start the graph execution, *pi\_sfu\_graph\_load()* must be called afterwards.

The call will always create new resource group for the given graph. See *pi\_sfu\_graph\_open\_in\_group* for description of resource group management. If application needs to control group management, *pi\_sfu\_graph\_open\_in\_group()* should be used instead of *pi\_sfu\_graph\_open()*.

---

**Note:** This function cannot be called from an interrupt handler.

### Parameters

- **graph\_desc – [in]** : A pointer to the SFU config descriptor generated by the SfuGen tool.\*

**Returns** A pointer to the sfu graph instance or NULL if it failed.

```
pi_sfu_graph_t *pi_sfu_graph_open_in_group(pi_sfu_graph_desc_t *graph_desc, int32_t rgroup_id_req,
                                         int32_t *rgroup_id)
```

Open an SFU graph in group.

This will open an SFU graph and allocate additional resources needed by the graph (e.g. uDMA channels). The function does not interact with SFU, rather it just prepares runtime structures. To actually start the graph execution, *pi\_sfu\_graph\_load()* must be called afterwards.

A non-SFU resource sharing is supported by rgroup\* arguments (uDMA linear channel is an example of a resource). Depending on life-cycle of the graphs in application, some groups of graphs may need to run concurrently, while some groups of graphs may never have to run concurrently. As pi\_sfu manages non-sfu resources internally, the user must give a hint about which graphs are never supposed to run concurrently. This provided by *rgroup\_id\_req* and *rgroup\_id* arguments.

Two graphs that belong to the same resource group will use the same pool of resources, meaning that they cannot run concurrently. Two graphs that belong to different resource groups will not overlap in resource usage, meaning that they can run concurrently.

Groups are internally managed by the driver. Group IDs can be 1, 2, and so on. Group ID 0 is not used because value 0 is used as special value of *rgroup\_id\_req* to ask for new group. If *rgroup\_id\_req* is 0, a new group will be created at first available ID. Otherwise, the graph will be added to existing group.

---

**Note:** This function cannot be called from an interrupt handler.

### Parameters

- **graph\_desc – [in]** : A pointer to the SFU config descriptor generated by the SfuGen tool.

- **rgroup\_id\_req** – [in] Requested group ID 0: Create new group with first available ID >0: Add graph to existing group with given ID. If the group doesn't exist the call will fail
- **rgroup\_id** – [in] Obtained group ID 0: Failure >0 Obtained group ID

**Returns** A pointer to the sfu graph instance or NULL if it failed.

```
int pi_sfu_graph_close(pi_sfu_graph_t *graph)
    Close an sfu graph.
```

This will free all resources allocated for a graph. After a call to this function, the graph can not be used anymore.

---

**Note:** This function can not be called from an interrupt handler.

---

#### Parameters

- **graph** – [in] A pointer to the graph instance.

**Returns** Error code

```
pi_sfu_graph_t *pi_sfu_graph_clone(pi_sfu_graph_t *original_graph, int self, pi_sfu_clone_patch_desc_t
    *patch_desc)
```

Clone graph by using static cloning. Static cloning relies on cloning info generated by SfuGen, where cloned graphs are selected in .src file by SFU\_CloneGraph() directive.

#### Parameters

- **original\_graph** – [in] : Pointer to the original graph to clone from
- **self** – [in] : Whether to clone original graph onto itself. This can be used if application needs to load SFU with original graph after some cloned graphs have been loaded.
- **patch\_desc** – [in] : Cloning patch descriptor generated by SfuGen

```
int pi_sfu_graph_apply_clone_patch(pi_sfu_graph_t *graph)
```

Apply patch information of the cloned graph to the L2 config of the original graph. To be called before loading next cloned graph. Can be used only with graph cloned with [pi\\_sfu\\_graph\\_clone\(\)](#).

#### Parameters

- **graph** – [in] : Pointer to the cloned graph descriptor

**Returns** Error code : 0 - Success

```
pi_sfu_graph_t *pi_sfu_graph_dynamic_clone(pi_sfu_graph_t *original_graph, int self, int full)
```

Clone graph by using dynamic cloning. Dynamic cloning allocates required memory for a cloned graph configuraiton at runtime. Cloned graph references the level 1 L2 config of the original graph, but allocates its own L2 copies of GFU coefficients and states. It doesn't require additional cloning info from SfuGen, except that it should be ensured that .Visible property is set for the GFU nodes that require cloning.

#### Parameters

- **original\_graph** – [in] : Pointer to the original graph to clone from
- **self** – [in] : Whether to clone original graph onto itself. This can be used if application needs to load (again) the original graph in SFU after some cloned graphs have been loaded.
- **full** – [in] : Whether to make entire new copy of original L2 graph. If 1, dynamic patching between loading graphs is not needed (at the cost of more memory spent). TODO: Not yet supported.

---

```
int pi_sfu_graph_apply_dynamic_clone_patch(pi_sfu_graph_t *graph)
```

Apply patch information of the cloned graph to the L2 config of the original graph. To be called before loading next cloned graph. Can be used only with graph cloned with [pi\\_sfu\\_graph\\_dynamic\\_clone\(\)](#).

#### Parameters

- **graph** – [in] : Pointer to the cloned graph descriptor

**Returns** Error code : 0 - Success

```
void pi_sfu_graph_load(pi_sfu_graph_t *graph)
```

Apply patch information of the cloned graph to the L2 config of the original graph. To be called before loading next cloned graph.

Load an sfu graph.

This will load the graph from L2 to sfu and start it, so that it is ready to process data.

---

**Note:** This function can not be called from an interrupt handler.

---

#### Parameters

- **graph** – [in] : Pointer to the cloned graph descriptor
- **graph** – [in] A pointer to the graph instance.

**Returns** Error code : 0 - Success

```
void pi_sfu_graph_load_async(pi_sfu_graph_t *graph, pi_evt_t *task)
```

Load a graph asynchronously.

This will load the graph from L2 to sfu and start it, so that it is ready to process data. This operation is done asynchronously, which can be interesting since this can take quite a long time for big graphs. The end of the load operation is notified through a pmsis task, which can be a callback, a blocking task, or an interrupt handler. See pmsis task documentation for more details.

#### Parameters

- **graph** – [in] A pointer to the graph instance.
- **task** – The task used to notify the end of the load.

```
void pi_sfu_graph_unload(pi_sfu_graph_t *graph)
```

Unload an sfu graph.

This will unload the graph from the SFU. The graph can be loaded again later on from L2 to sfu. Until it is loaded again, the graph cannot be used.

#### Parameters

- **graph** – [in] A pointer to the graph instance.

```
void pi_sfu_graph_end_wait(pi_sfu_graph_t *graph)
```

Wait for the end of an sfu graph.

This will wait until the graph has finished executing. This can happen in case of an error, or when the graph has no more input samples to process.

---

**Note:** This function can not be called from an interrupt handler.

---

**Parameters**

- **graph** – [in] A pointer to the graph instance.

```
void pi_sfu_graph_end_wait_async(pi_sfu_graph_t *graph, pi_evt_t *task)
```

Get notified by the end of an sfu graph.

This will register a pmsis task, which will be triggered when the graph has finished executing. This can happen in case of an error, or when the graph has no more input samples to process. The end of the graph is notified through a pmsis task, which can be a callback, a blocking task, or an interrupt handler. See pmsis task documentation for more details.

**Parameters**

- **graph** – [in] A pointer to the graph instance.
- **task** – The task used to notify the end of the graph.

```
int pi_sfu_graph_bind_pdm(pi_sfu_graph_t *graph, int node_id, pi_sfu_pdm_itf_id_t *itf_id)
```

Update SFU PDM\_IN or PDM\_OUT block configuration with SAI interface and channel to which they shall be connected. Doesn't take effect until next SFU graph load.

**Parameters**

- **graph** – [in] Pointer to the graph instance
- **node\_id** – [in] Node index in SFU descriptor list. This list is previously generated by SFUConfigurationGenerator and indexes of configurable blocks is provided in generated file <graph\_name>\_L2\_Descr.h.
- **itf\_id** – [in] Description of PDM interface channel (SAI, channel, tx/rx)

**Returns 0** : Success

**Returns Other** : Failure

**Returns** Error code

```
int pi_sfu_graph_bind_i2s(pi_sfu_graph_t *graph, int node_id, pi_sfu_i2s_itf_id_t *itf_id, int *stream_ch)
```

Unlike [\*pi\\_sfu\\_graph\\_bind\\_pdm\*](#), it doesn't modify configuration of SFU STREAM\_IN or STREAM\_OUT blocks, rather it just sets physical clock source given by 'itf\_id' to the SFU clock domain in which given STREAM\_IN or STREAM\_OUT node 'node\_id' belongs. Doesn't take effect until next SFU graph load.

**Parameters**

- **graph** – [in] Pointer to the graph instance
- **node\_id** – [in] Node index in SFU descriptor list. This list is previously generated by SFUConfigurationGenerator and indexes of configurable blocks is provided in generated file <graph\_name>\_L2\_Descr.h.
- **itf\_id** – [in] Description of I2S interface channel (SAI, tx/rx)
- **stream\_ch** – [out] Outputs uDMA stream ID of the channel assigned to 'node\_id'. This assignment is normally already done during graph load.

**Returns 0** : Success

**Returns Other** : Failure

**Returns** Error code

---

```
void pi_sfu_dump_l2_graph(pi_sfu_graph_t *graph)
    Dump raw L2 configuration of the SFU graph given by ‘graph’ to stdout.
```

**Parameters**

- **graph** – [in] Pointer to a SFU graph.

```
void pi_sfu_dump_l2_graph_patch(pi_sfu_graph_t *graph)
```

Dump cloned graph’s dynamic patch information. It shows diff elements with respect to the L2 config of the original graph.

**Parameters**

- **graph** – [in] Pointer to a SFU graph.

```
int pi_sfu_get_stream_id(pi_sfu_graph_t *graph, int node_id)
```

Get uDMA stream ID assigned to SFU stream block ‘node’ id’. The ID has already been allocated by SFU driver during graph opening.

**Parameters**

- **graph** – [in] Pointer to a SFU graph.
- **node\_id** – [in] Index of SFU node in generated node descriptor list.

```
pi_sfu_mem_port_t *pi_sfu_get_mem_port(pi_sfu_graph_t *graph, int id)
```

Get reference to mem in/out port context from node descriptor id.

**Parameters**

- **graph** – [in] Pointer to the SFU graph instance
- **id** – [in] Node index in SFU descriptor list

**Returns** Pointer to mem in/out port context

```
static inline void pi_sfu_buffer_init(pi_sfu_buffer_t *buffer, void *data, int nb_samples, int sample_size)
```

Initialize a data buffer.

Data buffers are containers for samples. They must be initialized with this function before they can be enqueued to a graph. A data buffer must be kept alive as long as the samples are used by a graph. A pointer to the memory for the samples must be given and must also be kept alive as long as the samples are used by a graph

**Parameters**

- **buffer** – [in] A pointer to the data buffer.
- **data** – [in] A pointer to the memory used for the samples.
- **nb\_samples** – [in] Number of samples that the buffer can contain.
- **sample\_size** – [in] Size in bytes of the samples.

```
static inline void pi_sfu_enqueue(pi_sfu_graph_t *graph, pi_sfu_mem_port_t *port, pi_sfu_buffer_t
                                *buffer)
```

Enqueue a data buffer to an sfu graph memin or memout.

This fonction can be used to transfer samples between an sfu graph and the L2. For memin, the data buffer is pushed to a graph input and is forwarded to the sfu block connected to this input. For memout, the data buffer is pushed to a graph output and samples coming from the sfu block connected to this output are written to the buffer. Only 2 buffers can be pushed at the same time. A new buffer can be pushed when once has finished.

**Parameters**

- **graph** – [in] A pointer to the data buffer.
- **port** – [in] Pointer to mem in/out port descriptor
- **buffer** – [in] The data buffer to be enqueued.

```
static inline void *pi_sfu_buffer_data(pi_sfu_buffer_t *buffer)
```

Get buffer data memory.

Return the pointer to the memory of samples given when the buffer was initialized.

#### Parameters

- **buffer** – [in] The data buffer.

**Returns** The pointer to the sample memory.

```
void pi_sfu_graph_save(pi_sfu_graph_t *graph)
```

Save an SFU graph.

Save the state of an sfu graph. The state is saved to L2. This can be used to switch to a different graph and come back to this one later on. The state will be automatically restored when the graph is loaded.

---

**Note:** This function can not be called from an interrupt handler.

---

#### Parameters

- **graph** – [in] The graph to be saved.

```
void pi_sfu_graph_save_async(pi_sfu_graph_t *graph, pi_evt_t *task)
```

Save an SFU graph asynchronously.

Save the state of an sfu graph. The state is saved to L2. This can be used to switch to a different graph and come back to this one later on. The state will be automatically restored when the graph is loaded.

- This operation is done asynchronously, which can be interesting since this can take quite a long time for big graphs. The end of the save operation is notified through a pmsis task, which can be a callback, a blocking task, or an interrupt handler. See pmsis task documentation for more details.

#### Parameters

- **graph** – [in] The graph to be saved.
- **task** – [in] The task used to notify the end of the save.

```
void pi_sfu_graph_reconfigure(pi_sfu_graph_t *graph)
```

Reconfigure GFU filters in an SFU graph.

Send RECONF command for ‘graph’ to the SFU and wait for it to finish.

On RECONF command, SFU will reload coefficients from L2 for all GFU filters that belong to the ‘graph’ and that are flagged as reconfigurable.

•

---

**Note:** This function cannot be called from an interrupt handler.

---

#### Parameters

- **graph** – [in] The graph to be reconfigured.

```
void pi_sfu_graph_reconfigure_async(pi_sfu_graph_t *graph, pi_evt_t *task)
    Reconfigure GFU filters in an SFU graph asynchronously.
```

Send RECONF command for ‘graph’ to the SFU and optionally register a ‘task’ to be notified when the reconfiguration is done. On RECONF command, SFU will reload coefficients from L2 for all GFU filters that belong to the ‘graph’ and that are flagged as reconfigurable. The end of the reconfiguration operation is notified through a pmsis task, which can be a callback, a blocking task, or an interrupt handler. See pmsis task documentation for more details.

#### Parameters

- **graph** – [in] The graph to be reconfigured.
- **task** – [in] The task used to notify the end of the reconfiguration.

```
int pi_sfu_graph_set_filter_coeffs(pi_sfu_graph_t *graph, int node_id, unsigned int *coeffs)
    Set pointer to filter coefficients of a GFU filter.
```

The function finds L2 configuration of the GFU filter node ‘node\_id’ in ‘graph’ and updates COEFF\_PTR field with given pointer ‘coeffs’. It does not interact with SFU.

#### Parameters

- **graph** – [in] Pointer to the graph context
- **node\_id** – [in] Node descriptor index in generated graph descriptor.
- **coeffs** – [in] Pointer to filter coefficient data. Coefficients must be prepared in SFU format. Each type of filter has its own layout of coefficients, normalization shifts and other parameters that may be part of this configuration (see SFU documentation for details). New coefficient set must preserve previous filter size as SFU does not support changing filter size on the fly.

```
int pi_sfu_graph_get_gfu_filter_refs(pi_sfu_graph_t *graph, int sfugen_id, pi_sfu_gfu_filter_refs_t
                                         *filter_refs)
```

```
unsigned int pi_sfu_asrc_ratio_get(pi_sfu_graph_t *graph, unsigned int node_id)
    Get current value of ASRC frequency ratio (Fout / Fin).
```

If ASRC is in automatic mode, function returns current ratio estimation (RO register). If ASRC is in manual mode, function returns value configured value (RW register).

#### Parameters

- **graph** – [in] Pointer to the graph context
- **node\_id** – [in] ASRC node descriptor index

**Returns** ASRC ratio in Q4.22 fixed-point format.

```
int pi_sfu_asrc_ratio_set(pi_sfu_graph_t *graph, int node_id, unsigned int ratio)
    Set ASRC Conversion Ratio.
```

Set the ASRC Conversion Ratio in manual mode

#### Parameters

- **graph** – [in] Pointer to the graph context
- **node\_id** – [in] ASRC node descriptor index
- **ratio** – [in] ASRC conversion ratio (Fout/Fin) in Q4.22 fixed-point format

```
int pi_sfu_status_asrc_lock_get(pi_sfu_graph_t *graph, int node_id)
```

Get ASRC frequency tracking lock status for ‘node\_id’ of the graph ‘graph’.

**Parameters**

- **graph** – [in] Pointer to the graph context
- **node\_id** – [in] ASRC node descriptor index

**Returns 0** Not locked

**Returns 1** Locked

**Returns -1** Error

**Returns** Lock status of ASRC node ‘node\_id’.

```
int pi_sfu_audio_clock_set(uint32_t phy_clock_id, uint32_t enable, uint32_t frequency)
```

Enable/disable SFU audio clock and set its frequency. This function ignores allocation status of the clock that may be (or not) done by SFU\_AllocateAudioClock() and writes directly to one of AUDIO\_CLK\_CFG\_[i] registers. Clock : Must be in range from CLK\_AUD0 to CLK\_AUD3.

**Returns** 0 on success and -1 on failure.

```
void pi_audio_conf_init(struct pi_audio_conf *conf)
```

Initialize an audio framework configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the audio framework is opened.

**Parameters**

- **conf** – [in] A pointer to the audio framework configuration.

```
int pi_audio_open(pi_audio_conf_t *conf)
```

Open the audio framework.

This will allocate and initialize all the resources needed to use the audio framework. Once this function is successfully called, other functions for managing graphs can be called.

**Parameters**

- **conf** – [in] A pointer to the audio framework configuration.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

```
void pi_audio_close()
```

Close the audio framework.

This will free all resources used by the audio framework. After a call to this function, the audio framework must be opened before it can be used again.

```
pi_audio_graph_t *pi_audio_graph_open(pi_audio_graph_desc_t *graph_desc)
```

Open a graph.

This will open a graph and instantiate it. During this step, the audio framework will go through the graph and allocate any resource needed by the graph. The graph must be started after this operation in order to start execution.

**Parameters**

- **graph\_desc** – [in] A pointer to the graph descriptor, generated by the audio toolchain.

**Returns** A pointer to the graph instance or NULL if it failed.

---

```
void pi_audio_graph_close(pi_audio_graph_t *graph)
    Close a graph.
```

This will free all resources allocated for a graph. After a call to this function, the graph can not be executed anymore.

#### Parameters

- **graph** – [in] A pointer to the graph instance.

```
void pi_audio_graph_start(pi_audio_graph_t *graph)
    Start a graph.
```

This will start the graph execution, which is then ready to process data.

#### Parameters

- **graph** – [in] A pointer to the graph instance.

```
void pi_audio_graph_stop(pi_audio_graph_t *graph)
    Stop a graph.
```

This will make sure a graph can not execute anymore. The graph can be started again.

#### Parameters

- **graph** – [in] A pointer to the graph instance.

```
void pi_audio_buffer_init(pi_audio_buffer_t *buffer, void *data, int nb_samples, int sample_size)
    Initialize a data buffer.
```

Data buffers are containers for samples. They must be initialized with this function before they can be enqueued to a graph. A data buffer must be kept alive as long as the samples are used by a graph. A pointer to the memory for the samples must be given and must also be kept alive as long as the samples are used by a graph

#### Parameters

- **buffer** – [in] A pointer to the data buffer.
- **data** – [in] A pointer to the memory used for the samples.
- **nb\_samples** – [in] Number of samples that the buffer can contain.
- **sample\_size** – [in] Size in bytes of the samples.

```
void pi_audio_input_enqueue(pi_audio_graph_t *graph, int input, pi_audio_buffer_t *buffer, pi_evt_t
                           *task)
```

Enqueue a data buffer to a graph input.

This fonction can be used to push samples to a graph. The data buffer is pushed to a graph input and is forwarded to the graph component connected to this input. The component will be notified samples are ready so that it can start processing them. As many buffer as needed can be enqueued to the same input, they will be put in a queue, and handled in order, which can be convenient to make sure there is no hole between the processing of 2 buffers. The end of buffer processing is managed through a PMSIS task. Once the buffer is empty, the task is triggered so that the caller is notified. This can be done either through function callback, IRQ handler callback or blocking task. See PMSIS documentation for more information.

#### Parameters

- **graph** – [in] A pointer to the data buffer.
- **input** – [in] The index of the graph input where the buffer is enqueued.
- **buffer** – [in] The data buffer to be enqueued.

- **task** – [in] PMSIS task used for completion.

```
void pi_audio_output_enqueue(pi_audio_graph_t *graph, int output, pi_audio_buffer_t *buffer, pi_evt_t  
                            *task)
```

Enqueue a data buffer to a graph output.

This fonction can be used to pop samples from a graph. The data buffer is pushed to a graph output and is forwarded to the graph component connected to this output. The component will be notified a new buffer is ready to received output samples. As many buffer as needed can be enqueued to the same output, they will be put in a queue, and handled in order, which can be convenient to make sure there is no hole between the processing of 2 buffers. The end of buffer processing is managed through a PMSIS task. Once the buffer is full, the task is triggered so that the caller is notified. This can be done either through function callback, IRQ handler callback or blocking task. See PMSIS documentation for more information.

#### Parameters

- **graph** – [in] A pointer to the data buffer.
- **input** – [in] The index of the graph output where the buffer is enqueued.
- **buffer** – [in] The data buffer to be enqueued.
- **task** – [in] PMSIS task used for completion.

```
static inline void *pi_audio_buffer_data(pi_audio_buffer_t *buffer)
```

Get buffer data memory.

Returns The pointer to the memory of samples given when the buffer was initialized.

#### Parameters

- **buffer** – [in] The data buffer.

Returns The pointer to the sample memory.

```
int pi_audio_graph_get_input_by_name(pi_audio_graph_t *graph, char *port_name)
```

Get a graph input index from its name.

The name should be the one used in the graph. The returned index is the one used on all the API functions, like the one for enqueueing buffers. This function is going through a symbol table, and so should only be used once after the graph has been opened.

#### Parameters

- **graph** – [in] The graph containing the input.
- **port\_name** – [in] The name of the graph input.

Returns The graph input index.

```
int pi_audio_graph_get_output_by_name(pi_audio_graph_t *graph, char *port_name)
```

Get a graph output index from its name.

The name should be the one used in the graph. The returned index is the one used on all the API functions, like the one for enqueueing buffers. This function is going through a symbol table, and so should only be used once after the graph has been opened.

#### Parameters

- **graph** – [in] The graph containing the output.
- **port\_name** – [in] The name of the graph output.

Returns The graph output index.

---

```
void pi_audio_configure_node(pi_audio_graph_t *graph, int input_id, int policy, void *params)
Configure a node.
```

This function can be used to configure the node connected to the specified input. The target node configuration handler will be called with the specific params. The effect of calling this function depends on the target node, refer to the node documentation for more information.

#### Parameters

- **graph** – [in] The graph containing the node to be configured.
- **input\_id** – [in] The input index where the node is connected.
- **policy** – [in] Configuration scheduling policy (see PI\_AUDIO\_CFG\_POLICY\_... macros)
- **params** – [in] The parameters to be passed to the node configuration handler.

```
int pi_audio_graph_reconfigure_async(pi_audio_graph_t *graph, int sfu_graph_id, pi_evt_t *task)
Trigger graph reconfiguration.
```

Reconfiguration is done in 2 steps. First nodes are reconfigured, and finally the new configuration is applied at graph-level. This function can be called to do the second step, i.e. commit the new configuration in the whole graph. Since the reconfiguration can be long, its termination can be notified through a PMSIS task.

#### Parameters

- **graph** – [in] The graph to reconfigure.
- **sfu\_graph\_id** – [in] The SFU graph ID.
- **task** – [in] The PMSIS task used to notify the end of reconfiguration.

```
void pi_audio_set_volume(pi_audio_graph_t *graph, int input_id, int mantissa, int exponent)
Set output volume (gain) of a node via public port ‘input_id’ accessible to the ‘graph’. Volume value should be given in SFU mantissa/exponent format.
```

#### Parameters

- **graph** – [in] Graph containing the node where volume should be set
- **input\_id** – [in]
- **mantissa** – [in] Positive value in range [0 .. (2^26 - 1)]
- **exponent** – [in] Signed value in range [-31 .. 31]

```
struct pi_sfu_conf
#include <sfu_pmsis_runtime.h> SFU configuration structure.
```

This structure is used to pass the desired sfu configuration to the runtime when opening it.

```
struct pi_audio_conf
#include <gap_audio_framework.h> Audio framework configuration structure.
```

This structure is used to pass the desired audio framework configuration to the runtime when opening it.

## Public Members

`pi_device_t *fs`

Specifies the file system where the graphs should be loaded from. NULL by default.

## 6.1.7 Examples

Examples are organized in several groups:

- **applications**

Full-stack examples that show work flow from high-level graph to its execution on the target:

- graph creation in one of the front end tools
- application code that compiles for the target together with code generated by the toolchain
- checkers to verify that the application executed on the target has produced expected outputs

Applications are using audio framework runtime API and not legacy SFU\_RT API.

- **frontend**

Examples that show various features of high-level graph creation by one of the front end tools.

- **archive**

Legacy examples based on SFU\_RT API and other deprecated features.

## Applications

### Application examples

#### Filters

Examples in this directory show usage of all filters currently supported by the library. Each example is based on a simple graph with one filter node and memory input/output ports.

Application enqueues one buffer on the input and one on the output (one-shot). *For example on how to enqueue continuous stream of buffers can be found in another directory: “streaming”.*

Examples here shows how to make a graph in three different ways:

- by Python script
- by C++ program
- by Graph Designer

Filter coefficients have been designed with Graph Designer. File generators/graph.gdg can be opened in GD to inspect filter characteristics in GUI mode.

Each example contains following files:

- `./example.c` - GAP9 target application C code
- `./Makefile` - Recipes to make graph with various options and to build and run it on target

- `./testset` - Regression test list. Also can be used to view supported options.
- `generators/graph.gdg` - Graph Designer project file.

This file cannot be used directly by the Toolchain. After changes have been made in GD, the project must be exported as a `.graph` file by “Publish” command of the GD. `.graph` file can be then fed to the toolchain by `make ggraph` command.

- `generators/graph.py` - Python variant of describing the graph (uses Python component classes)
- `generators/graph.cpp` - C++ variant of describing the graph (uses C++ component classes)
- `generators/CMakeLists.txt` - Cmake file for builing C++ graph generator
- `utils/check.py` - Script to validate outputs of grpah execution
- `utils/golden.raw` - Expected output samples (in 32-bit fixed-point format)
- `utils/input.raw` - Input samples (in 32-bit fixed-point format)
- `utils/show_spectrum.m` - Matlab/Octave script to plot output signal spectrum

## Usage

Graph design and generaton of runtime code can be done by using one of three methods:

- `make pgraph` - Generates graph with Python script
- `make cgraph` - Generates graph by compiling and running C++ application
- `make ggraph` - Generates runtime code from a graph published by Graph Designer in `.graph` format
- For some filters, LOCATION flag can be used with `pgraph` and `cgraph` Make targets to select filter implementation locaion (`sfu`, `fc` or `cluster`). See README of a particular filter for supported options.

Example: `make pgraph LOCATION="fc"`

The example can then be compiled and run with:

`make all run`

The output signal can be checked against a golden output file with:

`make check`

The spectrum of the output signal can be shown with (Octave installation required):

`make spectrum`

## List of filters

### Biquad cascade DF1 filter

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`, `fc`.

**Biquad cascade DF2 filter**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`, `fc`.

**FIR filter**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`, `fc`, `cluster`.

**IIR DF1 filter**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

**Warped-FIR filter**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

**Lattice-FIR filter graph**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

**Lattice-Allpole filter**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

**Lattice-ladder filter graph**

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

## Farrow structure

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

## Horner structure

See [Filters](#) page for an overview and usage.

Supported LOCATION values: `sfu`.

## PCM example

This example will just inject a PCM file (binary audio file with samples in 32-bit integer format) to a simple biquad graph and dump the output to another PCM file.

Execute `make graph` to generate the C descriptor from the graph designer graph , `make gen all run` to compile and run the example and `make spectrum` to display the spectrum of the output file.

## FIR filter graph reconfiguration with I2S streaming

### Introduction

This is an example of reconfiguration of a single FIR filter graph.

The data path of the graph is connected to I2S for both input and output. The output is also sent to memory so that a reconfiguration phase is triggered periodically with a period of a fixed number of samples.

Coefficients reconfiguration is done in 2 steps. First they are updated in memory. This step is not impacting the current execution, which is still using the previous set of coefficients. The second step is committing the new coefficients to the filter, and it's only after this step that the filter will be impacted by the new coefficients.

For that, everytime an output buffer termination is notified, the previous set of coefficients is committed, so that the next samples are handled with the new set. Once the commit operation is done, a new set of coefficients is updated in memory and is ready for the commit when the next buffer termination occurs.

This example supports both the description of the graph with a python graph or a C++ program.

The filter has been designed with Graph Designer and can be opened by opening the file `generators/graph.gdg` from Graph Designer

### Usage

For the python graph, the graph can be generated with:

```
make pgraph
```

For the C++ program, the graph can be generated with:

```
make cgraph
```

The example can then be compiled and run with:

```
make all run
```

This example needs to execute a script to inject input stimuli to the I2S and extract the output to a file:

```
make all run
```

The results can be checked against a golden output file with:

```
make check
```

The spectrum can be shown with:

```
make spectrum
```

## GVSOC support

On GVSOC, input samples can be read from a file and sent to I2S, and output samples can be captured and written to an output file.

For that the execution must be launched with this option:

```
make all run USE_GVSOC_TESTBENCH=1
```

This will open a GVSOC proxy and wait for an incoming connection.

From another terminal with the SDK configured, a GVSOC script can be launched with this command to interact with the I2S:

```
make proxy
```

This is launching a python script which is interacting with GVSOC to control the I2S.

## Streaming

Streaming directory contains examples for different variants to process continuous stream of audio samples.

### FIR filter graph with file streaming

#### Introduction

This is an example of a simple graph with a single FIR filter in a streaming use case where the input samples are continuously read from flash and the output samples continuously written to flash.

This examples enqueues input buffers to the flash in a double-buffering ways so that there is no missing sample between 2 buffers.

This example supports both the description of the graph with a python graph or a C++ program.

The filter has been designed with Graph Designer and can be opened by opening the file generators/graph.gdg from Graph Designer

## Usage

For the python graph, the graph can be generated with:

```
make pgraph
```

For the C++ program, the graph can be generated with:

```
make cgraph
```

The example can then be compiled and run with:

```
make all run
```

The results can be checked against a golden output file with:

```
make check
```

The spectrum can be shown with:

```
make spectrum
```

## FIR filter graph with I2S streaming

### Introduction

This is an example of a simple graph with a single FIR filter in a streaming use case where the input samples are continuously read from the I2S and the output samples continuously written to the I2S in the case where the graph is an SFU graph.

This example does not deal at all with buffers since the I2S is directly connected to the SFU by the hardware. It just binds the graph to the I2S and let the hardware take care of the streaming.

This example supports both the description of the graph with a python graph or a C++ program.

The filter has been designed with Graph Designer and can be opened by opening the file generators/graph.gdg from Graph Designer

## Usage

For the python graph, the graph can be generated with:

```
make pgraph
```

For the C++ program, the graph can be generated with:

```
make cgraph
```

The example can then be compiled and run with:

```
make all run
```

This example needs to execute a script to inject input stimuli to the I2S and extract the output to a file. To do this, first run the application with USE\_GVSOC\_TESTBENCH=1:

```
USE_GVSOC_TESTBENCH=1 make run
```

Then, in another terminal, run script that controls gvsoc testbench:

```
USE_GVSOC_TESTBENCH=1 make proxy
```

Alternatively, use run\_dual target to execute both in the same terminal:

```
make run_dual RUN_CMD="make run USE_GVSOC_TESTBENCH=1" PROXY_CMD="make proxy"
```

The results can be checked against a golden output file with:

```
make check
```

The spectrum can be shown with:

```
make spectrum
```

## FIR filter graph with PDM streaming

### Introduction

This is an example of a simple graph with a single FIR filter in a streaming use case where the input samples are continuously read from the PDM input and the output samples continuously written to the PDM output in the case where the graph is an SFU graph.

This example does not deal at all with buffers since the PDM is directly connected to the SFU by the hardware. It just binds the graph to the PDM and let the hardware take care of the streaming.

This example supports both the description of the graph with a python graph or a C++ program.

The filter has been designed with Graph Designer and can be opened by opening the file generators/graph.gdg from Graph Designer

### Usage

For the python graph, the graph can be generated with:

```
make pgraph
```

For the C++ program, the graph can be generated with:

```
make cgraph
```

The example can then be compiled and run with:

```
make all run
```

This example needs to execute a script to inject input stimuli to the I2S and extract the output to a file. To do this, first run the application with USE\_GVSOC\_TESTBENCH=1:

```
USE_GVSOC_TESTBENCH=1 make run
```

Then, in another terminal, run script that controls gvsoc testbench:

```
USE_GVSOC_TESTBENCH=1 make proxy
```

Alternatively, use run\_dual target to execute both in the same terminal:

```
make run_dual RUN_CMD="make run USE_GVSOC_TESTBENCH=1" PROXY_CMD="make proxy"
```

The results can be checked against a golden output file with:

```
make check
```

The spectrum can be shown with:

```
make spectrum
```

## SFU features

This directory contains examples of SFU blocks and features other than filters.

### Limiter graph

#### Introduction

This is an example of a simple graph with a Limiter.

The example just enqueues one buffer on the input and one on the output. Look at stream dedicated examples to see how to enqueue a continuous stream of buffers.

The example shows making the graph in three different ways:

- by Python script
- by C++ program
- by Graph Designer

The filter has been designed with Graph Designer and can be opened by opening the file generators/graph.gdg from Graph Designer.

#### Files:

- `./example.c` - GAP9 target application C code
- `./Makefile` - Recipes to make graph with various options and to build and run it on target
- `./testset` - Regression test list
- `generators/graph.gdg` - Graph Designer project file.

This file cannot be used directly by the Toolchain. After changes have been made in GD, the project must be exported as a `.graph` file by “Publish” command of the GD. `.graph` file can be then fed to the toolchain by `make ggraph` command.

- `generators/graph.py` - Python variant of describing the graph (uses Python component classes)
- `generators/graph.cpp` - C++ variant of describing the graph (uses C++ component classes)
- `generators/CMakeLists.txt` - cmake file for builing C++ graph generator
- `utils/check.py` - Script to validate outputs of grpah execution
- `utils/golden.raw` - Expected output samples (in 32-bit fixed-point format)
- `utils/input.raw` - Input samples (in 32-bit fixed-point format)
- `utils/show_spectrum.m` - Matlab/Octave script to plot output signal spectrum

## Usage

Graph design and generation of run time code can be done by using one of three methods: Python, C++, GD.

Generate graph with Python generator and produce runtime descriptors:

```
make pgraph
```

Generate graph with C++ generator and produce runtime descriptors:

```
make cgraph
```

Produce runtime descriptors from graph published by Graph Designer in .graph format:

```
make ggraph
```

The example can then be compiled and run with:

```
make all run
```

The output signal can be checked against a golden output file with:

```
make check
```

The spectrum of the output signal can be shown with:

```
make spectrum
```

## PDM graph

This example is using graph with two PDM inputs and two PDM outputs. Application code in `example.c` shows how to use framework runtime API to get references to PDM ports of the graph and to connect them to PDM interfaces that reside in SAI hardware blocks.

## Polyphase cascade graph

### Introduction

This is an example of a simple graph with a single Polyphase cascade node.

The example just enqueues one buffer on the input and one on the output. Contrary most examples of this type (one-shot memory to memory), input and output buffers of the graph are of different size because of different sample rate on input vs output port. Buffer sizes can be controlled with environment variables `NB_SAMPLES_IN` and `NB_SAMPLES_OUT` which are set to appropriate values in the `Makefile`.

The example shows making the graph in three different ways:

- by Python script
- by C++ program
- by Graph Designer

The filter has been designed with Graph Designer and can be opened by opening the file `generators/graph.gdg` from Graph Designer.

**Files:**

- `./example.c` - GAP9 target application C code
- `./Makefile` - Recipes to make graph with various options and to build and run it on target
- `./testset` - Regression test list
- `generators/graph.gdg` - Graph Designer project file.

This file cannot be used directly by the Toolchain. After changes have been made in GD, the project must be exported as a `.graph` file by “Publish” command of the GD. `.graph` file can be then fed to the toolchain by `make ggraph` command.

- `generators/graph.py` - Python variant of describing the graph (uses Python component classes)
- `generators/graph.cpp` - C++ variant of describing the graph (uses C++ component classes)
- `generators/CMakeLists.txt` - cmake file for builing C++ graph generator
- `utils/check.py` - Script to validate outputs of grpah execution
- `utils/golden.raw` - Expected output samples (in 32-bit fixed-point format)
- `utils/input.raw` - Input samples (in 32-bit fixed-point format)
- `utils/show_spectrum.m` - Matlab/Octave script to plot output signal spectrum

**Usage**

Graph design and generaton of run time code can be done by using one of three methods: Python, C++, GD.

Generate graph with Python generator and produce runtime descriptors:

```
make pgraph
```

Generate graph with C++ generator and produce runtime descriptors:

```
make cgraph
```

Produce runtime descriptors from graph published by Graph Designer in `.graph` format:

```
make ggraph
```

The example can then be compiled and run with:

```
make all run
```

The output signal can be checked against a golden output file with:

```
make check
```

The spectrum of the output signal can be shown with:

```
make spectrum
```

## Save/restore

### Introduction

This is an example of a graph save/restore feature of SFU. Save/resotre enables usage of SFU in time-sharing on graph level. Application shows switching in and out 2 graphs that use same resources in SFU while saving state of the graph being switched out to the L2 so it can resume later.

The example shows making the graph in three different ways:

- by Python script
- by C++ program
- by Graph Designer

The filter has been designed with Graph Designer and can be opened by opening the file generators/graph.gdg from Graph Designer.

### Files:

- `./example.c` - GAP9 target application C code
- `./Makefile` - Recipes to make graph with various options and to build and run it on target
- `./testset` - Regression test list
- `generators/graph.gdg` - Graph Designer project file.

This file cannot be used directly by the Toolchain. After changes have been made in GD, the project must be exported as a .graph file by “Publish” command of the GD. .graph file can be then fed to the toolchain by `make ggraph` command.

- `generators/graph.py` - Python variant of describing the graph (uses Python component classes)
- `generators/graph.cpp` - C++ variant of describing the graph (uses C++ component classes)
- `generators/CMakeLists.txt` - cmake file for builing C++ graph generator
- `utils/check.py` - Script to validate outputs of grpah execution
- `utils/golden.raw` - Expected output samples (in 32-bit fixed-point format)
- `utils/input.raw` - Input samples (in 32-bit fixed-point format)
- `utils/show_spectrum.m` - Matlab/Octave script to plot output signal spectrum

### Usage

Graph design and generaton of run time code can be done by using one of three methods: Python, C++, GD.

Generate graph with Python generator and produce runtime descriptors:

```
make pgraph
```

Generate graph with C++ generator and produce runtime descriptors:

```
make cgraph
```

Produce runtime descriptors from graph published by Graph Designer in .graph format:

```
make ggraph
```

The example can then be compiled and run with:

```
make all run
```

The output signal can be checked against a golden output file with:

```
make check
```

The spectrum of the output signal can be shown with:

```
make spectrum
```

## Stream I2s graph

This example is using graph with two SFU stream inputs and two SFU stream outputs. Application code in `example.c` shows how to use framework runtime API to get references to stream ports of the graph and to connect them (via uDMA streams) to I2S interfaces that reside in SAI hardware blocks.

### Volume control

#### Introduction

This is an example of volume control of a FIR filter in SFU. Volume control is managed via a named control port “fir\_volume” on graph boundary that is accessible by the application.

The example shows making the graph in three different ways:

- by Python script
- by C++ program
- by Graph Designer

The filter has been designed with Graph Designer and can be opened by opening the file `generators/graph.gdg` from Graph Designer.

#### Files:

- `./example.c` - GAP9 target application C code
- `./Makefile` - Recipes to make graph with various options and to build and run it on target
- `./testset` - Regression test list
- `generators/graph.gdg` - Graph Designer project file.

This file cannot be used directly by the Toolchain. After changes have been made in GD, the project must be exported as a `.graph` file by “Publish” command of the GD. `.graph` file can be then fed to the toolchain by `make ggraph` command.

- `generators/graph.py` - Python variant of describing the graph (uses Python component classes)
- `generators/graph.cpp` - C++ variant of describing the graph (uses C++ component classes)
- `generators/CMakeLists.txt` - cmake file for builing C++ graph generator
- `utils/check.py` - Script to validate outputs of grpah execution
- `utils/golden.raw` - Expected output samples (in 32-bit fixed-point format)
- `utils/input.raw` - Input samples (in 32-bit fixed-point format)

- `utils/show_spectrum.m` - Matlab/Octave script to plot output signal spectrum

### Usage

Graph design and generation of run time code can be done by using one of three methods: Python, C++, GD.

Generate graph with Python generator and produce runtime descriptors:

```
make pgraph
```

Generate graph with C++ generator and produce runtime descriptors:

```
make cgraph
```

Produce runtime descriptors from graph published by Graph Designer in .graph format:

```
make ggraph
```

The example can then be compiled and run with:

```
make all run
```

The output signal can be checked against a golden output file with:

```
make check
```

The spectrum of the output signal can be shown with:

```
make spectrum
```

## Frontend

### Frontend examples

TODO

## Archive

### Graph designer example

This example shows utilization of the Toolchain to generate SFU graph from a graph produced by Graph Designer. It supports graphs of the form MEM\_IN.0 -> (custom nodes) -> MEM\_OUT.0.

Input samples are taken from file bbsync2048\_24 dbl, a noise signal that occupies full audio spectrum [0Hz - 24Hz] so that the output side of the graph would produce a signal that shows frequency response of the graph.

### Usage

Generate C source for target that contains graph configuration translated for the SFU: `make graph`

Build and execute the example on default platform: `make all run`

View spectrum of the output signal in Octave (requires Octave installed): `make spectrum`

To run other examples, edit Makefile to put desired file name. Default example is bq\_biquad.graph: MEM\_IN.0 -> GFU.0.Biquad -> MEM\_OUT.0.

### 6.1.8 Glossary

- GD - Graph Designer
- AF - Audio Framework
- AMT - Audio Mapping Tool
- RTG - Runtime Generator
- TC - Toolchain (AMT + RTG)
- RNT - Runtime library

## 6.2 Autotiler

The Autotiler is a C library runs on users PCs before GAP code compilation and automatically generates GAP code for memory tiling and transfers between all memory levels: GAP has two levels in-chip memory (L1 and L2) and an external optional 3rd level (L3). The code generated makes use of micro-DMA (uDMA) which transfers data from external memories (Flash and L3) to internal L2 memory and of cluster-DMA (DMA) which transfer data from L2 to L1. The Autotiler is part of the GapFlow to convert ONNX and TFLite Models onto GAP code.

From a user perspective, the purpose of the auto-tiler is twofold:

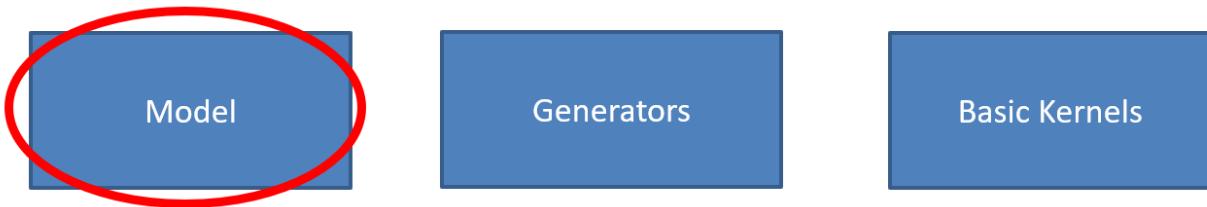
1. Use the Generators provided in the GAP SDK that contains several ready to go, optimized algorithms, like CNN/RNN layers, sound and image processing.
2. Create new generators with your own algorithms.

For several use cases the former will be enough to build your own application. The simplest way to use the Autotiler is to exploit the existing generators and just write your own model (or generate the model with NNtool).

### 6.2.1 Use generators provided within the SDK

To exploit generators provided in the sdk, the user needs to write his own model.

This is what a user needs to code



**Model contains configuration elements:**

- available memory
- compilation hints
- consumed and produced files,
- basic kernels loaded as libraries
- an ordered list of user kernels and/or user kernel groups

**Collection of User Kernel**

- A user kernel defines the different ways in which 2, 3 or 4-dimensional data is traversed
- one or more basic kernels are called.
- A user kernel model takes arguments that describe the input, working and output data that needs to be modeled.

**Pure C functions**

- written by the developer as if all the data structures they access can fit into shared level 1 memory

In the SDK we provide several examples on how to write your own model:

```
cd $GAP_SDK_HOME/examples/gap9/nn/autotiler/
```

All the generators source code can be found here:

```
cd $GAP_SDK_HOME/tools/autotiler_v3/
```

You can browse it here on [Github](#).

## 6.2.2 Autotiler Models

An Autotiler model is a list of calls to generators that compose the processing that you want to perform on the cluster. A model is composed of 3 main parts:

1. Control code
2. Generators calls. Each Generator call can be seen as the node generation of graph.
3. Graph creation (optional). The graph creation permits to interconnect all the nodes generated by the Generators call and set input and output of the graph.

Here is a simple example of a MNIST NN Autotiler Model that can be found in `$GAP_SDK_HOME/examples/gap9/nn/autotiler/`

```
#include <stdint.h>
#include <stdio.h>
#include "AutoTilerLib.h"
#include "CNN_Generators.h"

void MnistModel(unsigned int L1Memory, unsigned int L2Memory, unsigned int L3Memory,
                unsigned int L3Flash)
{
    KernelOper_T Cop = KOP_CONV_DP;

    SetSymbolDynamics();

    SetUsedFilesNames(0, 2, "CNN_BasicKernels.h", "MnistKernels.h");
    SetGeneratedFilesNames("MnistKernels.c", "MnistKernels.h");
    SetMemoryDeviceInfos(4,
        AT_MEM_L1, L1Memory, "Mnist_L1_Memory", 0, 0,
        AT_MEM_L2, L2Memory, "Mnist_L2_Memory", 0, 0,
        AT_MEM_L3_DEFAULTRAM, L3Memory, "Mnist_L3_Memory", 0, 0,
        AT_MEM_L3_DEFAULTFLASH, L3Flash, "0", "Mnist_L3_Flash_Const.dat", 0
    );
    AT_SetGraphCtrl(AT_GRAPH_TRACE_EXEC, AT_OPT_ON);

    LoadCNNLibrary();
    //Convolutional Layer
    CNN_ConvolutionPoolReLU("Conv5x5ReLUMaxPool2x2_0", 0, 2, 2, 2, 2, 12, 14, 14, 12, 1, 1, 1, 1,
                           1, 32, 28, 28,
                           Cop, 5, 5, 1, 1, 1, 1, 0,
                           KOP_MAXPOOL, 2, 2, 1, 1, 2, 2, 0, KOP_RELU);
    //Convolutional Layer
```

(continues on next page)

(continued from previous page)

```

    CNN_ConvolutionPoolReLU("Conv5x5ReLUMaxPool2x2_1", 0, 2, 2, 2, 2, 12, 14, 14, 12, 1, 1, 1, 1, 1,
    ↵ 32, 64, 12, 12,
        Cop, 5, 5, 1, 1, 1, 1, 0,
        KOP_MAXPOOL, 2, 2, 1, 1, 2, 2, 0, KOP_RELU);
    //Linear Layer
    CNN_LinearReLU("LinearLayerReLU_0", 0, 2, 2, 2, 2, 12, 12, 8, 11, 2, 2, 2, 2, 1024, 10,
        KOP_LINEAR, KOP_NONE);
    //Sofmax
    CNN_SoftMax("SoftMax_0", 0, 2, 2, 15, 15, 1, 1, 10, KOP_SOFTMAX);

#define GRAPH
#ifndef GRAPH
    //Open Graph Creation
    CreateGraph("MnistCNN",
        /* Arguments either passed or globals */
        //Here goes input, output and layers parameters(weights and biases)
        //ConstInfo function takes as input a tensor in CxHxW format and create a single
    ↵ binary file charged in flash
        //It can take in input float and automaticcally convert in fixed point format
        CArgs(8,
            TCArgInfo("short int * __restrict__", "Input0", ARG_SCOPE_ARG, ARG_DIR_IN, AT_
    ↵ MEM_L2, AT_MEM_UNDEF, 0),
            TCArgInfo("short int * __restrict__", "Step1Weights", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step1Weights.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Step1Biases", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step1Biases.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Step2Weights", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step2Weights.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Step2Biases", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step2Biases.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Step3Weights", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step3Weights.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Step3Biases", ARG_SCOPE_GLOBAL, ARG_
    ↵ DIR_CONSTIN, AT_MEM_L3_HFLASH, AT_MEM_UNDEF, ConstInfo("model/Step3Biases.tensor", 1,
    ↵ 1, 16, 0)),
            TCArgInfo("short int * __restrict__", "Output0", ARG_SCOPE_ARG, ARG_DIR_OUT,
    ↵ AT_MEM_UNDEF, AT_MEM_L2, 0)
        ),
        //Graph internal buffer between layers
        /* Locals, allocated dynamically */
        CArgs(3,
            TCArgInfo("short int * __restrict__", "OutputStep2", ARG_SCOPE_LOCAL, ARG_DIR_
    ↵ INOUT, AT_MEM_UNDEF, AT_MEM_UNDEF, 0),
            TCArgInfo("short int * __restrict__", "OutputStep3", ARG_SCOPE_LOCAL, ARG_DIR_
    ↵ INOUT, AT_MEM_UNDEF, AT_MEM_UNDEF, 0),
            TCArgInfo("short int * __restrict__", "OutputStep4", ARG_SCOPE_LOCAL, ARG_DIR_
    ↵ INOUT, AT_MEM_UNDEF, AT_MEM_UNDEF, 0)
    );
#endif

```

(continues on next page)

(continued from previous page)

```

);
//Node Connections with arguments
AddNode("Conv5x5ReLUMaxPool2x2_0", //Name of the Generated Layer
        Bindings(4),           //Number of parameters that generated layers has
        ↵and that needs to be connected
        ↵Filter,
        GNodeArg(GNA_IN, "Input0", 0),
        GNodeArg(GNA_IN, "Step1Weights", 0),
        GNodeArg(GNA_IN, "Step1Biases", 0),      //short int * __restrict__ Bias,
        GNodeArg(GNA_OUT, "OutputStep2", 0)       //short int * __restrict__ Out
        )
);

AddNode("Conv5x5ReLUMaxPool2x2_1", Bindings(4, GNodeArg(GNA_IN, "OutputStep2", 0),
    ↵GNodeArg(GNA_IN, "Step2Weights", 0), GNodeArg(GNA_IN, "Step2Biases", 0), GNodeArg(GNA_
    ↵OUT, "OutputStep3", 0)));
AddNode("LinearLayerReLU_0", Bindings(4, GNodeArg(GNA_IN, "OutputStep3", 0),
    ↵GNodeArg(GNA_IN, "Step3Weights", 0), GNodeArg(GNA_IN, "Step3Biases", 0), GNodeArg(GNA_
    ↵OUT, "OutputStep4", 0)));
AddNode("SoftMax_0", Bindings(2, GNodeArg(GNA_IN, "OutputStep4", 0), GNodeArg(GNA_
    ↵OUT, "Output0", 0)));
//Close Graph creation
CloseGraph();
#endif
}

int main(int argc, char **argv)
{
    if (TilerParseOptions(argc, argv)) {
        printf("Failed to initialize or incorrect output arguments directory.\n");
        ↵return 1;
    }
    MnistModel(48000, 300*1024, 8*1024*1024, 20*1024*1024);
    GenerateTilingCode();
    ↵return 0;
}

```

This code is compiled and linked with the Autotiler library, provided within the GAP SDK and executed on a PC. For examples you can compile and run the previous code with the following commands:

```
gcc -o GenMnist -fcommon -I$(TILER_INC) -I$(TILER_EMU_INC) $(CNN_GEN_INCLUDE) MnistModel.  
→c $(CNN_GEN) $(TILER_LIB)  
../GenMnist
```

The definition of the variables present in the previous code is done sourcing the GAP SDK. This will generate two files: `MnistKernels.c` and `MnistKernels.h`. This code can be found in the examples of the SDK if you want to see it and experiment with it `$GAP_SDK_HOME/examples/gap9/nn/autotiler/MnistGraph/`.

The functions that can be called from the user code are 3: **Graph Constructor**, **Graph Runner**, and **Graph Destructor**. The Graph constructor takes care of all memory allocations (L3,L2 and L1) and buffer initialization (in this document we also refers to *promotion*). The Graph executor executes the Graph with a given input. The Destructor

deallocates all the allocated memories.

```
int MnistCNN(
    short int * __restrict__ Input0,
    short int * __restrict__ Output0,);

int MnistCNN_Construct();

int MnistCNN_Destruct();
```

## AT Model Control code

Autotiler model has few APIs to control the code generation file names and includes, the memories used and the code generation options.

To set the include that are added at beginning of the generated files you can use SetUsedFilesNames API.

```
void SetUsedFilesNames(
    char *StdTypedefsName,           /*< File names where standard typedefs used by
→ basic kernels are declared */
    unsigned int LibKernelFileCount,  /*< Number of header files for basic kernels to
→ be imported into the current model, list of strings follows */
    ...
);
```

To set the name of generated code and header file the SetGeneratedFilesNames API provides two arguments.

```
void SetGeneratedFilesNames(
    char *CallTemplatesName,          /*< A .c file for user kernels C generated code */
    char *CallTemplatesNameHeader    /*< A .h file to export user kernels C
→ generated code */
);
```

The memory devices used by Autotiler to stock constants and dynamic buffers can be set with the SetMemoryDeviceInfos.

```
void SetMemoryDeviceInfos(
    int Count,           /*< Number of Items */
    ...
);
```

The item list is a list of Count (first argument) items where each item is a tuple of 5 elements:

1. AT\_MemLocation\_T which memory device (see list below) you want to configure
2. AvailableMemory int available memory to be used on this device
3. MemmoryBaseName char \* a legal C var name to be used as a base in this memory device
4. ConstFileName char \* a file name to be used if loading constant in this device is needed, flash device only
5. ExtManaged int 0 if device configuration code should be generated and used internally by Autotiler, != 0 if managed externally, in this case the user should take care of properly init, configure and allocate the memory Device

The list of possible memory device to configure is the following:

- AT\_MEM\_L3\_HRAM - L3 Hyper Ram

- AT\_MEM\_L3\_QSPIRAM - L3 QuadSPI Ram
- AT\_MEM\_L3\_OSPIRAM - L3 OctoSPI Ram
- AT\_MEM\_L3\_DEFAULTRAM - L3 Default Ram. In the newest SDK the RAM available is set by the BSP thus the API hides the burdon of selecting which kind of ram you are using
- AT\_MEM\_L3\_HFLASH - L3 Hyper Flash
- AT\_MEM\_L3\_QSPIFLASH - L3 QuadSPI Flash
- AT\_MEM\_L3\_OSPIFLASH - L3 OctoSPI Flash
- AT\_MEM\_L3\_MRAMEFLASH - L3 MRAM Flash
- AT\_MEM\_L3\_DEFAULTFLASH - L3 Default Flash. In the newest SDK the Flash available is set by the BSP thus the API hides the burdon of selecting which kind of flash you are using
- AT\_MEM\_L2 - L2 Fabric Controller memory
- AT\_MEM\_L1 - L1 Cluster Memory

There are then several code generation options that can be set with the AT\_SetGraphCtrl API.

```
void AT_SetGraphCtrl(  
    AT_GraphCtrl_T Ctrl,      /**< Which option */  
    void *Val                /**< Value for this option. Use APT_OPT_ON, AT_OPT_OFF, AT_  
    ↪OPT_VAL(Val) */  
);
```

The list of all options that can be set is the following:

- AT\_KERNEL\_BUFFER\_PROMOTE - When all user kernel arguments can fit into given L1 memory promote them to buffer, default is 1
- AT\_KERNEL\_PARTIAL\_BUFFER\_PROMOTE - When all tile of a user kernel argument across Input Features can fit into given L1 memory promote them to partial buffer, default is 1
- AT\_KERNEL\_NOSOLUTION\_ERROR - Report an error when no tiling solution is found, default is 1
- AT\_GRAPH\_MONITOR\_CYCLES - Enable automatic cycle capture for each node of the graph, default is 0
- AT\_GRAPH\_MONITOR\_CVAR\_NAME - When monitor cycles is on name of the C var array to receive results, default is AT\_GraphPerf
- AT\_GRAPH\_PRODUCE\_NODE\_NAMES - Enable production of an array containing the name of each graph node, default is 0
- AT\_GRAPH\_PRODUCE\_NODE\_CVAR\_NAME - When producing node names is on name of the C array receiving the names as strings, default is ATGraphNodeNames
- AT\_GRAPH\_PRODUCE\_OPERINFOS - Enable production of number of macs for each layer, default is 0
- AT\_GRAPH\_PRODUCE\_OPERINFOS\_CVAR\_NAME - When Number of oper Infos is on name of the C array receiving mac infos for each node, default is AT\_GraphOperInfosNames
- AT\_GRAPH\_REORDER\_CONSTANT\_IN - Enable reordering of constant inputs in order to transform 2D accesses into 1D accesses, default is 1
- AT\_GRAPH\_TRACE\_EXEC - Enable trace of activity, default is 1
- AT\_GRAPH\_NOINLINE\_NODE - If 1 all user kernel function is marked as noinline, default is 0
- AT\_GRAPH\_PREF\_L3\_EXEC - In case a symbol must be allocated in L3 for execution this is the prefered memory, default is AT\_MEM\_L3\_HRAM

- AT\_GRAPH\_CONST\_EXEC\_FROM\_FLASH - If 1, for constant symbol executes from home location, default is 0
- AT\_GRAPH\_PREF\_L3\_HOME - For constant symbols which L3 flash prefered memory, default is AT\_MEM\_L3\_HFLASH
- AT\_GRAPH\_DUMP\_TENSOR - Trace selected tensors arguments at inference time, either all nodes or selected node
- AT\_GRAPH\_DUMP\_ONE\_NODE - Trace one specific graph node
- AT\_GRAPH\_ARG2STRUCT - Kernel C arguments are promoted to struct
- AT\_GRAPH\_SIZE\_OPT - Graph constructor, runner and destructor are compiled with -Os
- AT\_GRAPH\_WARM\_CONSTRUCT - If Warm arg should be added to constructor to bypass all but L1 mem allocation
- AT\_GRAPH\_CHECKSUM - Trace checksum output tensors at inference time
- AT\_GRAPH\_GROUP\_WEIGHTS - Group const inputs of a user kernel when they have identical structure
- AT\_GRAPH\_ASYNC\_FORK - Replace sync fork by async one when feasible (inner call and a single call)
- AT\_GRAPH\_DUMP\_GRAPH\_OUTPUTS - Trace graph output tensors at inference time

## Warm Constructor

The Warm contrctrur permits to handle the allocation of L3, L2 and L1 static and dynamic data. There are 3 types of warm contrctrur, *only L1, L1 and dynamic L2* (the L2 portion which contains only the data used during a graph execution) and *all*, which is L1, L2 dynamic and static and L3. The examples shown below can be found in `$GAP_SDK_HOME/examples/gap9/nn/autotiler/MnistGraphWarm`.

To select the warm constructor use the `AT_SetGraphCtrl` api with the appropriate option:

```
#define WARM_DISABLED 0
#define WARM_L1      1
#define WARM_L2_DYN_L1 2
#define WARM_ALL     3

AT_SetGraphCtrl(AT_GRAPH_WARM_CONSTRUCT, AT_OPT_VAL(WARM_ALL));
```

If warm is equal to 1 the generated contructor and destructor will look like this:

```
int MnistCNN_Construct(int Warm)
{
    if (Warm) {
        Mnist_L1_Memory = (AT_L1_POINTER) AT_L1_ALLOC(0, 41632);
        if (Mnist_L1_Memory == 0) return 4;
        return 0;
    }
    AT_DEFAULTFLASH_FS_FC_EVENT _UchanHF1, *UchanHF1 = &_UchanHF1;
    AT_DEFAULTFLASH_FS_CONF_T DefaultFlashConf;

    int Error;
    AT_DEFAULTFLASH_FS_CONF_INIT(&DefaultFlashConf, AT_MEM_L3_DEFAULTFLASH, 0);
    AT_DEFAULTFLASH_FS_OPEN(&DefaultFlash, &DefaultFlashConf, "Mnist_L3_Flash_Const.dat",
    &Error);
```

(continues on next page)

(continued from previous page)

```

if (Error) return 1;

Mnist_L2_Memory = (AT_L2_POINTER) AT_L2_ALLOC(0, 135956);
if (Mnist_L2_Memory == 0) return 3;
Mnist_L1_Memory = (AT_L1_POINTER) AT_L1_ALLOC(0, 41632);
if (Mnist_L1_Memory == 0) return 4;

...
}

int MnistCNN_Destruct(int Warm)
{
    if (Warm) {
        AT_L1_FREE(0, Mnist_L1_Memory, 41632);
        return 0;
    }
    AT_L2_FREE(0, Mnist_L2_Memory, 135956);
    AT_L1_FREE(0, Mnist_L1_Memory, 41632);
    AT_DEFAULTFLASH_FS_CLOSE(&DefaultFlash);
    return 0;
}

```

As you can see in the snippet above the Warm parameter only controls Dynamic L1 allocation, it can be 0 or >=1. If warm is equal to 1 only the L1 will be allocated in the constructor and deallocated in the destructor. A typical use case is to use the Constructor with the 0 warm parameter to initialize L2 and L3 memories and reserve the allocation (and deallocation) of L1 for later during the execution.

If Warm is equal to 2 (WARM\_L2\_DYN\_L1) the L1 and Dynamic allocated L2 are warm. The generated constructor and destructor will look like this:

```

int MnistCNN_Construct(int Warm)

{
    if (Warm) {
        Mnist_L1_Memory = (AT_L1_POINTER) AT_L1_ALLOC(0, 41632);
        if (Mnist_L1_Memory == 0) return 4;
        if (Warm==2) {
            Mnist_L2_Memory_Dyn = (AT_L2_POINTER) AT_L2_ALLOC(0, 11264);
            if (Mnist_L2_Memory_Dyn == 0) return 3;
        }
        return 0;
    }
    AT_DEFAULTFLASH_FS_FC_EVENT _UchanHF1, *UchanHF1 = &_UchanHF1;
    AT_DEFAULTFLASH_FS_CONF_T DefaultFlashConf;

    int Error;
    AT_DEFAULTFLASH_FS_CONF_INIT(&DefaultFlashConf, AT_MEM_L3_DEFAULTFLASH, 0);
    AT_DEFAULTFLASH_FS_OPEN(&DefaultFlash, &DefaultFlashConf, "Mnist_L3_Flash_Const.dat",
    &Error);
    if (Error) return 1;

    Mnist_L2_Memory = (AT_L2_POINTER) AT_L2_ALLOC(0, 124692);
    if (Mnist_L2_Memory == 0) return 3;
}

```

(continues on next page)

(continued from previous page)

```

Mnist_L2_Memory_Dyn = (AT_L2_POINTER) AT_L2_ALLOC(0, 11264);
if (Mnist_L2_Memory_Dyn == 0) return 3;
Mnist_L1_Memory = (AT_L1_POINTER) AT_L1_ALLOC(0, 41632);
if (Mnist_L1_Memory == 0) return 4;
...
}

int MnistCNN_Destruct(int Warm)

{
    if (Warm) {
        AT_L1_FREE(0, Mnist_L1_Memory, 41632);
        if (Warm==2) {
            AT_L2_FREE(0, Mnist_L2_Memory_Dyn, 11264);
        }
        return 0;
    }
    AT_L2_FREE(0, Mnist_L2_Memory, 124692);
    AT_L2_FREE(0, Mnist_L2_Memory_Dyn, 11264);
    AT_L1_FREE(0, Mnist_L1_Memory, 41632);
    AT_DEFAULTFLASH_FS_CLOSE(&DefaultFlash);
    return 0;
}

```

With relation to the previous one with WARM\_L2\_DYN\_L1 you can also control the dynamic part of the L2 allocated memory. The dynamic allocation L2 is the portion of the used L2 which is only used during Graph execution. Thus can be allocated just before the Graph execution and deallocated right after to be available for other data.

If Warm is equal to 3 (WARM\_ALL) all memories are warm. The generated Constructor and Destructor will look like:

```

int MnistCNN_Construct(int DoL1Alloc, int DoL2Alloc, int DoL2DynAlloc, int DoL3Init, int_
DoL3Alloc, int DoPromotion)

{
    if (DoL3Init) {
        AT_DEFAULTFLASH_FS_CONF_T DefaultFlashConf;

        int Error;
        AT_DEFAULTFLASH_FS_CONF_INIT(&DefaultFlashConf, AT_MEM_L3_DEFAULTFLASH, 0);
        AT_DEFAULTFLASH_FS_OPEN(&DefaultFlash, &DefaultFlashConf, "Mnist_L3_Flash_Const.
dat", &Error);
        if (Error) return 1;
    }

    if (DoL2Alloc) {
        Mnist_L2_Memory = (AT_L2_POINTER) AT_L2_ALLOC(0, 124692);
        if (Mnist_L2_Memory == 0) return 3;
    }
    if (DoL2DynAlloc) {
        Mnist_L2_Memory_Dyn = (AT_L2_POINTER) AT_L2_ALLOC(0, 11264);
        if (Mnist_L2_Memory_Dyn == 0) return 3;
    }
}

```

(continues on next page)

(continued from previous page)

```

if (DoL1Alloc) {
    Mnist_L1_Memory = (AT_L1_POINTER) AT_L1_ALLOC(0, 41632);
    if (Mnist_L1_Memory == 0) return 4;
}
if (DoPromotion) {
    AT_DEFAULTFLASH_FS_FC_EVENT _UchanHF1, *UchanHF1 = &_UchanHF1;
    /* Moving Step1Weights, size 1600 from DefaultFlash at 122880 to (size 1600) L2 at 122880..124479 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 122880), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 122880), 1600, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
    /* Moving Step1Biases, size 64 from DefaultFlash at 124608 to (size 64) L2 at 124608..124671 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 124608), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 124608), 64, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
    /* Moving Step2Weights, size 102400 from DefaultFlash at 0 to (size 102400) L2 at 0..102399 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 0), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 0), 102400, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
    /* Moving Step2Biases, size 128 from DefaultFlash at 124480 to (size 128) L2 at 124480..124607 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 124480), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 124480), 128, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
    /* Moving Step3Weights, size 20480 from DefaultFlash at 102400 to (size 20480) L2 at 102400..122879 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 102400), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 102400), 20480, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
    /* Moving Step3Biases, size 20 from DefaultFlash at 124672 to (size 20) L2 at 124672..124691 */
    AT_DEFAULTFLASH_FS_FC_COPY(&DefaultFlash, ((AT_DEFAULTFLASH_FS_EXT_ADDR_TYPE) 0 + 124672), ((AT_DEFAULTFLASH_FS_INT_ADDR_TYPE) Mnist_L2_Memory + 124672), 20, 0, UchanHF1);
    AT_DEFAULTFLASH_FS_FC_WAIT(&DefaultFlash, UchanHF1);
}
return 0;
}

int MnistCNN_Destruct(int DoL1Dealloc, int DoL2Dealloc, int DoL2DynDealloc, int DoL3Dealloc, int DoL3DeInit)

{
    if (DoL2Dealloc) {
        AT_L2_FREE(0, Mnist_L2_Memory, 124692);
    }
    if (DoL2DynDealloc) {

```

(continues on next page)

(continued from previous page)

```

        AT_L2_FREE(0, Mnist_L2_Memory_Dyn, 11264);
    }
    if (DoL1Dealloc) {
        AT_L1_FREE(0, Mnist_L1_Memory, 41632);
    }
    if (DoL3DeInit) {
        AT_DEFAULTFLASH_FS_CLOSE(&DefaultFlash);
    }
    return 0;
}

```

As you can see L1, dynamic L2, static L2, L3 and also L3/L2 initialization (promotion) can be fully controllable from user code. **Thus you need to be carefull to memory fragmentation and proper allocation of all buffer and promotion before the execution of the Graph.** While in the other modes the Graph runner is always equal to the version with Warm option desactivated, with WARM\_ALL you can also handle the L1. In the runner you can optionally pass a pointer to L1 if you want to handle it in your user code (this is always the case in reentrant mode, see next paragraph), otherwise you can pass NULL and will handled by the constructor / runner / destructor.

```

int MnistCNN(
    short int * __restrict__ Input0,
    short int * __restrict__ Output0,
    void * _L1_Memory)

{
    if (_L1_Memory) Mnist_L1_Memory = (AT_L1_POINTER) _L1_Memory;
    ...
    [Call to all nodes]
    ...
    if (_L1_Memory) Mnist_L1_Memory = 0;
    return 0;
}

```

## Reentrant (preemptible) Graph Executor

**Warning:** This section will be soon completed

## AT Model Generators Calls

**Warning:** This section will be soon completed

## AT Model Graph dependecies

**Warning:** This section will be soon completed

### 6.2.3 Create your own generators aka the complete auto-tiler guide

**Warning:** This documentation may be out-of-date.

If you want to extend the generators suite that we provide in our SDK in this paragraph we explain all the auto-tiler fundamentals.

#### GAP's memory hierarchy

GAP's memory hierarchy is made up of three levels:

1. **Shared level 1 memory** Internal and tightly coupled with GAP cluster, it can deliver up to 8 parallel memory accesses in a single cycle. This is by far GAP's fastest memory and has the highest bandwidth. As it is quite costly, its size has had to be kept relatively small, 64 Kbytes in the current configuration.
2. **Level 2 memory** Internal memory significantly larger than level 1, but with higher access latency (approximately 6 cycles) and lower bandwidth. Its primary role is to store programs that are then fetched by the different instruction caches attached to GAP's various cores and to store relatively large data structures. In the current version its size is 512 Kbytes.
3. **Level 3 memory** External and optional (in the case of RAM). It is either read only (Flash) or read/write (RAM). Read only memory is mapped onto either the quad-SPI or HyperBus interfaces. Read/write memory is mapped onto the HyperBus interface. The latency, the access time and the bandwidth is even more limited than the other 2 memory areas and importantly accesses to level 3 memory consume more energy.

There are two DMA units. The micro-DMA unit, responsible for transfers to and from peripherals into the level 2 memory and the cluster-DMA unit, which can be used to schedule unattended transfers between level 2 and level 1 memory.

The level 1 and level 2 memories are also directly accessible by all the cores in the chip.

To keep the size of the chip as small as possible and to reduce the amount of energy spent in memory accesses GAP does not use data caching. Level 3 memory is the most constrained since data must be moved into the chip level 2 memory with the micro-DMA unit (streaming).

#### Auto-tiler architecture

The ideal memory model for a developer is to view memory as one large continuous area that is as big and as fast as possible. This is normally achieved by a data cache which automatically moves data between memory areas. Since GAP does not implement data caching and since GAP's cluster is optimized for processing data in a linear or piece-wise linear fashion, we provide a software tool, the GAP auto-tiler, to help the developer by automating memory movements for programs of this type.

The auto-tiler uses defined patterns of data access to anticipate data movements ensuring that the right data is available in level 1 memory when needed. Since GAP's cluster-DMA and micro-DMA units operate in parallel with all the GAP cores, the auto-tiler can use these units to make these pipelined memory transfers quasi-invisible from a performance point of view. The auto-tiler decomposes 1, 2, 3 and 4-dimensional data structures into a set of tiles (2-dimensional structures) and generates the C code necessary to move these tiles in and out of shared level 1 memory. The developer concentrates on the code that handles simple 2D tiles and the auto-tiler takes care of moving tiles into and out of level 1 memory as necessary and calling the developer's code.

Below is a list of the entities that make up the configuration or data model necessary for the GAP auto-tiler to generate functioning code. We refer extensively to a 'model' which is used to indicate the use of the auto-tiler API to declare the signature of developer functions (basic kernels) and define iterated assemblies of basic kernels which actually cause the auto-tiler to generate code.

**1. Basic kernels** Pure C functions, these are written by the developer as if all the data structures they access can fit into shared level 1 memory (data tiles). Basic kernels can also use arguments that are prepositioned in memory (not tiled). Basic kernel functions are modeled, their call template formally described, by basic kernel models. This allows the GAP auto-tiler to generate code that calls them. They are described in detail in the section [Basic kernels](#).

**2. User kernels** User kernel models group calls to basic kernels and allow the GAP auto-tiler to generate a C function for that grouping. A user kernel defines the different, predefined ways in which of 2, 3 or 4-dimensional data is traversed, and one or more basic kernels are called. A user kernel model takes arguments that describe the input, working and output data that needs to be modeled.

User kernels consume and process data through these user kernel arguments. Kernel argument models describe argument location in the memory hierarchy (level 3, level 2 or level 1), direction (in, out, in out, pure buffer), inner dimension (width and height), dimensionality (1D, 2D, 3D, 4D). Kernel argument models include several other attributes that are used to constrain the tiles that are generated from the argument (preferred size, odd/even size, etc.) or to provide hints that control the double or triple-buffering strategy used in the generated code. Calls to basic kernels can be inserted in different places in the generated iterative code (inner loop, loop prologue, loop epilogue, etc.). The calls are bound to arguments which can either be from the kernel argument model described above, direct C arguments or immediates. User kernels are described in detail in the section [User kernels](#).

**3. User kernel groups** User kernel groups are models that combines or groups several user kernels together in a given order. A C function is generated from the user kernel group whose body contains calls to the sequence of user kernels with proper argument bindings between them. User kernel groups are described in detail in the section [User kernel groups](#).

**4. Model control** Model control contains configuration elements such as available memory, compilation hints, consumed and produced files, basic kernels loaded as libraries and an ordered list of user kernels and/or user kernel groups. Model control is described in the section [Controlling tiled code generation](#).

The auto-tiler model is created through a series of calls to functions from the auto-tiler library. In addition to these calls, the developer can add whatever application specific code needed. Compiling and running the model on the build system creates a set of C source files that are then compiled and run on GAP.

The basic object on which the GAP auto-tiler works is a 2D space that we call a data plane. Each user kernel argument corresponds to a data plane and potentially each user kernel argument can have a different width and height. For example, if the kernel we want to write produces one output for each 2x2 input sub region the input argument will be a data plane of WxH in size and the output argument will be a data plane of size (W/2)x(H/2).

This basic data plane can then be extended to 3 or 4 dimensions. Extending the dimension of a data plane is simply a repletion of the 2-dimensional basic data plane.

The GAP auto-tiler splits basic data planes into tiles, the number of tiles for each user kernel argument is identical but their dimension can vary from one argument to the other.

The GAP auto-tiler makes the following hypothesis about the user algorithm:

```
OutputDataPlane = Kernel(InputDataPlane1, InputDataPlane2, ...)
```

Which can be rewritten as:

```
For i in [0..NumberOfTiles-1]
  Tile(OutputDataPlane, i) =
    Kernel(Tile(InputDataPlane1, i), Tile(InputDataPlane2, i), ...)
```

Not all algorithms fits into this template but we believe it captures a large family of useful algorithms.

The illustrative examples below show how an entire auto-tiler model is constructed. Don't worry if they are confusing at the start. As you read the other sections of the manual the examples should become clear.

### Illustrative example 1 - Matrix addition

In this example, we want to add two integer matrices and store the result in a third matrix.

You can see the full code for the example in `examples/autotiler/MatrixAdd`.

The basic kernel that does the job of addition is `MatSumPar`. It takes arguments of pointers to two input tiles and one output tile, these three tiles are expected to have the same dimensions which are passed as `W` and `H`. It is expected that the 3 matrices fit into shared level 1 memory.

The basic kernel for this example is shown in the basic kernel section below.

We first model the template of basic kernel `MatSumPar` function call.

```
LibKernel("MatSumPar", CALL_PARALLEL,
    CArgs(5,
        TCArg("Word32 * __restrict__ ", "In1"),
        TCArg("Word32 * __restrict__ ", "In2"),
        TCArg("Word32 * __restrict__ ", "Out"),
        TCArg("unsigned int", "W"),
        TCArg("unsigned int", "H")
    ),
    "MatrixAdd_Arg_T"
);
```

And then we model a user kernel generator with no restrictions on the matrices dimensions (of course they need to fit into the level 2 memory). This describes the input and output parameters of the generated function and the way that the data is iterated.

```
void MatAddGenerator(char *UserKernelName, int W, int H)
```

During our build process the generator code is compiled and `MatAddGenerator("MatAdd", 200, 300)` is called. The GAP auto-tiler generates the following code:

```
void MatAdd(
    Word32 * __restrict__ In1,
    Word32 * __restrict__ In2,
    Word32 * __restrict__ Out,
    Kernel_T *Ker)

{
    /* Local variables used by this kernel */
    int DmaR_Evt1;
    int DmaR_Evt2;
    int DmaW_Evt1;
    int Iter, Last, NextLast, NextNextLast, InPlane, OutPlane=0;
    int N_Ti = 0, N_TiIp = 0;
    MatrixAdd_Arg_T S_KerArg0, *KerArg0 = &S_KerArg0;

    /* Initialize KerArg, Kernel invariant arguments */
    KerArg0->W = (unsigned int) (200);
    KerArg0->H = (unsigned int) (10);
    /* =====Read First Tile===== */
    /* Initial reads in L2, O_DB or O_BUFF */
    DmaR_Evt1 = gap8_dma_memcpy((unsigned int) In1+(0),
```

(continues on next page)

(continued from previous page)

```

    (unsigned int) (L1_Memory + 0)+0, 8000, DMA_COPY_IN);
DmaR_Evt2 = gap8_dma_memcpy((unsigned int) In2+(0),
    (unsigned int) (L1_Memory + 16000)+0, 8000, DMA_COPY_IN);
/* =====End Read First Tile===== */
/* Kernel Iteration Loop on tiled inner space */
for (Iter=0; Iter<30; Iter++) {
    /* Loop Iteration Body on tiled inner space */
    /* Elaborate Last, Next_Last, Next_Next_Last */
    Last = ((Iter+1) == 30);
    NextLast = ((Iter+2) == 30);
    NextNextLast = ((Iter+3) == 30);
    /* =====Read Next Tile===== */
    gap8_dma_wait(DmaR_Evt1);
    gap8_dma_wait(DmaR_Evt2);
    if (!Last) {
        DmaR_Evt1 = gap8_dma_memcpy((unsigned int) In1 + ((Iter+1)*8000),
            (unsigned int) (L1_Memory + 0) + 8000*((N_Ti+1) % 2), 8000, DMA_COPY_IN);
        DmaR_Evt2 = gap8_dma_memcpy((unsigned int) In2 + ((Iter+1)*8000),
            (unsigned int) (L1_Memory + 16000) + 8000*((N_Ti+1) % 2), 8000, DMA_COPY_IN);
    }
    /* =====End Read Next Tile===== */
    /* Call Kernel LOC_INNER_LOOP */
    KerArg0->In1 = (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 0) + 8000*(N_Ti % 2));
    KerArg0->In2 = (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 16000) + 8000*(N_Ti % 2));
    KerArg0->Out = (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 32000) + 8000*(N_Ti % 2));
    gap8_task_dispatch((1<<gap8_ncore())-1, MatrixAdd, (unsigned int) KerArg0);
    MatrixAdd(KerArg0);
    /* =====Write Tile===== */
    if (Iter) {
        gap8_dma_wait(DmaW_Evt1);
    }
    DmaW_Evt1 = gap8_dma_memcpy((unsigned int) Out + ((Iter)*8000),
        (unsigned int) (L1_Memory + 32000) + 8000*(N_Ti % 2), 8000, DMA_COPY_OUT);
    /* =====End Write Tile===== */
    N_Ti++;
    /* End Kernel Iteration Loop on tiled inner space */
}
Iter=30;
/* =====Write Last Tile===== */
gap8_dma_wait(DmaW_Evt1);
/* =====End Write Last Tile===== */
}

```

## Illustrative example 2 - Searching the maximum in a 2D matrix

This user kernel also operates on a 2D integer matrix but returns the maximum value in this matrix. Here we model a classic parallel map/reduce algorithm to accomplish the task. We use a basic kernel, `KerMatrixMax`, which operates on a tile. `KerMatrixMax` takes as arguments: a pointer to a tile `In`, the dimensions of this tile (`W` and `H`), a pointer to the vector `Out` (a buffer whose dimension is the number of tiles) to store the max for each tile. `KerMatrixMax` also takes an argument `CompareWithOut`, which indicates if a valid maximum is already available in the output vector and should be compared against the maximum calculated in the current tile.

Each basic kernel call will return the maximum in a sub section of the full input matrix. To get the final result we have to reduce the set of sub-results into a single result. This is our second kernel: `KerMatrixMaxReduction`. As a first argument `In` it takes the vector of maximums produced by the first basic kernel as well as its dimension `Ntile`. It produces a pointer to a single maximum in `Out` (a  $1 \times 1$  tile). It should be executed when we are done with all the tiles so the call is inserted in the inner space iterator prologue (see the user model below).

The 2 basic kernels are first modeled.

```
LibKernel("KerMatrixMax", CALL_PARALLEL,           // A parallel call
  CArgs(6,
    // A tile
    TCArg("Word32 * __restrict__", "In"),
    // A vector of maximums
    TCArg("Word32 * __restrict__", "Out"),
    TCArg("unsigned int", "W"),      // Tile width
    TCArg("unsigned int", "H"),      // Tile Height
    // Which Tile - index into Out for result
    TCArg("unsigned int", "TileIndex"),
    // Out[TileIndex] contains a Max
    TCArg("unsigned int", "CompareWithOut")
  ),
  "MatrixMax_Arg_T"
);
LibKernel("KerMatrixMaxReduction", CALL_SEQUENTIAL, // A sequential call
  CArgs(3,
    TCArg("Word32 * __restrict__", "In"),    // A vector of Maximums
    TCArg("Word32 * __restrict__", "Out"),    // Pointer to the unique Maximum
    TCArg("unsigned int", "Ntile")          // Number of entries into In
  ),
  "MatrixMaxReduction_Arg_T"
);
```

Then the Matrix Max user kernel generator:

```
/* A user kernel generator computing the max of a 2D plain matrix of size WxH */
void MatrixMax(char *Name, unsigned int W, unsigned int H)
{
  UserKernel(Name,
    // Dimension is WxH
    KernelDimensions(1, W, H, 1),
    // 2D iteration space
    KernelIterationOrder(KER_DIM2, KER_TILE),
    // Tile horizontally
    TILE_HOR,
```

(continues on next page)

(continued from previous page)

```

// User kernel C template
CArgs(2,
    TCArg("int * __restrict__", "In"),
    TCArg("int * __restrict__", "Out")
),
// 2 calls to basic kernels
Calls(2,
    // First KerMatrixMax in the inner iterator
    Call("KerMatrixMax", LOC_INNER_LOOP,
        Bindings(6,
            K_Arg("In", KER_ARG_TILE), K_Arg("TiledOut", KER_ARG_TILE),
            K_Arg("In", KER_ARG_TILE_W), K_Arg("In", KER_ARG_TILE_H),
            K_Arg("In", KER_ARG_TILEINDEX), Imm(0))
        ),
    // Second KerMatrixMaxReduction after we have consumed all tiles.
    // Final result goes directly to *Out thanks to C_Arg("Out")
    Call("KerMatrixMaxReduction", LOC_INNER_LOOP_EPILOG,
        Bindings(3,
            K_Arg("TiledOut", KER_ARG), C_Arg("Out"),
            K_Arg("TiledOut", KER_ARG_NTILES))
        )
    ),
// 2 User kernel arguments
KerArgs(2,
    // A double buffered input taken from level 2 memory
    // bound to C user kernel In
    KerArg("In", OBJ_IN_DB, W, H,
        sizeof(int), 0, 0, 0, "In", 0),
    // A dynamic buffer declared as W=1 and height=H but H will
    // be reduced to the number of used tiles
    KerArg("TiledOut", OBJ_BUFFER_DYN,
        1, H, sizeof(int), 0, 0, 0, 0, 0)
    )
);
}

```

And here is the code that is generated by the auto-tiler after a call to `MatrixMax("MatMax", 200, 300)`.

```

void MatMax(
    int * __restrict__ In,
    int * __restrict__ Out,
    Kernel_T *Ker)

{
    /* Local variables used by this kernel */
    int DmaR_Evt1;
    int Iter, Last, NextLast, NextNextLast, InPlane, OutPlane=0;
    int N_Ti = 0, N_TiIp = 0;
    MatrixMax_Arg_T S_KerArg0, *KerArg0 = &S_KerArg0;

    /* Initialize KerArg, Kernel invariant arguments */
    KerArg0->W = (unsigned int) (200);

```

(continues on next page)

(continued from previous page)

```

KerArg0->CompareWithOut = (unsigned int) (0);
/* =====Read First Tile===== */
/* Initial reads in L2, O_DB or O_BUFF */
DmaR_Evt1 = gap8_dma_memcpy((unsigned int) In+(0),
    (unsigned int) (L1_Memory + 0)+0, 24800, DMA_COPY_IN);
/* =====End Read First Tile===== */
/* Kernel Iteration Loop on tiled inner space */
for (Iter=0; Iter<10; Iter++) {
    /* Loop Iteration Body on tiled inner space */
    /* Elaborate Last, Next_Last, Next_Next_Last */
    Last = ((Iter+1) == 10);
    NextLast = ((Iter+2) == 10);
    NextNextLast = ((Iter+3) == 10);
    /* =====Read Next Tile===== */
    gap8_dma_wait(DmaR_Evt1);
    if (!Last) {
        DmaR_Evt1 = gap8_dma_memcpy(
            (unsigned int) In + ((Iter+1)*24800),
            (unsigned int) (L1_Memory + 0) + 24800*((N_Ti+1) % 2),
            NextLast?16800:24800, DMA_COPY_IN);
    }
    /* =====End Read Next Tile===== */
    /* Call Kernel LOC_INNER_LOOP */
    KerArg0->In = (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 0) + 24800*(N_Ti % 2));
    KerArg0->Out = (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 49600) + 0 + (Iter)*4);
    KerArg0->H = (unsigned int) (Last?21:31);
    KerArg0->TileIndex = (unsigned int) Iter;
    gap8_task_dispatch((1<<gap8_ncore())-1, KerMatrixMax,
        (unsigned int) KerArg0);
    KerMatrixMax(KerArg0);
    N_Ti++;
    /* End Kernel Iteration Loop on tiled inner space */
}
Iter=10;
/* Call Kernel LOC_INNER_LOOP_EPILOG */
KerMatrixMaxReduction(
    (Word32 * __restrict__)
        ((unsigned int) (L1_Memory + 49600) + 0),
    (Out),
    (unsigned int)10
);
}
}

```

## Basic kernels

Basic kernels are written assuming all their data can fit into the shared level 1 memory. Usually a kernel function will access a data chunk through a pointer argument and will be informed about the data chunk characteristics by means of dimension arguments. A basic kernel manipulates a tile: one access pointer, one width argument and one height argument (a tile can also be of dimension 1). Scalar arguments, shared level 1 preloaded arguments, arguments accessed directly in level 2 can also be used by a basic kernel.

Basic kernels can be either sequential or parallel. A sequential kernel will run on a single core (core 0 of the cluster). For example, a sequential kernel can handle the configuration of the HWCE convolutional accelerator. A parallel kernel is meant to be run on all the available cores of the cluster. When it is called it is dispatched on all the active cores. Writing an optimized kernel usually involves dealing with vectorization to get as much as performance as possible from a single core and then dealing with parallelism to use as many cores in the cluster as possible.

Basic kernels define the functions manipulated by the GAP tile generator. Their interfaces (C data type template) as well as their calling nature (parallel versus sequential) are modeled.

The functions that the basic kernel models describe are called by the auto-tiler at runtime so should be in functions in source external to the model.

The following API is used to add a basic kernel model.

```
void LibKernel(
    // A string. The name of the kernel
    char *KernelName,
    // CALL_SEQUENTIAL when called only by cluster's core 0
    // CALL_PARALLEL when dispatched on all available cluster cores
    KernelCallTypeT CallType,
    // List of C arguments <Type, Name> where Type and Name are strings.
    // Provided by CArgs(ArgCount, List of CArgs)
    CKernel_Arg_T **Cargs,
    // C typedef name, used when CallType is parallel since
    // in this case arguments have to be promoted to a C structure
    char *ParArgTypeName
);
```

### Example 1 - Parallel basic kernel

In this first example we show how to write a basic kernel performing a parallel addition between two 2D integer matrices with a fixed, bias offset.

The parallel addition will look like:

```
typedef struct {
    Word32 * __restrict__ In1;
    Word32 * __restrict__ In2;
    Word32 * __restrict__ Out;
    Word32 Bias;
    Wordu32 W;
    Wordu32 H;
} MatrixAdd_Arg_T;

void MatrixAdd(MatrixAdd_Arg_T *Arg)
```

(continues on next page)

(continued from previous page)

```
{
    Word32 * __restrict__ In1 = Arg->In1;    // First input int matrix
    Word32 * __restrict__ In2 = Arg->In2;    // Second input int matrix
    Word32 * __restrict__ Out = Arg->Out;    // output int matrix
    // Pointer to a bias to be added to each matrix element sum
    Word32 * __restrict__ Bias = Arg->Bias;
    Wordu32 W = Arg->W;          // Width of the working space
    Wordu32 H = Arg->H;          // Height of the working space
    Wordu32 CoreId = gap8_coreid(); // Who am I?
    // The size of the working space is W*H, divide it by number of cores
    // (a ChunkCell)
    Wordu32 ChunkCell = ChunkSize(W*H);
    // Given who we are this is the first chunk in the working space
    // we are interested in
    Wordu32 First = CoreId*ChunkCell;
    // Given First chunk, last chunk is either full sized or capped
    // to working space size
    Wordu32 Last = Min(First+ChunkCell, W*H);
    int i;
    for (i=First; i<Last; i++)
        Out[i] = In1[i] + In2[i] + Bias;
    // Wait on barrier until all the cores have got to here
    rt_team_barrier();
}
```

And this is how the function is modeled as a basic kernel:

```
LibKernel("MatrixAdd", CALL_PARALLEL,
CArgs(6,
    TCArg("Word32 * __restrict__", "In1"),
    TCArg("Word32 * __restrict__", "In2"),
    TCArg("Word32 * __restrict__", "Out"),
    TCArg("Word32", "Bias"),
    TCArg("Wordu32", "W"),
    TCArg("Wordu32", "H")
),
"MatrixAdd_Arg_T"
);
```

## Example 2 - Sequential basic kernel

The second example shows how to model a sequential function to switch on the HWCE accelerator.

In this case, this is a simple sequential call to a preexisting library function. The C function to switch on the HWCE is in the GAP run-time and is called `HWCE_Enable()`.

This is the way this call is modeled.

```
LibKernel("HWCE_Enable", CALL_SEQUENTIAL, CArgs(0), "");
```

## User kernels

### Iteration dimension, iteration space, iteration order, tiling direction

The iteration space of a user kernel can be of dimension 1, 2, 3, or 4.

The inner level of the iteration space is assumed to be 2D (with 1D as a special case where one of the inner dimensions is set to 1). The inner level is the one that will be tiled by GAP auto-tiler. In this document we refer to this inner level as a plane, either input or output.

- A 2D input has width W and height H.
- A 3D input is a collection of Nip 2D structures, so it's dimension is [Nip x W x H], where Nip stands for number of input planes.
- Similarly, a 3D output is a collection of Nop 2D structures, with dimension [W x H x Nop], where Nop stands for number of output planes.
- A 4D input is a collection of Nip x Nop 2D structures, with dimension [Nip x W x H x Nop]

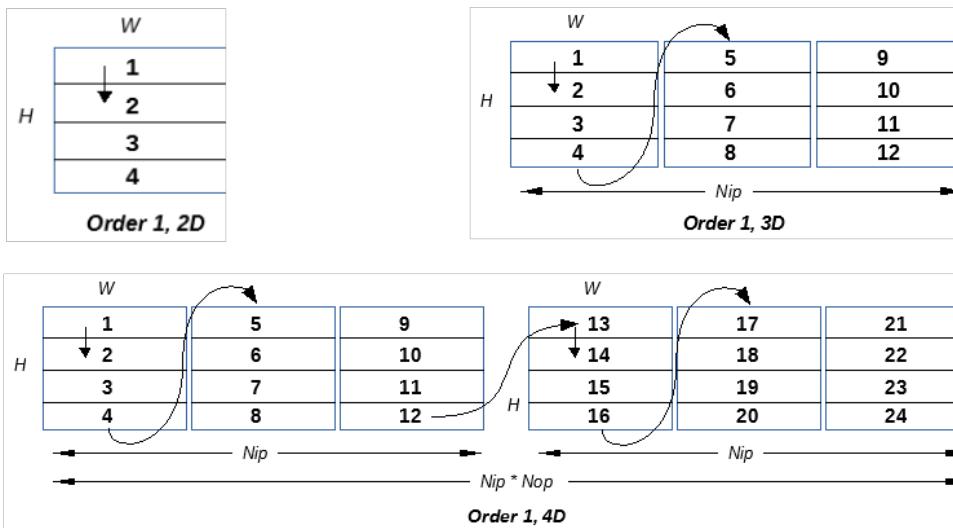
**\*Note: A 2D input or output can be embedded into an iteration space whose dimension is greater than 2. In this case each 2D plane is consumed several times.\***

Two different iteration orders are supported when dimension is greater or equal than 3:

#### Order 1

```
for (Op=0; Op<Nop; Op++) {
    for (Ip=0; Ip<Nip; Ip++) {
        for (Tile=0; Tile<LastTile; Tile++) {
            Foo(DataTile[Ip, Tile, Op]);
        }
    }
}
```

The diagrams Below illustrate how tiles are traversed as a function of the dimension of the iteration space in Order 1.



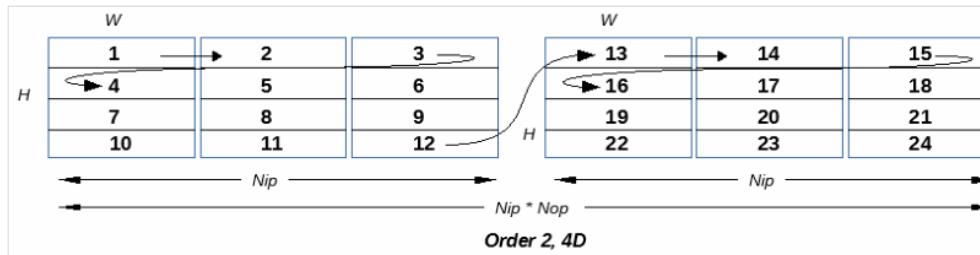
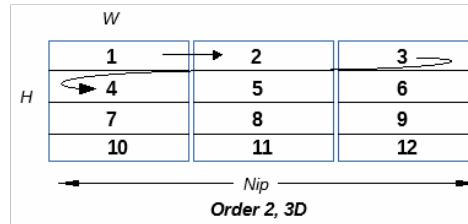
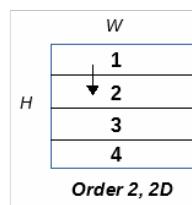
**Order 2**

```

for (Op=0; Op<Nop; Op++) {
    for (Tile=0; Tile<LastTile; Tile++) {
        for (Ip=0; Ip<Nip; Ip++) {
            Foo(DataTile[Ip, Tile, Op]);
        }
    }
}

```

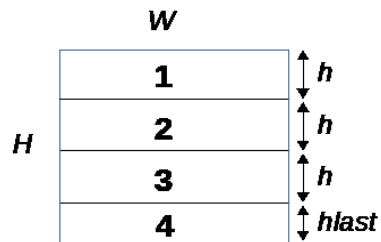
The diagrams Below illustrate how tiles are traversed as a function of the dimension of the iteration space in Order 2.



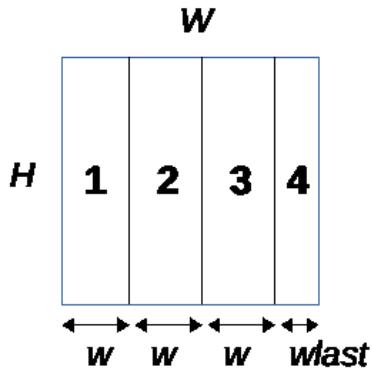
In the current implementation only order 2 is fully supported.

Tiles have 2 possible orientations:

**\*Horizontal:** The  $[W \times H]$  data plane is divided into  $N$  tiles of size  $[W \times h]$  and one last tile of size  $[W \times h_{last}]$  where  $h_{last} < h$



**\*Vertical:** The  $[W \times H]$  data plane is divided into  $N$  tiles of size  $[w \times H]$  and one last tile of size  $[w_{last} \times H]$  where  $w_{last} < w$



It is important to note that one of the 2 dimensions is left untouched so a single line or a single column must fit into the memory constraints given to GAP auto-tiler.

Deciding which orientation to choose is driven by the nature of the algorithm. For example, a function computing a bi-dimensional FFT on a 2D input plane of size  $[S \times S]$  will execute in two passes. A first pass where a 1D FFT is applied on each line so the natural choice is horizontal. Then the second pass will apply a 1D FFT on each column of the 2D plane produced by the first pass, so vertical is the natural choice.

In the current implementation the orientation choice applies to all user kernel arguments. In future version this constraint will be removed to allow the developer to decide a different orientation for each kernel argument.

## User kernel fields

A user kernel is a collection of fields. The following library function is used to create a user kernel:

```
Kernel_T *UserKernel(
    char *TemplateName,
    KernelDimensionT *KerDim,
    KernelIterationT *KerIter,
    Tile_Orientation_T Orientation,
    CKernel_Arg_T **CArg,
    CKernelCall_T **CCalls,
    Object_T **KerArg);
```

The following sections describe the content of each of the user kernel fields.

### TemplateName - kernel name

A string with the kernel name which must be unique.

```
UserKernel("MyFavoriteKernel",
    ...
);
```

## KerDim - kernel dimensions

Specifies the number of input planes (`Nip`), number of output planes (`Nop`), default width (`W`) and height (`H`) of a plane for the user kernel.

Note that `Nip` and `Nop` are shared by all kernel arguments while `W` and `H` can be redefined for each kernel argument.

The following library function is provided to capture the user kernel dimension info:

```
KernelDimensionT *KernelDimensions(  
    unsigned int InPlanes,  
    unsigned int W,  
    unsigned int H,  
    unsigned int OutPlanes);
```

For example, `MyFavoriteKernel` below has 4 input planes, 4 output planes and a default data plane of 32 column and 28 lines:

```
UserKernel("MyFavoriteKernel",  
    KernelDimensions(4, 32, 28, 4),  
    ...  
) ;
```

## KerIter - kernel iteration order

The iteration order captures the overall structure of the user kernel. First the number of dimensions and then the way the iteration is traversed.

Dimensions: `KER_DIM2`, `KER_DIM3`, `KER_DIM4`

### \*Iteration Order 1\*

`KER_DIM2`, `KER_TILE`

: 2D. Tiled inner data plane

`KER_DIM3`, `KER_IN_PLANE`, `KER_TILE`

: 3D. For each `Nip` in data planes all tiles in a plane, `Nop` is treated as equal to 1

`KER_DIM3`, `KER_OUT_PLANE`, `KER_TILE`

: 3D. For each `Nop` out data planes all tiles in a single input plane, `Nip` is treated as equal to 1

`KER_DIM4`, `KER_OUT_PLANE`, `KER_IN_PLANE`, `KER_TILE`

: 4D. For each `Nop` out data planes, for each `Nip` input data planes, for all tile in plane

### \*Iteration Order 2\*

`KER_DIM2`, `KER_TILE`

: 2D. Tiled inner data plane

`KER_DIM3`, `KER_TILE`, `KER_IN_PLANE`

: 3D. For each tile of each `Nip` input planes, `Nop` is treated as equal to 1

`KER_DIM3`, `KER_OUT_PLANE`, `KER_TILE`

: 3D. For each `Nop` out data planes all tiles in a single input plane, `Nip` is treated as equal to 1

KER\_DIM4, KER\_OUT\_PLANE, KER\_TILE, KER\_IN\_PLANE

: 4D. For each Nop out data planes, for each tile in each Nip input data planes

This is the general iteration pattern for a user kernel, then each kernel argument can traverse the whole iteration space or only a subset. For example, a 2D kernel argument when embedded in a 4D user kernel will iterate  $Nip * Nop$  times on the same WxH data plane.

Note: currently only Iteration Order 2 is fully supported.

For example, MyFavoriteKernel below follows iteration Order 2: 4 dimensions, first out-planes then tiles then in-planes:

```
UserKernel("MyFavoriteKernel",
    KernelDimensions(4, 32, 28, 4),
    KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
    ...
);
```

### Orientation - Tiling orientation

In a 2D data plane of dimension  $W \times H$ , tiling can be performed horizontally or vertically. Currently this is a user kernel level parameter and all user kernel arguments are tiled in the same direction. In the future global orientation will be able to be overridden on a per argument basis.

When a 2D data plane is tiled, all the tiles except the last one will have the same size. This size is computed to gain maximum benefit from the configured memory budget.

Orientation can be: TILE\_HOR or TILE\_VER.

For example MyFavoriteKernel below will tile the data plane horizontally.

```
UserKernel("MyFavoriteKernel",
    KernelDimensions(4, 32, 28, 4),
    KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
    TILE_HOR,
    ...
);
```

### CArg - User kernel function template

A user kernel will end up as a C function after it has been processed by the auto tiler. For this reason, the C template of this function must be provided. This is like the template provided for a basic kernel, i.e. a list of C variable names and their associated C types.

The CArgs library function is used to do this. It takes two parameters. Firstly, the number of C arguments and secondly a list of parameters modeled as <Type Name, Arg Name> pairs, the TCArg function is used to create the pair.

```
CKernel_Arg_T **CArgs(
    unsigned int ArgCount,
    ...
);

CKernel_Arg_T *TCArg(
    char *ArgType,
    char *ArgName);
```

For example MyFavoriteKernel below has 4 C arguments: In1, In2, Out and Bias with their respective C types.

```
UserKernel("MyFavoriteKernel",
    KernelDimensions(4, 32, 28, 4),
    KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
    TILE_HOR,
    CArgs(4,
        TCArg("Word32 * __restrict__", "In1"),
        TCArg("Word32 * __restrict__", "In2"),
        TCArg("Word32 * __restrict__", "Out")
        TCArg("Word32 * __restrict__", "Bias")
    ),
    ...
);
```

You will note here that the dimension of the arguments is not passed, they will be captured in the kernel argument description part of the model if they are candidates for tiling. In case that they are pure C arguments, not candidates for tiling, then their dimensions should be passed.

## CCalls - Basic kernels call sequence, call position and arguments

The CCalls field indicates the link between the User Kernel and one or more Basic Kernels. It models the sequence, call position and argument bindings for Basic Kernels called from this user kernel.

### Call sequence

The location of calls to basic kernels in the user kernel iteration sequence depends on the iteration order, Order 1 or Order 2.

Here are the locations where calls can be inserted as a function of iteration order:

```
LOC_INNER_LOOP
LOC_INNER_LOOP_PROLOG
LOC_INNER_LOOP_EPILOG
LOC_IN_PLANE_PROLOG
LOC_IN_PLANE_EPILOG
LOC_PROLOG
LOC_EPILOG
```

The code fragments below show where the calls are inserted in relation to the two possible iteration orders:

#### \*Iteration Order 1\*

```
<LOC_PROLOG>


```

(continues on next page)

(continued from previous page)

```

    }
    <LOC_IN_PLANE_EPILOG>
}
<LOC_EPILOG>
```

### \*Iteration Order 2\*

```

<LOC_PROLOG>
for (Op=0; Op<Nop; Op++) {
    <LOC_INNER_LOOP_PROLOG>
    for (Tile=0; Tile<LastTile; Tile++) {
        <LOC_INNER_PLANE_EPILOG>
        for (Ip=0; Ip<Nip; Ip++) {
            <LOC_INNER_LOOP>
            Foo(DataTile[Ip, Tile, Op]);
        }
        <LOC_INNER_PLANE_EPILOG>
    }
    <LOC_INNER_LOOP_EPILOG>
}
<LOC_EPILOG>
```

## Call order

At each insertion point, calls are inserted in the order they appear in the user kernel call sequence.

User kernel calls are captured by the following library call, the number of calls in the user kernel and then a list of basic kernels calls:

```
CKernelCall_T **Calls(
    unsigned int CallCount,
    ...
);
```

Each call is captured by:

```
CKernelCall_T *Call(
    char *CallName,
    KernelCallLocationT CallLocation,
    ArgBindingDescr_T **BindingList
);
```

Where **CallName** is a basic kernel name that must exist, **CallLocation** is a call location in the iteration template and **BindingList** is a list of bindings between basic kernel C formal arguments and different entities:

```
ArgBindingDescr_T **Bindings(
    int BCount,
    ...
);
```

## Call bindings

Each argument for each call can be bound to combination of user kernel arguments (tiles or attribute of tiles such as tile width and tile height), plain C arguments or immediate values.

The possible binding sources are listed below.

### User kernel arguments - tile

`K_Arg(UserKernelArgName, KER_ARG_TILE)`

: Pointer to a tile

### User kernel arguments - entire data plane

`K_Arg(UserKernelArgName, KER_ARG)`

: Pointer to data plane

### User kernel argument attributes

Apply to tiled kernel arguments.

`K_Arg(UserKernelArgName, KER_ARG_TILE_W)`

: Width of the current tile. Unsigned int.

`K_Arg(UserKernelArgName, KER_ARG_TILE_W0)`

: Default tile width, last tile can be smaller. Unsigned int.

`K_Arg(UserKernelArgName, KER_ARG_TILE_H)`

: Height of the current tile. Unsigned int

`K_Arg(UserKernelArgName, KER_ARG_TILE_H0)`

: Default tile height, last tile can be smaller

`K_Arg(UserKernelArgName, KER_ARG_NTILES)`

: Number of tiles in the data plane. Unsigned int.

`K_Arg(UserKernelArgName, KER_ARG_TILEINDEX)`

: Current tile index. Unsigned int.

`K_Arg(UserKernelArgName, KER_ARG_TILEOFFSET)`

: Current tile offset starting from the origin of the iteration sub space of this user kernel argument. Unsigned int.

## User kernel C arguments

C\_Arg(UserKernelCArgName)

## Subscripted user kernel C arguments

Subscripted by the current index of input plane, output plane or tile multiplied by a constant

C\_ArgIndex(UserKernelCArgName, [Iterator], MultFactor)

: where [Iterator] is KER\_IN\_PLANE, KER\_OUT\_PLANE or KER\_TILE. Note that the C kernel argument has to be a pointer for this binding to be legal

## Immediate values

*Imm(ImmediateIntegerValue)*

**Note:** the binding list order has to follow the basic kernel argument list order.

For example MyFavoriteKernel below contains a single call to the library kernel MatrixAdd located in the inner loop.

```
UserKernel("MyFavoriteKernel",
    KernelDimensions(4, 32, 28, 4),
    KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
    TILE_HOR,
    CArgs(4,
        TCArg("Word32 * __restrict__ ", "In1"),
        TCArg("Word32 * __restrict__ ", "In2"),
        TCArg("Word32 * __restrict__ ", "Out"),
        TCArg("Word32 * __restrict__ ", "Bias")
    ),
    Calls(1,
        Call("MatrixAdd", LOC_INNER_LOOP,
            Bindings(6,
                K_Arg("KerIn1", KER_ARG_TILE), // A tile
                K_Arg("KerIn2", KER_ARG_TILE), // A tile
                K_Arg("KerOut", KER_ARG_TILE), // A tile
                C_ArgIndex("Bias", KER_OUT_PLANE, 1), // Bias[CurrentOutputPlane*1]
                K_Arg("KerIn1", KER_ARG_TILE_W), // Tile width
                K_Arg("KerIn1", KER_ARG_TILE_H) // Tile height
            )
        )
    ),
    ...
);
```

## KerArg - User kernel arguments

User kernel arguments are the inputs and outputs to the user kernel, the entities that will undergo tiling to allow them to fit into the available level 1 memory.

1. A user kernel argument has a direction: input, output or input output. It can be buffered or not. Being buffered means that the entire argument content is moved into level 1 memory before the user kernel iteration space is traversed and is returned to level 2 or level 3 memory afterwards.
2. A user kernel argument is a collection of data planes of width W and height H. In the simplest case the data plane is a collection of 1, it is a 2D structure (Note that if you need to process 1D data you set H to 1). In more elaborate cases the kernel argument can be of dimension 3 or of dimension 4. If the direction of the argument is IN then a 3D argument is a collection of  $N_{IP} \times W \times H$  input data planes. Similarly, if the direction of the argument is OUT and only OUT (IN\_OUT is considered as IN) then the kernel argument will be a collection of  $N_{OP} \times W \times H$  output data planes. A 4D argument in the current implementation can only be an input and is a collection of  $N_{IP} \times N_{OP} \times W \times H$  input data planes. A user kernel argument is fully characterized by its declared width and height, its dimension and the number of input and output data planes shared by all user kernel arguments and declared in the user kernel dimension section. Note that the width and height override the default values of the kernel dimension section.
3. A user kernel argument has a home location in level 2 memory, the default, or in external, level 3 memory. In the case that an argument is in the external memory it can be of OUT type only if the external memory can support it. External flash only supports IN arguments while external RAM supports both IN and OUT arguments. As a special case, uninitialized buffers are transient objects with a home location in the shared level 1 memory.
4. A user kernel argument can be accessed in a non-pipelined way or in a pipelined way (double or triple buffered). When non-pipelined, memory transfers will affect performance since the kernel must wait for the end of the memory transfer before moving on. When pipelined, the memory transfer is performed in a different buffer than the one used for the current tile. Therefore it has all the cycles between the read and the first usage in the next iteration to complete. If the compute time spent in the basic kernels is higher than the number of cycles spent for the memory transfer, the memory transfer will not affect performance. External memory kernel arguments are always treated as pipelined object. The bandwidth gap between the external memory and the shared level 1 or the level 2 memory is such that it makes little sense to perform non-pipelined accesses to or from level 3 memory. For arguments in level 2 memory, the user can decide to let the kernel access them in a pipelined way or not.
5. Kernel arguments can be bound to user kernel arguments but can also be bound to an output argument of another user kernel argument belonging to the same group of user kernels (see after). They can even be working transient memory (buffer) that the kernel needs for its own sake. This buffer can be a full size data plane or only a single tile whose exact dimension will be figured out by the auto tiler.

As you can see, each user kernel argument can have a different width and height. The GAP automatic tiler models the ratios between these different dimensions and tries to find out a tile size (potentially different for each user kernel argument) that will fit within the shared level 1 memory budget it has been given. Tiles can be subject to additional constraints:

- They may have to overlap. This is the case when a 2D filter is applied on a 2D data plane. Let's assume that the set of filter coefficients is a 5 by 5 matrix then 2 adjacent tiles must overlap by 4 to be able to produce the correct output.
- For algorithmic reasons it might be necessary to constrain a tile to be of odd or even size. For example, if the algorithm performs sub-sampling by a factor of 2 then all tiles but the last one must be of even dimension. Similarly, it can be desirable, for hardware constraint or performance reasons, to constrain a tile to be a multiple of a given constant. For example, on GAP the hardware convolution engine works best when it is consuming vertical strips of width 32 therefore if we set the tiling orientation to vertical and set the preferred tile size to be a multiple of 32, we will get maximum performance. As another example, if each line of a tile is given to a core for processing, then having a tile dimension being a multiple of 8 will ensure the 8 cores of the cluster are optimally balanced.

All these constraints can be expressed in a user kernel argument.

To create a kernel argument, the following library call is used:

```
Object_T *KerArg(
    char *KerArgName,
    Object_Type_T ObjType,
    unsigned int W,
    unsigned int H,
    unsigned int ItemSize,
    unsigned int TileOverlap,
    KernelArgConstraints_T Constraint,
    unsigned int PreferedTileSize,
    char *CArgName,
    unsigned int NextPlaneUpdate);
```

## User kernel argument object type

A user kernel argument object type is either a flag built from a set of basic user kernel argument properties or a pre-defined name.

### \*Names built as a flag from user kernel argument properties\*

The set of pre-defined properties which can be OR'ed (|) together is:

**O\_IN, O\_NIN** : Is an input or not an input

**O\_OUT, O\_NOUT** : Is an output or not an output

**O\_BUFF, O\_NBUFF** : Is a buffer or not a buffer

**O\_TILED, O\_NTILED** : Is tiled or not. When not tiled, the whole data plane is accessible in shared level 1 memory.  
For example, a 2D convolution has one argument that is tiled (the input data) and another one that is not tiled (the coefficients).

**O\_ONETILE, O\_NONETILE** : A buffer property. When ONETILE is set, a buffer of dimension WxH will be given only the dimension of a tile whose size is proportional to the size of the tiles generated for the other kernel arguments.

**O\_DYNTILE, O\_NDYNTILE** : A buffer property. When O\_DYNTILE, the height if tiling orientation is horizontal or the width if orientation is vertical, of the buffer will be adjusted to the number of tile computed by GAP auto-tiler.  
This is useful when a dynamic buffer is needed to implement a reduction phase after a result has been computed for each tile independently and final result must be obtained combining all these results into a single one. Usually the declared W or H is the one of another user kernel input and the auto-tiler adjusts it.

**O\_DB, O\_NDB** : Argument is double or triple buffered in L1 memory, or it is not multi-buffered

**O\_L2DB, O\_NL2DB** : Argument home location is an external memory and is double or triple buffered in level 2 memory, or it is not an external memory

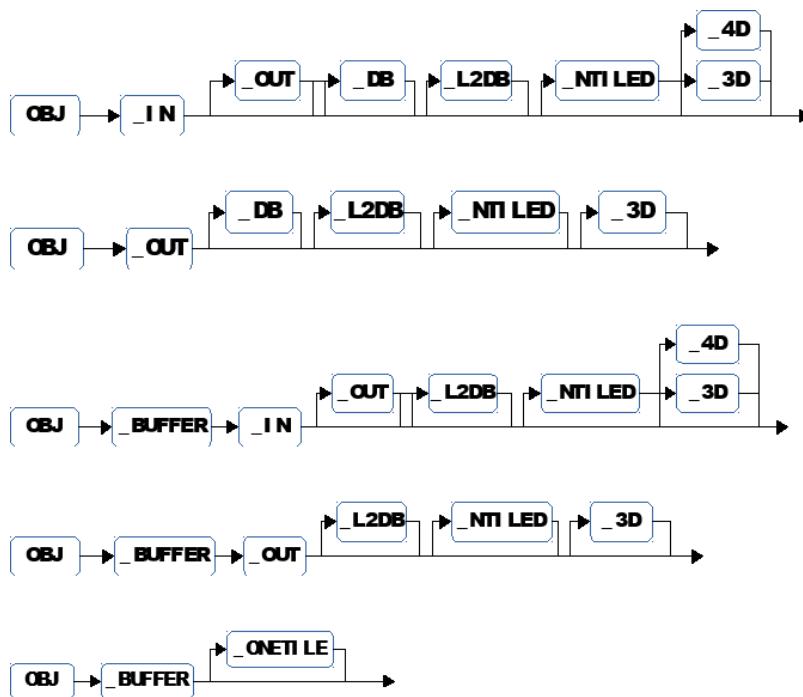
**O\_3D, O\_N3D** : Argument has 3 dimensions or not

**O\_4D, O\_N4D** : Argument has 4 dimensions or not (note that a 4D argument is also a 3D one)

For example, *O\_IN|O\_DB|O\_L2DB|O\_4D* is an input pipelined in level 1 memory and in level 2 memory whose home location is external memory. Its dimension is 4.

### \*Pre-defined names\*

The diagrams below summarize the set of pre-defined names.



For example **OBJ\_IN\_DB\_L2DB\_4D** is an input pipelined in level 1 and in level 2 memory whose home location is external memory. It's dimension is 4.

### User kernel argument width and height

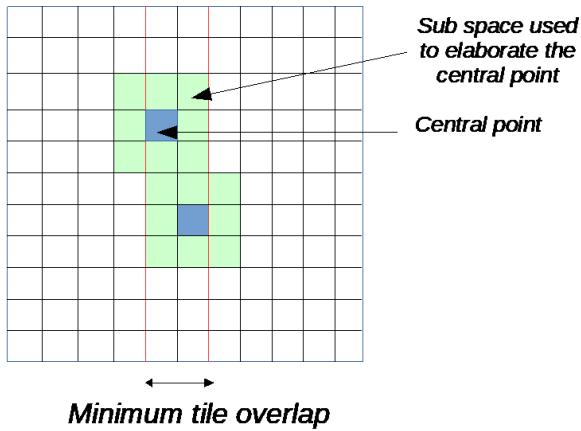
A pair of unsigned ints specifying the dimension of the data plane.

### User kernel argument item size

An unsigned int specifying the size in bytes of the data plane elementary data type.

## User kernel argument tile overlap

An unsigned int capturing the amount of overlap between two adjacent tiles. This is useful in case an output is computed as a function of a point in the input data plane and its neighborhood. If to compute an output you need inputs at a maximum distance K from the point of computation then 2 adjacent tiles should overlap by at least  $2 \times K$  points.



## User kernel argument constraints

Specifies a constraint on a property of the tile dimension that is calculated by the GAP auto-tiler. When tiling is horizontal the constraint is on the height of the tile, and when tiling is vertical the constraint is on the width of the tile. A constraint will be applied by the auto-tiler to all the tiles except the last one. If this is not possible the model cannot be tiled.

These are the possible constraints:

```
typedef enum {
    OBJ_CONSTRAINTS_NONE = 0,
    OBJ_CONSTRAINTS_EVEN = (1<<1),      /* Tile variable size is even */
    OBJ_CONSTRAINTS_ODD = (1<<2),       /* Tile variable size is odd */
    OBJ_CONSTRAINTS_ONEPREFILE = (1<<3),  /* Limit number of used tile to just one */
    OBJ_CONSTRAINTS_TILE_HOR = (1<<4),    /* Overide default orientation to TILE_HOR */
    OBJ_CONSTRAINTS_TILE_VER = (1<<5),    /* Overide default orientation to TILE_VER */
} KernelArgConstraints_T;
```

### User kernel argument preferred tile size

Specifies the developer's preference for the dimension of the tile that is calculated by the GAP auto-tiler. It is expressed as an unsigned int and when not zero the preferred dimension of the tile chosen is a multiple of this value.

### User kernel argument binding to C user kernel argument

Generally, a user kernel argument is connected to a C argument and in this case the name of this C argument should be provided. In the case where the user kernel argument is only internal to the user kernel or user kernel group then there is no binding and null (0) should be used.

### User kernel argument next plane update

Generally, the mechanism to move from one data plane to another one in the iteration space for user kernel arguments with dimensions strictly greater than 2 is inferred from the object type. In some cases, it can be desirable to give a better control on this update process. For example, the GAP hardware convolution engine can produce 3 full 3 x 3 convolution outputs. Here 3 adjacent output data planes are involved and therefore the move to next group of outputs should use a step of 3 \* size and not 1 \* size of the output plane as is the case for an implicit update.

Next plane update is expressed as a non 0 unsigned integer. If 0 then default rule is applied.

Back to our simple matrix addition example, a possible final version is shown below:

```
UserKernel("MyFavoriteKernel",
    KernelDimensions(4, 32, 28, 4),
    KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
    TILE_HOR,
    CArgs(4,
        TCArg("Word32 * __restrict__", "In1"),
        TCArg("Word32 * __restrict__", "In2"),
        TCArg("Word32 * __restrict__", "Out"),
        TCArg("Word32 * __restrict__", "Bias")
    ),
    Calls(1,
        Call("MatrixAdd", LOC_INNER_LOOP,
            Bindings(6,
                K_Arg("KerIn1", KER_ARG_TILE), // A tile
                K_Arg("KerIn2", KER_ARG_TILE), // A tile
                K_Arg("KerOut", KER_ARG_TILE), // A tile
                C_ArgIndex("Bias", KER_OUT_PLANE, 1), // Bias[CurrentOutputPlane*1]
                K_Arg("KerIn1", KER_ARG_TILE_W), // Tile width
                K_Arg("KerIn1", KER_ARG_TILE_H) // Tile height
            )
        )
    ),
    KerArgs(3,
        KerArg("KerIn1", OBJ_IN_DB_3D, 75, 75, sizeof(Word32), 0, 0, 0, "In1", 0),
        KerArg("KerIn2", OBJ_IN_DB_3D, 75, 75, sizeof(Word32), 0, 0, 0, "In2", 0),
        KerArg("KerOut", OBJ_OUT_DB_3D, 75, 75, sizeof(Word32), 0, 0, 0, "Out", 0)
    )
);
```

The first argument is an input coming from L2 memory and it is multi buffered in shared L1 memory so that performance is optimized. The basic plane is  $75 \times 75$  of integers (4 bytes). It has 3 dimensions and since we have declared that we have 4 input planes in the KernelDimensions section we have 4 basic planes.

The second argument has the same characteristics than the first argument.

The third argument is an output that should go to level 2 memory and that is multi buffered in level 1 memory again to incur no performance penalty due to memory transfers. It has 3 dimensions and since we have declared that we have 4 output planes in the KernelDimensions section we have 4 basic planes.

The 3 user kernel arguments are tiled.

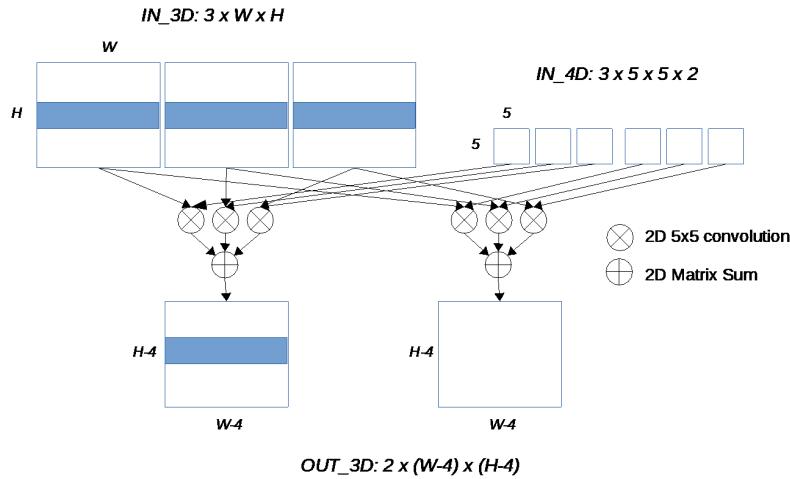
What this example does is to add in each output matrix the sum of all input matrices plus a scalar bias.

```
Out[0] = (In1[0][75:75]+In2[0][75:75]+Bias[0]) +
          (In1[1][75:75]+In2[1][75:75] + Bias[0]) +
          (In1[2][75:75]+In2[2][75:75]+Bias[0]) +
          (In1[3][75:75]+In2[3][75:75] + Bias[0]);
Out[1] = (In1[0][75:75]+In2[0][75:75]+Bias[1]) +
          (In1[1][75:75]+In2[1][75:75] + Bias[1]) +
          (In1[2][75:75]+In2[2][75:75]+Bias[1]) +
          (In1[3][75:75]+In2[3][75:75] + Bias[1]);
Out[2] = (In1[0][75:75]+In2[0][75:75]+Bias[2]) +
          (In1[1][75:75]+In2[1][75:75] + Bias[2]) +
          (In1[2][75:75]+In2[2][75:75]+Bias[2]) +
          (In1[3][75:75]+In2[3][75:75] + Bias[2]);
Out[3] = (In1[0][75:75]+In2[0][75:75]+Bias[3]) +
          (In1[1][75:75]+In2[1][75:75] + Bias[3]) +
          (In1[2][75:75]+In2[2][75:75]+Bias[3]) +
          (In1[3][75:75]+In2[3][75:50] + Bias[3]);
```

As you can see each matrix occupies  $75 * 75 * 4 = 22.5$  Kbytes. There are 4 of them for In1, 4 for In2 and 4 for Out so a total of 270Kbytes. Clearly this does not fit in the shared L1 memory. The GAP auto-tiler produces code that will transparently move sections of the 270Kbytes back and forth from L2 to shared L1 and make them available to the basic kernel function doing the actual calculation (our special matrix addition) making sure all the cores are always active.

### User kernel example - CNN convolution layer

We assume 3 input data planes of size  $W \times H$  and 2 output data planes of size  $W - 4 \times H - 4$  (padding is not performed). The diagram below gives a high-level view of what needs to be done.



Since to get the result, we must sum up the convolution results from all input planes, we can also assume that we start the summation with a matrix made up of identical values, a bias. Planes contain fixed point numbers made up of 16 bits (short int).

Instead of having a specialized implementation, we want to allow the number of input and output data planes to be specified so we simply embed the user kernel in a C function with proper arguments.

Our 2 basic kernels are KerSetInBias that copies the same constant value in a matrix and KerDirectConv5x5\_fp that takes care of the convolution itself. The convolution takes a 2D input In, a set of filter coefficients (25 in this case) and produces a 2D output. It performs normalization shifting the result by Norm.

```
void GenerateCnnConv5x5(char *Name, unsigned int InPlane,
    unsigned int OutPlane, unsigned int W, unsigned int H)
{
    UserKernel(Name,
        KernelDimensions(InPlane, W, H, OutPlane),
        KernelIterationOrder(KER_DIM4, KER_OUT_PLANE, KER_TILE, KER_IN_PLANE),
        TILE_HOR,
        CArgs(5,
            TCArg("short int * __restrict__ ", "In"),
            TCArg("short int * __restrict__ ", "Filter"),
            TCArg("short int * __restrict__ ", "Out"),
            TCArg("unsigned int", "Norm"),
            TCArg("short int * __restrict__ ", "Bias"))
    ),
    Calls(2,
        Call("KerSetInBias", LOC_IN_PLANE_PROLOG,
            Bindings(4,
                K_Arg("Out", KER_ARG_TILE),
                K_Arg("Out", KER_ARG_TILE_W),
                K_Arg("Out", KER_ARG_TILE_H),
                C_ArgIndex("Bias", KER_OUT_PLANE, 1)
            )
        ),
        Call("KerDirectConv5x5_fp", LOC_INNER_LOOP,
            Bindings(6,
                K_Arg("In", KER_ARG_TILE),
                K_Arg("In", KER_ARG_TILE_W),
                K_Arg("In", KER_ARG_TILE_H),
                K_Arg("In", KER_ARG_TILE_W),
                K_Arg("In", KER_ARG_TILE_H),
                K_Arg("In", KER_ARG_TILE_W)
            )
        )
    )
}
```

(continues on next page)

(continued from previous page)

```

        K_Arg("Filter", KER_ARG_TILE),
        K_Arg("Out", KER_ARG_TILE),
        C_Arg("Norm")
    )
)
),
KerArgs(3,
    KerArg("In", OBJ_IN_DB_3D, W, H, sizeof(short int), 5-1, 0, 0, "In", 0),
    KerArg("Filter", OBJ_IN_DB_NTILED_4D, 5, 5,
        sizeof(short int), 0, 0, 0, "Filter", 0),
    KerArg("Out", OBJ_OUT_DB_3D, W-5+1, H-5+1,
        sizeof(short int), 0, 0, 0, "Out", 0)
)
);
}
}

```

This example illustrates:

- 3D and 4D user kernel arguments
- Multi-buffered in and out
- Overlapping tiles (first argument)
- Untiled arguments (second argument). We need this because the filter applies to the entire input data plane not to a given point in it
- Calling several basic kernels at different steps in the iteration. Setting the bias must be performed before we start producing the series of convolutions e.g before we start traversing the input planes hence in LOC\_IN\_PLANE\_PROLOG. The convolution itself must be performed on every single tile so it is in the inner loop.
- Bindings to user kernel arguments tiles and tile attributes (K\_Arg())
- Bindings to user kernel C arguments either as plain scalar (C\_Arg()) or as plane indexed (C\_ArgIndex()).

## User kernel groups

It is useful to decompose a user kernel into a group of connected user kernels. User kernel groups allow this.

A user kernel group is described by a set of 2 anchors around a list of user kernel definitions. The first anchor marks the beginning of the user kernel group and defines its name. The second anchor closes the group. Then a second section models the C kernel template for the group and how the user kernels in the user kernel group should be connected together.

Here is the API to open and to close a kernel group:

```

void OpenKernelGroup(
    // The name of the group, will be the name of the C template
    // generated for this group.
    char *GroupName
);

void CloseKernelGroup();

```

To model a kernel group the API is:

```
void UserKernelGroup(
    // The same name than the one used in OpenKernelGroup
    char *GroupName,
    // A list of C arguments created with CArgs(Count, ...)
    CKernel_Arg_T **Cargs,
    // A list of user kernel calls created with Calls(Count, ...)
    CKernelCall_T **Calls
);
```

As a reminder:

```
CKernel_Arg_T **Cargs(
    unsigned int ArgCount,
    ... // A list of TCArg()
);

CKernel_Arg_T *TCArg(
    char *ArgType,
    char *ArgName);

CKernelCall_T **Calls(
    unsigned int CallCount,
    ... // A list of UserKernelCall()
);
```

Instead of Call() which is used to model a call to a basic kernel, we use UserKernelCall(). Its interface is the same as Call() but it has some additional restrictions.

```
CKernelCall_T *UserKernelCall(
    char *CallName,
    KernelCallLocationT CallLocation, // Here can only be LOC_GROUP
    ArgBindingDescr_T **BindingList // Here only these 2 bindings
    // can be used: CArg() and Imm()
);

ArgBindingDescr_T **Bindings(
    int Bcount,
    ... // A list of bindings, CArg() and Imm() only in this context
);

ArgBindingDescr_T *C_Arg( char *ArgName);

ArgBindingDescr_T *Imm( int Value);
```

Some details on the user kernel group call sequence:

1. By construction a user kernel group is made up of a sequential call to a sequence of user kernels so all calls will be performed by the cluster master core (core 0) only.
2. In user kernels, the call sequence is made up of basic kernels and each call is inserted at a provided location in the iteration structure. In a user kernel group, calls are only to user kernels and since there is no iteration structure in a group, the call location is LOC\_GROUP.

## User kernel group example, 2D FFT

The input is a 2D byte pixel image of dimension Dim x Dim. Dim has to be a power of 2.

We first have to expand the input image into I/Q pairs. I and Q are the imaginary and real parts of a complex number, both represented as Q15 fixed point numbers in a short int. We use the Image2Complex() basic kernel. Tiling is horizontal.

Then we compute the FFT of each line of the expanded input, all the FFTs are independent. Therefore they can be evaluated in parallel. We use either a radix 4 or radix 2 FFT decimated in frequency (inputs in natural order, output in bit reverse order).

Finally, we reorder the FFT outputs with SwapSamples\_2D\_Horizontal\_Par.

These 3 steps can be grouped together into a single user kernel in charge of the horizontal FFTs.

Once horizontal FFTs have been computed, we have to compute a 1D FFT on each column of the horizontally transformed input, so we use vertical tiling. Since the vertical FFT is also decimated in frequency, we have to reorder the vertical FFTs output. These 2 basic kernels are grouped into a single user kernel in charge of the vertical FFTs.

```
void FFT2D(char *Name, unsigned int Dim, int ForceRadix2)
{
    char *KerHorizontal, *KerVertical;

    // Dim is the dimension of the FFT, First select the right 1D FFT (Radix 2
    // or 4) depending on dim
    if (__builtin_popcount(Dim) != 1)
        GenTilingError(
            "FFT2D: %s, Incorrect Dim: %d, it has to be a power of 2", Name, Dim);
    else if ((__builtin_ctz(Dim)%2)==0) {
        /* Radix 4 FFT */
        KerHorizontal = "Radix4FFT_DIF_2D_Horizontal";
        KerVertical = "Radix4FFT_DIF_2D_Vertical";
    } else {
        /* Radix 2 FFT */
        KerHorizontal = "Radix2FFT_DIF_2D_Horizontal";
        KerVertical = "Radix2FFT_DIF_2D_Vertical";
    }
    // Here we open the kernel group
    OpenKernelGroup(Name);
    // First user kernel in the group.
    // The group input is a byte image, we have to expand it to a
    // complex representation (I and Q in Q15).
    // Then we perform a 1D FFT on each line of the expanded input
    // Finally we reorder the FFT output
    UserKernel(AppendNames(Name, "Horizontal"),
               KernelDimensions(1, Dim, Dim, 1),
               KernelIterationOrder(KER_DIM2, KER_TILE),
               TILE_HOR,
               CArgs(4,
                     TCArg("Wordu8 * __restrict__", "In"),
                     TCArg("v2s * __restrict__", "Out"),
                     TCArg("Word16 * __restrict__", "Twiddles"),
                     TCArg("Word16 * __restrict__", "SwapTable")
               )),

```

(continues on next page)

(continued from previous page)

```

Calls(3,
    Call("Image2Complex", LOC_INNER_LOOP,
        Bindings(4,
            K_Arg("In", KER_ARG_TILE),
            K_Arg("Out", KER_ARG_TILE),
            K_Arg("In", KER_ARG_TILE_W),
            K_Arg("In", KER_ARG_TILE_H))),
    Call(KerHorizontal, LOC_INNER_LOOP,
        Bindings(4,
            K_Arg("Out", KER_ARG_TILE),
            C_Arg("Twiddles"),
            K_Arg("Out", KER_ARG_TILE_W),
            K_Arg("Out", KER_ARG_TILE_H))),
    Call("SwapSamples_2D_Horizontal_Par", LOC_INNER_LOOP,
        Bindings(4,
            K_Arg("Out", KER_ARG_TILE),
            C_Arg("SwapTable"),
            K_Arg("Out", KER_ARG_TILE_W),
            K_Arg("Out", KER_ARG_TILE_H)))
),
KerArgs(2,
    KerArg("In", OBJ_IN_DB, Dim, Dim, sizeof(Wordu8), 0, 0, 0, "In", 0),
    KerArg("Out", OBJ_OUT_DB, Dim, Dim, sizeof(Word32), 0, 0, 8, "Out", 0)
)
);
// Second user kernel in the group.
// This user kernel takes as an input the output of the horizontal FFT step
// It performs a 1D FFT on each column of the input matrix (note that here
// the tiling is vertical)
// Finally we reorder the FFT output and we are done
UserKernel(AppendNames(Name, "Vertical"),
    KernelDimensions(1, Dim, Dim, 1),
    KernelIterationOrder(KER_DIM2, KER_TILE),
    TILE_VER,
    CArgs(3,
        TCArg("Word16 * __restrict__", "InOut"),
        TCArg("Word16 * __restrict__", "Twiddles"),
        TCArg("Word16 * __restrict__", "SwapTable")
),
Calls(2,
    Call(KerVertical, LOC_INNER_LOOP,
        Bindings(4,
            K_Arg("InOut", KER_ARG_TILE),
            C_Arg("Twiddles"),
            K_Arg("InOut", KER_ARG_TILE_H),
            K_Arg("InOut", KER_ARG_TILE_W))),
    Call("SwapSamples_2D_Vertical_Par", LOC_INNER_LOOP,
        Bindings(4,
            K_Arg("InOut", KER_ARG_TILE),
            C_Arg("SwapTable"),
            K_Arg("InOut", KER_ARG_TILE_H),
            K_Arg("InOut", KER_ARG_TILE_W)))
)
);

```

(continues on next page)

(continued from previous page)

```

),
KerArgs(1,
    KerArg("InOut", OBJ_IN_OUT_DB, Dim, Dim, sizeof(Word32), 0, 0, 8, "InOut", 0)
)

};

// The kernel group is closed
CloseKernelGroup();

// Now we model the kernel group C template, then we provide the list of
// user calls, in this case horizontal and vertical FFT and we model the
// bindings between user kernel group C arguments and the 2 called
// user kernels in the group
UserKernelGroup(Name,
    CArgs(4,
        TCArg("Wordu8 * __restrict__", "In"),
        TCArg("Word32 * __restrict__", "Out"),
        TCArg("Word16 * __restrict__", "Twiddles"),
        TCArg("Word16 * __restrict__", "SwapTable")
    ),
    Calls(2,
        UserKernelCall(AppendNames(Name, "Horizontal"), LOC_GROUP,
            Bindings(4, C_Arg("In"), C_Arg("Out"), C_Arg("Twiddles"), C_Arg("SwapTable"))),
        UserKernelCall(AppendNames(Name, "Vertical"), LOC_GROUP,
            Bindings(3, C_Arg("Out"), C_Arg("Twiddles"), C_Arg("SwapTable")))
    )
);
}
}

```

## Controlling tiled code generation

As already mentioned a model is just a C program. To get the generated code you need to compile and execute the model.

The model makes use of the GAP auto-tiler library calls for which a header file needs to be included and a library to be added to the link list. All interfaces and data types are declared in `AutoTilerLib.h` and the library itself is `AutoTilerLib.a`.

Below is a simple example of a complete generator using the auto-tiler, `MyProgram.c`:

```

#include <stdint.h>
#include <stdio.h>
#include "AutoTilerLib.h"

void MySetOfUserKernels(unsigned int L1_Memory_Budget)
{
    /*
        Here you include:
        Loading basic kernel libraries
        List of user kernels and user kernels groups
    */
}

```

(continues on next page)

(continued from previous page)

```

void ConfigureTiler()
{
    /*
        Here you include
        header files for the basic kernels
        Files to be produced
        Tiler options
    ...
}
int main(int argc, char **argv)
{
    // First use library API to parse program options and initialize the tile
    if (TilerParseOptions(argc, argv)) {
        printf("Failed to initialize or incorrect arguments.\n"); return 1;
    }
    // Setup Gap8 auto tiler preferences
    ConfigureTiler();
    // Then call the function in which the user kernel models are, we give 51200 bytes of
    // memory as a budget
    MySetOfUserKernels(51200);
    // Now that the model is built we produce code for it
    GenerateTilingCode();
    return 0;
}

```

To compile and use the sample:

```

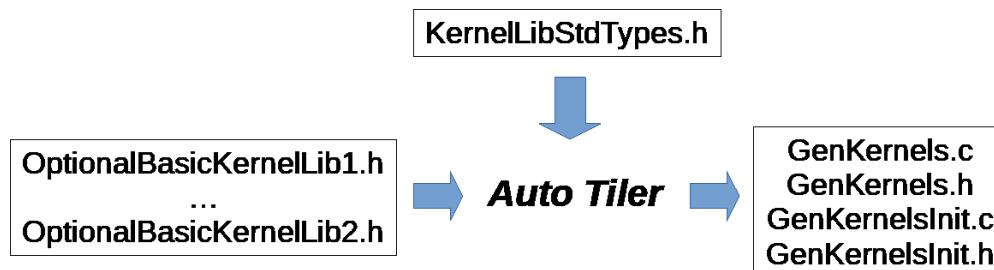
gcc
-o MyGenerator
-I<path to where tiler libs h files are>
MyModel.c
-L<path to where the tiler lib is>AutoTilerLib.a

```

And run MyGenerator to have the tiled user kernels generated:

```
./MyGenerator
```

The GAP auto-tiler uses and produces files, the general scheme is illustrated below:



**OptionalBasicKernelLibn.h** : This is where the C templates for the basic kernels are expected to be found for proper compilation of the generated code. Both the C templates as well as the struct typedefs for the basic kernel

arguments are included. It can be empty.

**KernelLibStdTypes.h** : Contains some internal tiler argument types definitions. This is the default name but it can be redefined.

The API to redefine KernelLibStdTypes.h and to provide which basic kernel libraries to use is the following:

```
void SetUsedFilesNames(
    /* To redefine KernelLibStdTypes.h, if 0 uses default */
    char StdTypedefsName,
    /* Number of basic kernel libraries to be passed */
    unsigned int LibKernelFileCount,
    /* List of basic kernel libraries passed as strings */
    ...
)
```

Four files are produced by the auto tiler

**GenKernels.c** : The generated code for all user kernels and groups in the model

**GenKernels.h** : Include file for the generated code

**GenKernelsInit.c** : Some variable definitions

**GenKernelsInit.h** : Include file for the initialization code

The names used here are the default names, they can be overridden using the following function:

```
void SetGeneratedFilesNames(
    char *ArgInitsName,      /* Redefinition of GenKernelsInit.c */
    char *ArgInitsHeaderName, /* Redefinition of GenKernelsInit.h */
    char *CallTemplatesName, /* Redefinition of GenKernels.c */
    char *CallTemplatesNameHeader /* Redefinition of GenKernels.h */
)
```

Four other symbols are also manipulated by the auto tiler:

**L1\_Memory** : One symbol, a pointer or an array, in which all the tiles generated by the tiler will be allocated in the shared L1 memory.

**L2\_Memory** : One symbol, a pointer or an array, in which all the tiles generated by the tiler will be allocated in the L2 memory.

**AllKernels** : One symbol, an array, in which user kernel descriptor is stored. Descriptors are generated only when user kernels are not inlined.

**AllKernelsArgs** : One symbol, an array, in which user kernel arguments descriptors are stored, AllKernels will point into it. These descriptors are generated only when user kernels are not in-lined.

These 4 names are default names, they can be changed with the following function:

```
void SetSymbolNames(
    char *L1SymbName,
    char *L2SymbName,
    char *KernelDescrName,
    char *KernelDescArgName
)
```

By default these 4 symbols are managed as arrays, you can make them dynamic (pointers) in order to have the memory allocated and deallocated dynamically. The execution log of the auto tiler tells you the number of bytes to be allocated for each of them.

```
void SetSymbolDynamics()
```

## 6.3 GVSOC

### 6.3.1 Usage

The virtual platform must be launched through the common runner called *pulp-run*.

The specified platform must be *gvsoc* (through option *--platform*) and the only mandatory option is either *--config-file* or *--config* in order to give the system configuration to simulate (see next section).

As the virtual platform will generate several temporary files, it is also good to launch it from a specific folder or to specify it through the *--dir* option.

All the other options for the platform must be given through the configuration file (see next sections). The most common one is the binary to be simulated. A few common options has a shortcut as a direct option to *pulp-run*, you can execute this command to get them:

```
$ pulp-run --platform=gvsoc --config=gap_rev1 --help
```

After the options, the set of commands to execute must be specified. The usual ones are *prepare* for generating the platform stimuli (e.g. to prepare the flash image containing the test binary), and *run* for running the simulation.

Here is a full example:

```
$ pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run
```

### 6.3.2 Configuration

The virtual platform is simulating the architecture which is described by the specified system configuration, described with a JSON file.

This can be first done through the option *--config-file* to give the path of the JSON file. This can be either an absolute path or a relative path, in which case the config is search in the paths given by the environment variable *SDK\_CONFIGS\_PATH*, which contains a list of possible paths separated by *:*.

This can also be done through the option *--config* which gives the name of the configuration to simulate. This is equivalent to *--config-file=chips/<name>/<name>.json* where *<name>* is the value of the option.

The configuration is a high-level description of the architecture, where all important properties are specified (e.g. memory sizes). This high-level view of the architecture is used to generate a low-level and detailed view of the architecture which is used by *gvsoc* to know what to instantiate, configure and connect. Both levels can be customized by the user. The high-level view is called the template, and can be customized to easily change architecture properties such as memory sizes. The low-level view is called the configuration and can be customized to change properties of one specific component, such as a specific behavior of one core.

### 6.3.3 Options

Options to the virtual platform are passed by customizing the system configuration.

This can be first done using the option `--property=<path>=<value>` to specify a property in the JSON file to be overwritten. `<path>` is giving the property path in the JSON file where the property must be overwritten and `<value>` the value to be set. As a JSON file is hierarchical, `<path>` describes a hierarchical path, similar to a file system path. As described in the previous section, a property can be changed either in the template or in the configuration. Any property beginning with `config/` will change a property in the configuration while the others will change it in the template.

Here is an example to activate instructions traces:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --property=config/
↪ gvsoc/trace=insn
```

There is however a shortcut for this property, which can be set with this option:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --trace=insn
```

The second way to customize the system configuration is to provide a INI-style configuration file containing a set of properties through the option `--config-user`. The JSON path of a property is the concatenation of the section name, and the INI property name.

Here is an example of such a file to activate instruction traces:

```
[config.gvsoc]
trace=insn
```

And the command to specify it:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --config-
↪ user=myconfig.ini
```

In both ways, refer to other sections to get the various properties which can be set to configure the system.

### 6.3.4 System traces

#### Description

The virtual platform allows dumping architecture events to help developers debugging their applications by better showing what is happening in the system.

For example, it can show instructions being executed, DMA transfers, events generated, memory accesses and so on.

This feature can be enabled and configured through the option `--trace`. This option takes an argument which specifies a regular expression of the path in the architecture where the traces must be enabled, and optionally a file where the traces should be dumped. All components whose path matches the specified one will dump traces. Several paths can be specified by using the option several times. Here is an example that activates instruction traces for core 0 and core 1:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --trace=pe0 insn --
↪ trace=pe1 insn"
```

The trace file should look like the following:

```
194870000: 19487: [/sys/board/chip/soc/cluster/pe1/insn] M 1c001a96 c.li
  ↳ a2, 0, 0           a2:00000000
194870000: 19487: [/sys/board/chip/soc/cluster/pe0/insn] M 1c001a2c beq
  ↳ a1, s4, 76         a1:00000020 s4:00000025
```

There is usually one line per event, although an event can sometimes take several lines to display more information.

The number on the left gives the timestamp of the event, in picoseconds, and the one right after the number of cycles. Both are given because different blocks like clusters can have different frequencies. The timestamp is absolute and will increase linearly while the cycle count is local to the frequency domain.

The second part, which is a string, gives the path in the architecture where the event occurred. This is useful to differentiate blocks of the same kind that generate the same event. This path can also be used with the `--trace` option to reduce the number of events.

The third part, which is also a string, is the information dumped by the event, and is totally specific to this event. In our example, the core simulator is just printing information about the instruction that has been executed.

### Trace path

One difficulty is usually to find out which paths should be activated to get the needed information. One method is to dump all the events with `--trace=.*`, then find out which ones are interesting and then put them on the command line. Here are the paths for the main components (note that this can differ from one chip to another):

Path	Description
/sys/board/chip/cluster/pe	Processing element, useful to see the IOs made by the core, and the instruction it executes. You can add /iss to just get instruction events
/sys/board/chip/cluster/ev	Hardware synchronizer events, useful for debugging inter-core synchronization mechanisms
/sys/board/chip/cluster/pc	Shared program cache accesses
/sys/board/chip/cluster/l1	Shared L1 interconnect
/sys/board/chip/cluster/l1b	Memory banks (the X should be replaced by the bank number)
/sys/board/chip/soc/l2	L2 memory accesses
/sys/board/chip/cluster/dm	DMA events

At first, the most interesting traces are the core instruction traces. As they show not only the instructions executed but also the registers accessed, their content and the memory accesses, they are very useful for debugging bugs like memory corruptions.

### Instruction traces

Here is an example of instruction trace:

```
4890000: 489: [/sys/board/chip/soc/cluster/pe0/insn] M 1c001252 p.sw 0, 4(a5!) ↳
  ↳ a5=10000010 a5:1000000c PA:1000000c
```

The event information dumped for executed instructions is using the following format:

```
<address> <instruction> <operands> <operands info>
```

<address> is the address of the instruction.

<instruction> is the instruction label.

<operands> is the part of the decoded operands.

<operands info> is giving details about the operands values and how they are used.

The latter information is using the following convention:

- When a register is accessed, its name is displayed followed by = if it is written or : if it is read. In case it is read and written, the register appears twice. It is followed by its value, which is the new one in case it is written.
- When a memory access is done, PA: is displayed, followed by the address of the access.
- The order of the statements is following the order on the decoded instruction

The memory accesses which are displayed are particularly interesting for tracking memory corruptions as they can be used to look for accesses to specific locations.

### How to dump to a file

By default, all traces are dumped to the standard output and it is possible to specify the file where the traces should be dumped. The file must be given for every `--trace` option. The same file can be used, to get all traces into the same file, or different files can be used.

Here is an example to get all possible traces into one file:

```
make run PLT_OPT=--trace=.*:log.txt
```

And another example to get instruction traces to one file and L2 memory accesses to another file:

```
make run PLT_OPT=--trace=insn:insn.txt --trace=l2:l2.txt
```

### 6.3.5 Debug symbols

Some features like instruction traces can use debug symbols to display more information. These features are by default enabled and can be disabled with the option `--no-debug-syms`.

To have such features working, the binaries must be compile in debug mode so that debug symbols are present in the binaries and the virtual platform can generate debug symbols information.

The toolchain must be accessible for this option to work, either by making sure it is in accessible through environment variable PATH or by defining this environement variable:

```
export PULP_RISCV_GCC_TOOLCHAIN=<path containing bin/riscv32-unknown-elf-readelf>
```

Once this works, the instruction trace should look like the following:

```
9398037447: 466538: [/sys/board/chip/soc/fc/insn ] _get_next_
↳ timeout_expiry:167      M 1c001d7c sw          ra, 28(sp)    ra:1c002154 ↳
↳ sp:1b000db0  PA:1b000dcc
```

There is a column which displays the debug information. There are 2 information separated by :, the first one is the function which this instruction belongs to, and the second is the line number of the instruction in the source code.

### 6.3.6 VCD traces

The virtual platform can dump VCD traces which show the state of several components over the time, like the cores PC, the DMA transfers, etc, and thus gives a better overview than the system traces.

#### Configuration

VCD tracing can be activated through option `--vcd`:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --vcd
```

Once the platform is run, this will generate a VCD file called `all.vcd` in the build folder. This file is a raw file containing all the signals value.

Another file called `view.gtkw` is generated and can be opened using Gtkwave. This is a script file which will setup the view with the most interesting signals. The command to be executed is displayed at the beginning of the simulation when VCD traces are enabled.

#### Trace format

The default format is the FST gtkwave format, as it is much faster and smaller than VCD. However, it is less robust and can make Gtkwave crash. The following option can be used to change the format to VCD:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --vcd --event-  
--format=vcd
```

#### Display

Any VCD viewer can be used to display the traces. On Linux the free Gtkwave viewer can be used. For example to display the PC traces, you can launch it with:

```
gtkwave <vcd file path>
```

Then click on Search->Search Signal Regexp, enter “pc”, click on Select All and Insert, and close the box. You should now see the PC traces in the view, you can zoom out to see the full window.

It is also possible to open the generated script file mentioned above with this command:

```
gtkwave <script path>
```

#### Trace selection

More traces can be activated by either specifying trace tags or names. Tags will activate a set of traces while names will activate specific traces.

Tags can be activated with the option `--event-tag=<name>`. This option can be given several times to specify several tags. The tag `overview` is always selected, and others can be selected from this list: `debug`, `asm`, `pc`, `core_events`, `clock`.

Here is an example:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --vcd --event-  
--tag=debug --event-tag=core_events
```

Specific events can be selected with the option `--event=<name>`. This option can be given several times to specify several traces. Like for system traces, the name is a regular expression which will be compared against the path of each trace. Any trace which will match the regular expression will be enabled.

Here is an example to activate all traces:

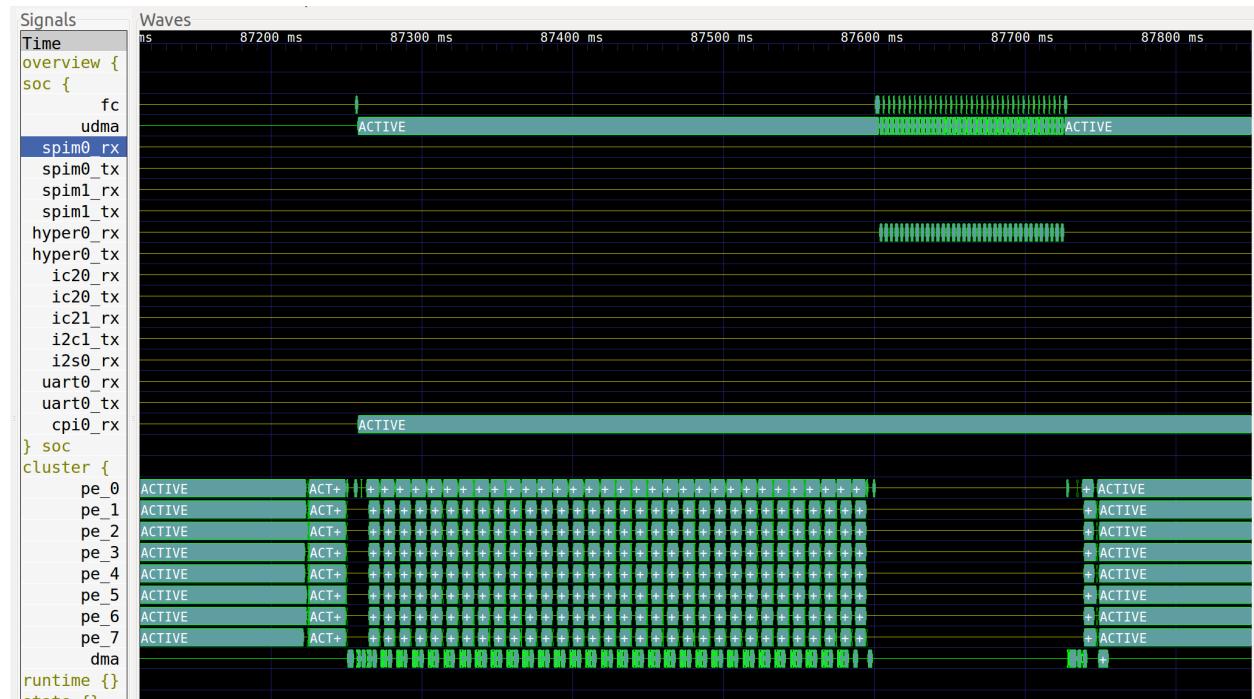
```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --vcf --event-
→ tag=debug --event=.*
```

## View description

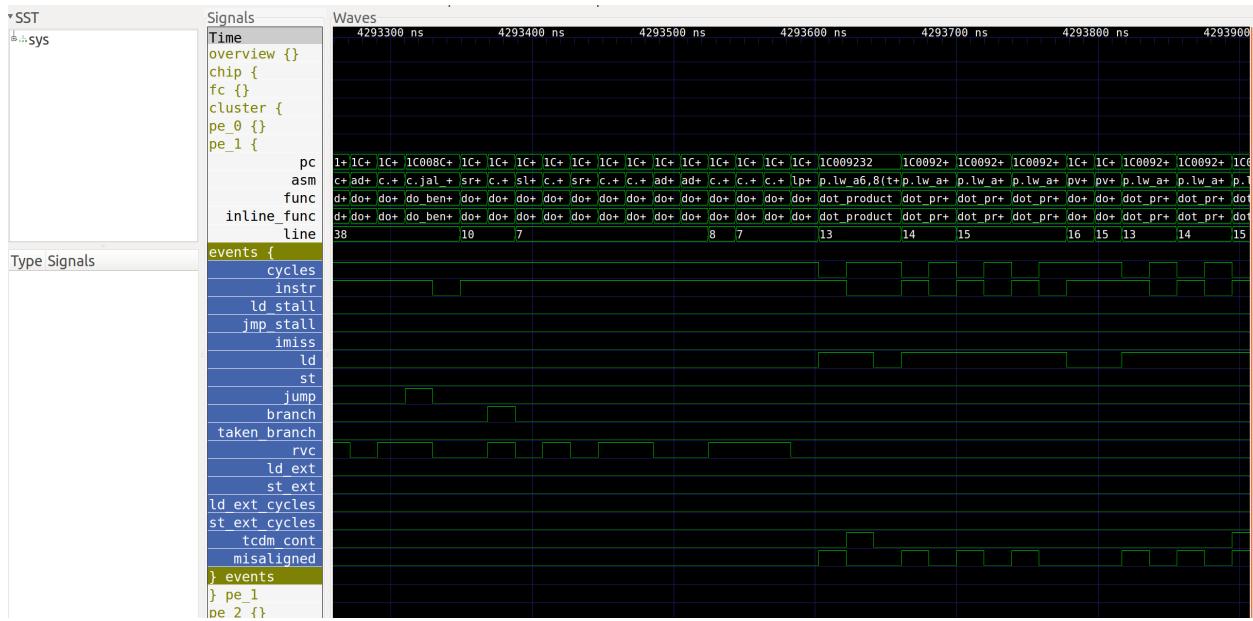
The view displayed from the Gtkwave script is made of 2 parts.

The first part, on the top (see the image below), is showing an overview of the execution with the most useful signals. It basically shows the state of each important block in the system. This is useful to quickly check what is being executed in the whole system.

Some groups of this part are by default closed, and can be opened by double-clicking on them, like the group *stats* which shows the number of instructions per cycle (IPC) for each core. This number is an average and can be slightly shifted with respect to the instructions executed.



The second (see the image below), is showing a more detailed view of the execution (additional tags or traces must be specified). The program counter is shown, with also debug information about the function being executed, the disassembled instructions and so on. For each core, a group called *events* contained information about the state of the core (stalls, loads, instructions, etc). This is useful to understand why a core is being stalled.



### Interactive mode

In case the trace file becomes too big, it is possible to open gtkwave in interactive mode so that it is getting the traces in real time. For that launch the platform with this option:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --event=.* --gtkw
```

This will automatically open Gtkwave and the traces are automatically updated.

### 6.3.7 Application profiling

The virtual platform is for now not providing any particular feature in terms of profiling except for hardware performance counters whose most of them are modeled.

To use them, the test should configure and use them as on the real silicon, with the difference that on gvsoc all performance counters are implemented, not only one.

### 6.3.8 Timing models

Timing models are always active, there is no specific option to set to activate them. They are mainly timing the core model so that the main stalls are modeled. This includes branch penalty, load-use penalty and so on. The rest of the architecture is slightly timed. Remote accesses are assigned a fixed cost and are impacted by bandwidth limitation, although this still not reflect exactly the HW (the bus width may be different). L1 contentions are modeled with no priority. DMA is modeled with bursts, which gets assigned a cost. All UDMA interfaces are finely modeled.

### 6.3.9 Power models

This is for now a very preliminary work. Power traces can be showed when VCD traces are activated. Each power source is able to register an amount of energy, which is showed inside a VCD trace as a pulse.

For now only the core is registering the energy consumed by an instruction, but all instructions are assigned a fixed cost, which just has an arbitrary value.

A more detailed power report will soon be produced, and power sources added.

### 6.3.10 Devices

#### Bluetooth

##### NINA B112

The Nina B112 model mimicks basic functions of a real Ublox Nina B112 SPP (Serial Peripheral Profile).

The real device is used to act as a UART bridge over Bluetooth.

Current model supports only a minimal subset of all AT commands. It does not support extended data mode (EDM).

When in data mode it acts as a loopback. Once a number of bytes were received, it sends everything back.

It supports RTS generation. When enabled, the model generates RTS for a specific duration on two conditions:

- once a number of words have been received,
- randomly using a uniform random number generator. It generates a number between 0 and a high limit. If the generated number is below a threshold it triggers a RTS.

#### Serial parameters

Parameter	Supported values
Baudrate	<ul style="list-style-type: none"> <li>• 19200</li> <li>• 38400</li> <li>• 57600</li> <li>• 115200 (default)</li> <li>• 230400</li> <li>• 460800</li> <li>• 1000000</li> </ul>
Data bits	<ul style="list-style-type: none"> <li>• 8 (default)</li> </ul>
Stop bits	<ul style="list-style-type: none"> <li>• 1 (default)</li> </ul>
Parity	<ul style="list-style-type: none"> <li>• none (default)</li> <li>• even</li> </ul>
Flow control	<ul style="list-style-type: none"> <li>• enabled (default)</li> <li>• disabled</li> </ul>

## AT commands

By default, if a command is not supported, model will respond ERROR. If command arguments are invalid, it will also respond with an error.

Note: AT is omitted.

Command	Support	Comments
(empty)	Complete	
E	Partial	No effect
+UFACTORY	Partial	No effect
O	Partial	Extended Data Mode not supported
+UBTUB=	Partial	No effect
+UBTLE=	Partial	No effect
+UBTLN=	Partial	Notes: <ul style="list-style-type: none"><li>• Command is accepted.</li><li>• Local name is not saved/checkered.</li></ul>
+UBTLN?	Complete	Returns local name
+UMRS=	Complete	Change serial parameters: <ul style="list-style-type: none"><li>• immediately for RX,</li><li>• when send queue has been emptied for TX.</li></ul>
+UMRS?	Complete	Returns serial parameters

## JSON Configuration

Here are all the parameters that can be configured:

```
{  
    "behavior": {  
        "loopback_size": 200  
    },  
    "rts": {  
        "enabled" : true,  
        "buffer_limit": 200,  
        "duration": 200,  
        "random_seed": 12345,  
        "random_high": 10000,  
        "random_threshold": 100  
    }  
}
```

behavior controls how the model reacts:

- loopback\_size is the number of words that need to be received in data mode before sending it back.

rts controls the RTS generation of the model:

- enabled: whether RTS generation is enabled or not.
- buffer\_limit: once this number of words have been received, it triggers a RTS.

- **duration**: duration of the RTS in number of UART cycles.
- **random\_seed**: seed used by the random generator for the random RTS generation.
- **random\_high**: high limit of the random number generator.
- **random\_threshold**: threshold below which the random generated number will trigger a RTS.

The virtual platform is by default simulating only a stand-alone chip with a few default devices which are required to boot a simple example. Device models such as camera, flash or microphones can be connected in order to run full applications.

The devices to be simulated must be specified using the standard runner feature for customizing peripherals.

### 6.3.11 Most usefull commands

For activating instruction traces with debug symbols:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --trace=insn --  
--debug-syms
```

For activating all traces:

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --trace=.*
```

For activating VCD traces (traces are dumped to the file all.vcd):

```
pulp-run --platform=gvsoc --config=gap_rev1 --binary=test prepare run --event=.*
```

### 6.3.12 Remote control

#### Introduction

GVSOC can open a socket and wait for incoming connections so that an external process can interact with it during execution.

The python classes documented in the following section can be used to handle all these interactions.

#### API Reference

**class gv.gvsoc\_control.Proxy(host: str = 'localhost', port: int = 42951)**  
A class used to control GVSOC through the socket proxy

##### Parameters

- **host** – str, a string giving the hostname where the proxy is running
- **port** – int, the port where to connect

##### close()

Close the proxy.

This will free resources and close threads so that simulation can properly exit.

##### event\_add(event: str)

Enable an event.

**Parameters** **event** – A regular expression used to enable events

**event\_remove**(*event: str*)  
Disable a trace.

**Parameters** **event** – A regular expression used to enable events

**quit**(*status: int = 0*)  
Exit simulation.

**Parameters** **status** – Specify the status value.

**register\_exit\_callback**(*callback, \*kargs, \*\*kwargs*)  
Register exit callback

The callback is called when GVSOC exits. If no callback is registered, os.\_exit is called when GVSOC exits.

**Parameters**

- **callback** – The function to be called when GVSOC exits
- **kargs** – Arguments propagated to the callback
- **kwargs** – Arguments propagated to the callback

**run**(*duration: Optional[int] = None*)  
Starts execution.

**Parameters** **duration** – Specify the duration of the execution in picoseconds (will execute forever by default)

**stop()**  
Stop execution.

**trace\_add**(*trace: str*)  
Enable a trace.

**Parameters** **trace** – A regular expression used to enable traces

**trace\_level**(*level: str*)  
Changes the trace level.

**Parameters** **level** – The trace level, can be “error”, “warning”, “info”, “debug” or “trace”

**trace\_remove**(*trace: str*)  
Disable a trace.

**Parameters** **trace** – A regular expression used to disable traces

**wait\_running()**  
Wait until GVSOC is running.

This will block the caller until gvsoc starts execution.

**wait\_stop()**  
Wait until execution stops.

This will block the caller until gvsoc stops execution.

**class gv.gvsoc\_control.Router**(*proxy: gv.gvsoc\_control.Proxy, path: str = '\*\*/chip/soc/axi\_ico'*)  
A class used to inject memory accesses into a router

**Parameters**

- **proxy** – The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **path** – The path to the router in the architecture.

**mem\_read(addr: int, size: int) → bytes**

Inject a memory read.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

**Parameters**

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.

**Returns** bytes, The sequence of bytes read, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_read\_int(addr: int, size: int) → int**

Read an integer.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

**Parameters**

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.

**Returns** int, The integer read.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_write(addr: int, size: int, values: bytes)**

Inject a memory write.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

**Parameters**

- **addr** – The address of the access.
- **size** – The size of the access in bytes.
- **values** – The sequence of bytes to be written, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_write\_int(addr: int, size: int, value: int)**

Write an integer.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

**Parameters**

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.
- **value** – int, The integer to be written.

**Raises** RuntimeError, if the access generates an error in the architecture.

**class gv.gvsoc\_control.Testbench(proxy: gv.gvsoc\_control.Proxy, path: str = '\*\*/testbench/testbench')**

Testbench class.

This class can be instantiated to get access to the testbench.

**Parameters**

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **path** – string, optional, The path to the testbench in the architecture.

**i2s\_get**(*id*: int = 0)

Open an SAI.

Open an SAI and return an object which can be used to interact with it.

**Parameters** **id** – int, optional, The SAI identifier.**Returns** Testbench\_i2s, An object which can be used to access the specified SAI.**uart\_get**(*id*: int = 0)

Open a uart interface.

Open a uart interface and return an object which can be used to interact with it.

**Parameters** **id** – int, optional, The uart interface identifier.**Returns** Testbench\_uart, An object which can be used to access the specified uart interface.**class** gv.gvsoc\_control.Testbench\_i2s(*proxy*: gv.gvsoc\_control.Proxy, *testbench*: gv.gvsoc\_control.Testbench, *id*=0)

Class instantiated for each manipulated SAI.

It can used to interact with the SAI, like injecting streams.

**Parameters**

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **testbench** – int, The testbench object.
- **id** – int, optional, The identifier of the SAI interface.

**clk\_start()**

Start clock.

This can be used when the clock is generated by the testbench to start the generation.

**Raises** RuntimeError, if there is any error while starting the clock.**clk\_stop()**

Stop clock.

This can be used when the clock is generated by the testbench to stop the generation.

**Raises** RuntimeError, if there is any error while stopping the clock.**close()**

Close SAI.

**Raises** RuntimeError, if there is any error while closing.**open**(*word\_size*: int = 16, *sampling\_freq*: int = -1, *nb\_slots*: int = 1, *is\_pdm*: bool = False, *is\_full\_duplex*: bool = False, *is\_ext\_clk*: bool = False, *is\_ext\_ws*: bool = False, *is\_sai0\_clk*: bool = False, *is\_sai0\_ws*: bool = False, *clk\_polarity*: int = 0, *ws\_polarity*: int = 0, *ws\_delay*: int = 1)

Open and configure SAI.

**Parameters**

- **word\_size** – int, optional, Specify the frame word size in bits.

- **sampling\_freq** – int, optional, Specify the sampling frequency. This is used either to generate the clock when it is external or to check that internally generated one is correct.
- **nb\_slots** – int, optional, Number of slots in the frame.
- **is\_pdm** – bool, optional, True if the stream is a PDM stream.
- **is\_full\_duplex** – bool, optional, True if the SAI is used in full duplex mode.
- **is\_ext\_clk** – bool, optional, True is the clock is generated by the testbench.
- **is\_ext\_ws** – bool, optional, True is the word strobe is generated by the testbench.
- **is\_sai0\_clk** – bool, optional, True is the the clock should be taken from SAI0.
- **is\_sai0\_ws** – bool, optional, True is the word strobe should be taken from SAI0.
- **clk\_polarity** – int, optional, Clock polarity, definition is the same as SAI0 specifications.
- **ws\_polarity** – int, optional, Word strobe polarity, definition is the same as SAI0 specifications.
- **ws\_delay** – int, optional, Word strobe delay, definition is the same as SAI0 specifications.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_close(slot: int = 0)**

Close a slot.

**Parameters** **slot** – int, optional, Slot identifier

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_open(slot: int = 0, is\_rx: bool = True, word\_size: int = 16, is\_msb: bool = True, sign\_extend: bool = False, left\_align: bool = False)**

Open and configure a slot.

**Parameters**

- **slot** – int, optional, Slot identifier
- **is\_rx** – bool, optional, True if gap receives the samples.
- **word\_size** – int, optional, Slot width in number of bits.
- **is\_msb** – bool, optional, True if the samples are received or sent with MSB first.
- **sign\_extend** – bool, optional, True if the samples are sign-extended.
- **left\_align** – bool, optional, True if the samples are left aligned.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_rx\_file\_reader(slot: Optional[int] = None, slots: list = [], filetype: str = 'wav', filepath: Optional[str] = None, encoding: str = 'asis', channel: int = 0, width: int = 0)**

Read a stream of samples from a file.

This will open a file and stream it to the SAI so that gap receives the samples. It can be used either in mono-channel mode with the slot parameter or multi-channel mode with the slots parameter. In multi-channel mode, the slots parameters give the list of slots associated to each channel. To allow empty channels, a slot of -1 can be given.

**Parameters**

- **slot** – int, optional, Slot identifier
- **slots** – list, optional, List of slots when using multi-channel mode. slot must be None if this one is not empty.

- **filetype** – string, optional, Describes the type of the file, can be “wav”, “raw”, “bin” or “au”.
- **width** – int, optional, width of the samples, in case the file is in binary format
- **filepath** – string, optional, Path to the file.
- **encoding** – string, optional, Encoding type for binary files, can be: “asis”, “plusminus”
- **channel** – int, optional, If the format supports it, this will get the samples from the specified channel in the input file.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_stop**(*slot: int = 0, stop\_rx: bool = True, stop\_tx: bool = True*)

Stop a slot.

This will stop the streamings (file reader or dumper) configured on the specified slot.

#### Parameters

- **slot** – int, optional, Slot identifier
- **stop\_rx** – bool, optional, Stop the stream sent to gap.
- **stop\_tx** – bool, optional, Stop the stream received from gap.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_tx\_file\_dumper**(*slot: Optional[int] = None, slots: list = [], filetype: str = 'wav', filepath: Optional[str] = None, encoding: str = 'asis', channel: int = 0, width: int = 0*)

Write a stream of samples to a file.

This will open a file and write to it all the samples received from gap. It can be used either in mono-channel mode with the slot parameter or multi-channel mode with the slots parameter. In multi-channel mode, the slots parameters give the list of slots associated to each channel. To allow empty channels, a slot of -1 can be given. A slot can be given several times in order to push the samples to several channels.

#### Parameters

- **slot** – int, optional, Slot identifier
- **slots** – list, optional, List of slots when using multi-channel mode. slot must be None if this one is not empty.
- **filetype** – string, optional, Describes the type of the file, can be “wav”, “raw”, “bin” or “au”.
- **encoding** – string, optional, Encoding type for binary files, can be: “asis”, “plusminus”
- **width** – int, optional, width of the samples, in case the file is in binary format
- **filepath** – string, optional, Path to the file.
- **channel** – int, optional, If the format supports it, this will dump the samples to the specified channel in the output file.

**Raises** RuntimeError, if there is any invalid parameter.

**class gv.gvsoc\_control.Testbench\_uart**(*proxy: gv.gvsoc\_control.Proxy, testbench: gv.gvsoc\_control.Testbench, id=0*)

Class instantiated for each manipulated uart interface.

It can be used to interact with the uart interface, like injecting streams.

#### Parameters

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **testbench** – int, The testbench object.
- **id** – int, optional, The identifier of the uart interface.

**close()**

Close the uart interface.

**Raises** RuntimeError, if there is any error while closing.

**open(baudrate: int, word\_size: int = 8, stop\_bits: int = 1, parity\_mode: bool = False, ctrl\_flow: bool = True, is\_usart: bool = False, usart\_polarity: int = 0, usart\_phase: int = 0)**

Open and configure a uart interface.

**Parameters**

- **baudrate** – int, Specify the uart baudrate in bps
- **word\_size** – int, optional, Specify the size in bits of the uart bytes.
- **stop\_bits** – int, optional, Specify the number of stop bits.
- **parity\_mode** – bool, optional, True if parity is enabled.
- **ctrl\_flow** – bool, optional, True if control flow is enabled.
- **is\_usart** – bool, optional, True if uart is in usart mode.
- **usart\_polarity** – int, optional, Usart polarity.
- **usart\_phase** – int, optional, Usart phase.

**Raises** RuntimeError, if there is any invalid parameter.

**rx(size=None)**

Read data from the uart.

Once reception on the uart is enabled, the received bytes are pushed to a fifo. This method can be called to pop received bytes from the FIFO.

**Parameters** **size** – int, The number of bytes to be read. If it is None, it returns the bytes which has already been received.

**Returns** bytes, The sequence of bytes received, in little endian byte ordering.

**Raises** RuntimeError, If the access generates an error in the architecture.

**rx\_attach\_callback(callback, \*kargs, \*\*kwargs)**

Attach callback for receiving bytes from the uart.

All bytes received from the uart now triggers the execution of the specified callback. This must be called only when uart reception is disabled. The callback will be called asynchronously by a different thread and so special care must be taken to access shared variables using locks. Also the proxy can not be used from the callback.

**Parameters**

- **callback** – The function to be called when bytes are received from the uart. First parameters will contain number of bytes and received, and second one will be the bytes received.
- **kargs** – Arguments propagated to the callback
- **kwargs** – Arguments propagated to the callback

**Returns** bytes, The sequence of bytes received, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_detach\_callback()**

Detach a callback.

The callback previously attached won't be called anymore. This must be called only when uart reception is disabled.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_disable()**

Disable receiving bytes from the uart.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_enable()**

Enable receiving bytes from the uart.

Any byte received from the uart either triggers the callback execution if it has been registered, or is pushed to a FIFO which can read.

**Raises** RuntimeError, if the access generates an error in the architecture.

**tx(*values: bytes*)**

Send data to the uart.

This enqueues an array of bytes to be transmitted. If previous transfers are not finished, these bytes will be transferred after.

**Parameters** **values** – bytes, The sequence of bytes to be sent, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

## 6.4 Profiler

### 6.4.1 Introduction

Profiler is a part of GWT GAP SDK and used with GVSOC, GWT Full System SoC Simulator. Profiler gives a visual view of what is happening inside the chip and allows to control the simulator through a graphic interface. Profiler is an extremely useful tool for developing and debugging applications on GAP processors.

Profiler allows you to do many things, from simple GAP core signals display, to fine stall signal analysis, to applications optimization. Following is a list of the most important features:

- **GAP signals display:** It displays the internal signals of the GAP processor issued from the Gvsoc simulation
- **GAP cores traces fine analysis:** It allows to view which function is executed at a given timestamp on any core of GAP chip and on the fabric controller
- **Stall signals analysis:** It allows you to view all the stall signals of the GAP cores and fabric controller.
- **Hotspots Analysis:** It allows to see Hotspots information such as Hotspot function, Hotspot line number, calls number, total execution time, source code in different dock windows.
- **Display monitoring:** It allows to select signals by groups, zooming (button, mouse wheel, double click), time intervals measurement, dock windows management.

## 6.4.2 Getting Started

Follow this guide to:

- Install the profiler and its dependencies
- run fork application

### Install dependencies

You'll need to install some dependencies

- GAP SDK : see [https://github.com/GreenWaves-Technologies/gap\\_sdk](https://github.com/GreenWaves-Technologies/gap_sdk)
- Qt >= 5.13: see <https://doc.qt.io/qt-5/gettingstarted.html>

### Requirements on Ubuntu 18

```
sudo apt install qt5-default libqt5charts5-dev
```

### How to build

```
# First initialize gap_sdk
cd ~/gap_sdk
# source the initialization file and choose GAP9_v2
source configs/gap9_v2.sh
#Now build the profiler
make profiler_v2
```

## 6.4.3 Running the Profiler

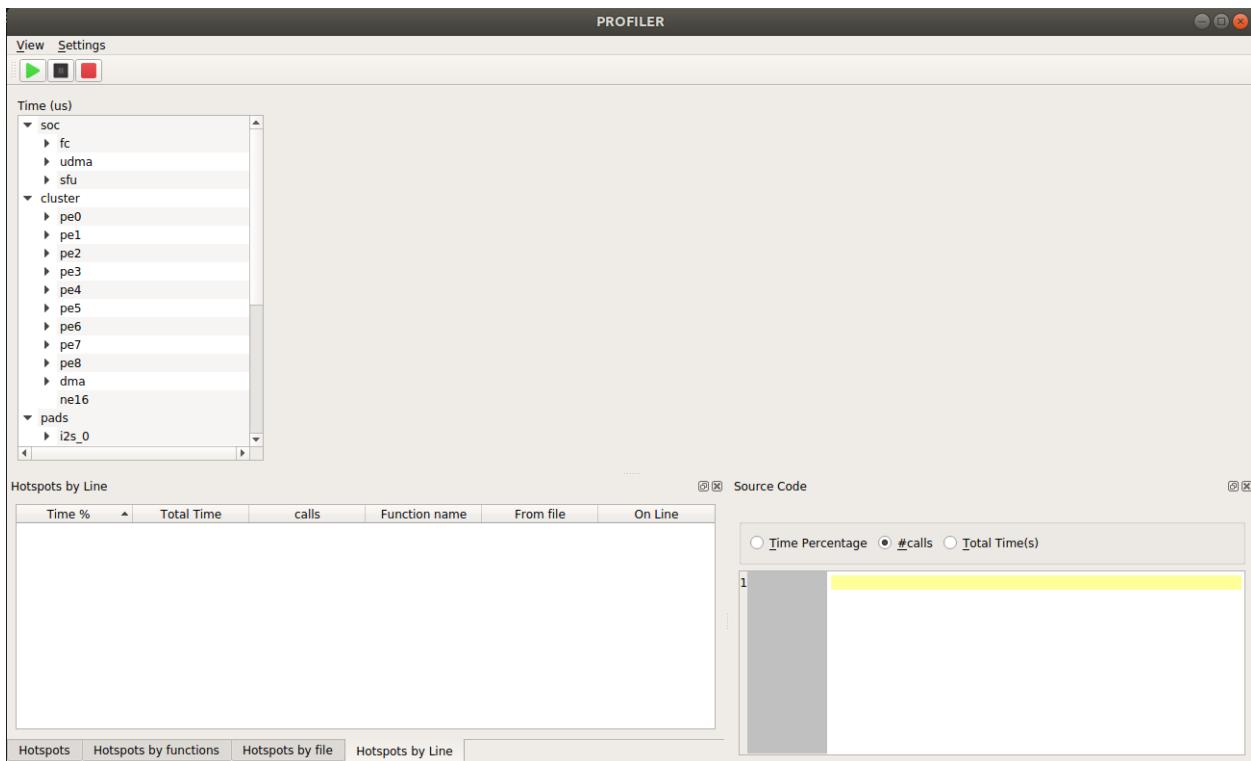
To run the profiler, first go to an example. Let's try the fork example:

```
>cd ~/gap_sdk/tests/pmsis_tests/quick/cluster/fork
```

Then build the example and run the profiler:

```
>make all profile platform=gvsoc
```

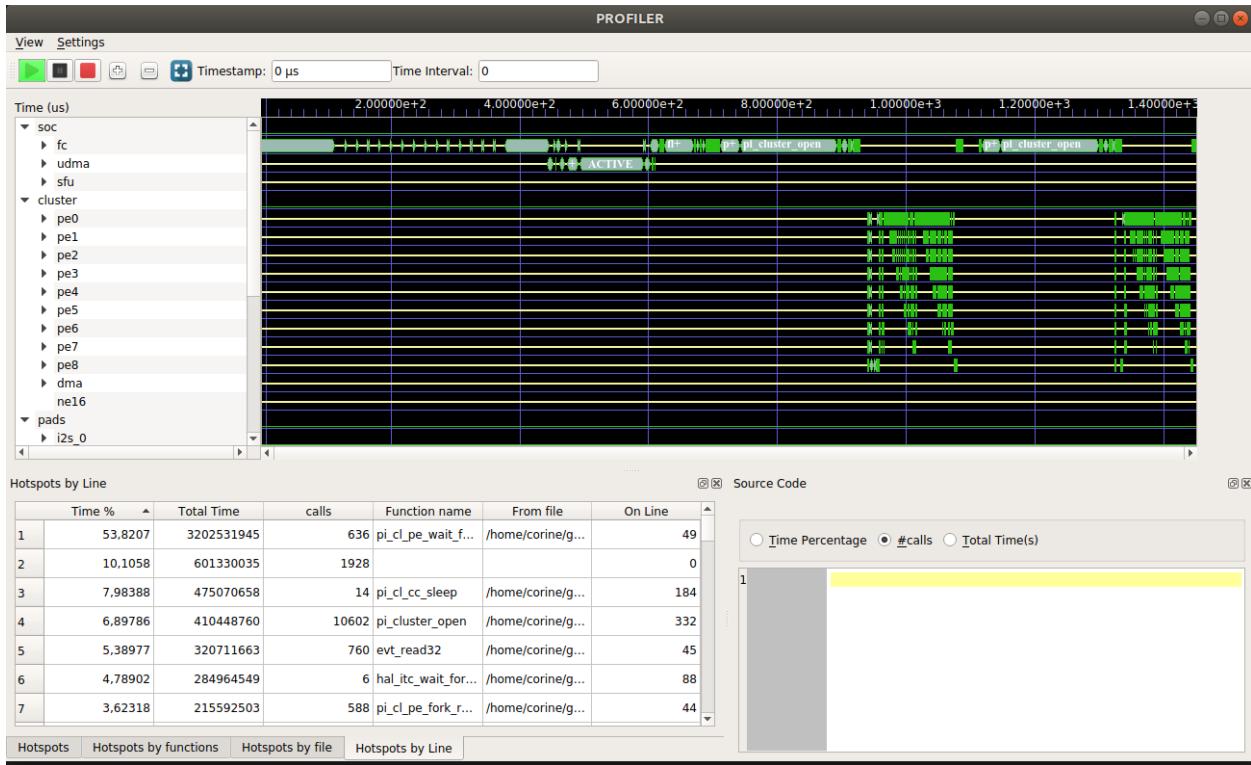
This launches the profiler graphics application and the profiler window is displayed with an empty timeline window that will later on display all the GAP9 signals transferred from Gvsoc.

**Fig 1: Profiler Window**

#### 6.4.4 Starting the Gvsoc Simulation

For starting the Gvsoc simulation, just click on the green arrow button on the left of the actions bar. The simulation can be paused by clicking on the black square button. It can be continued by re-clicking on the green arrow.

The simulation can be cancelled by clicking on the red square button. It can be started again by clicking on the green arrow.



**Fig2: Profiler Actions Bar**

During the simulation, the GAP9 signals are displayed on the Timeline Window as soon as they are transmitted by the Gvsoc simulator. Two dock windows are simultaneously displayed:

- The Hotspots dock window with its different tab views (by time, by function, by file, by line)
- The Source code dock window, which is empty at this stage

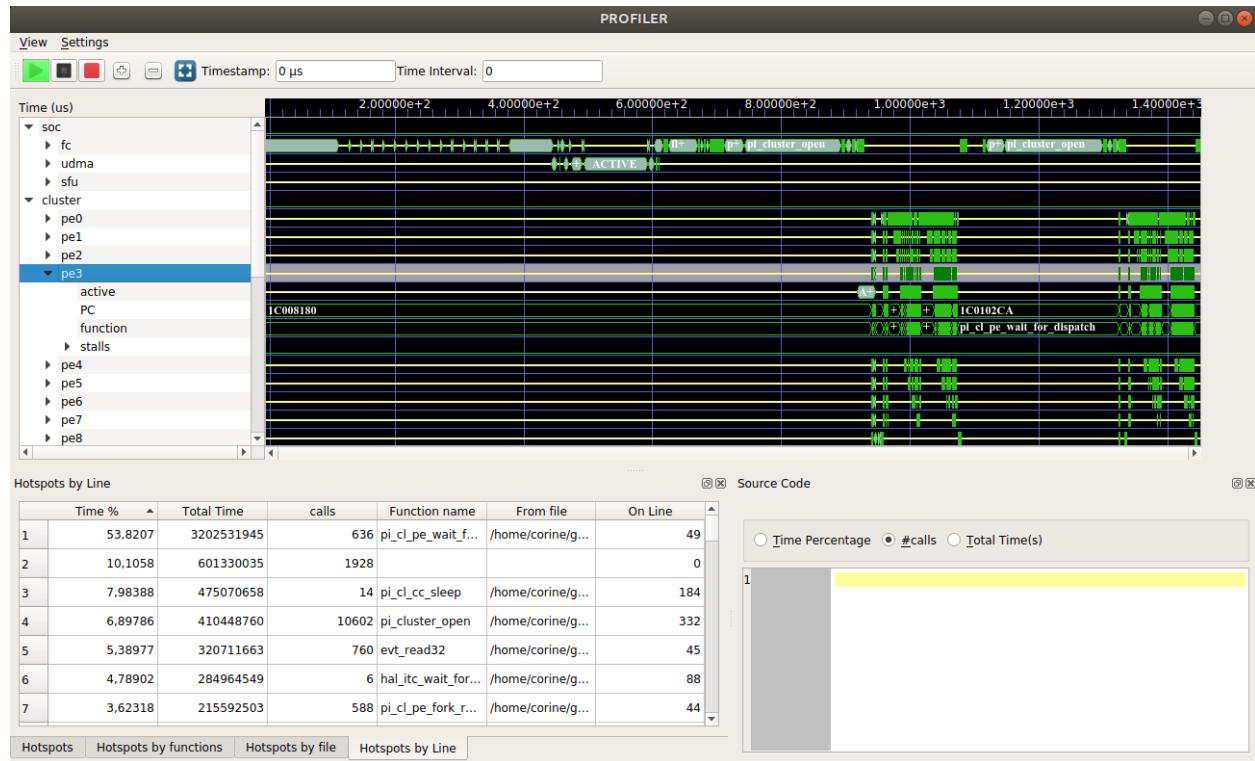
#### 6.4.5 Timeline window Commands

While the Timeline Window is being filled up with the signal elements, many functionalities can be achieved with the mouse.

##### Selecting a Signal in the Signals Tree

A **Left Mouse Click** on a signal name in the left Signals Tree Window automatically **highlights** the signal on the Timeline Window with a grey background.

As an example, in the following image, the “Pe3” signal has been selected by a left mouse click on its name.



**Fig 3: Selecting a signal in the Signals Tree Window**

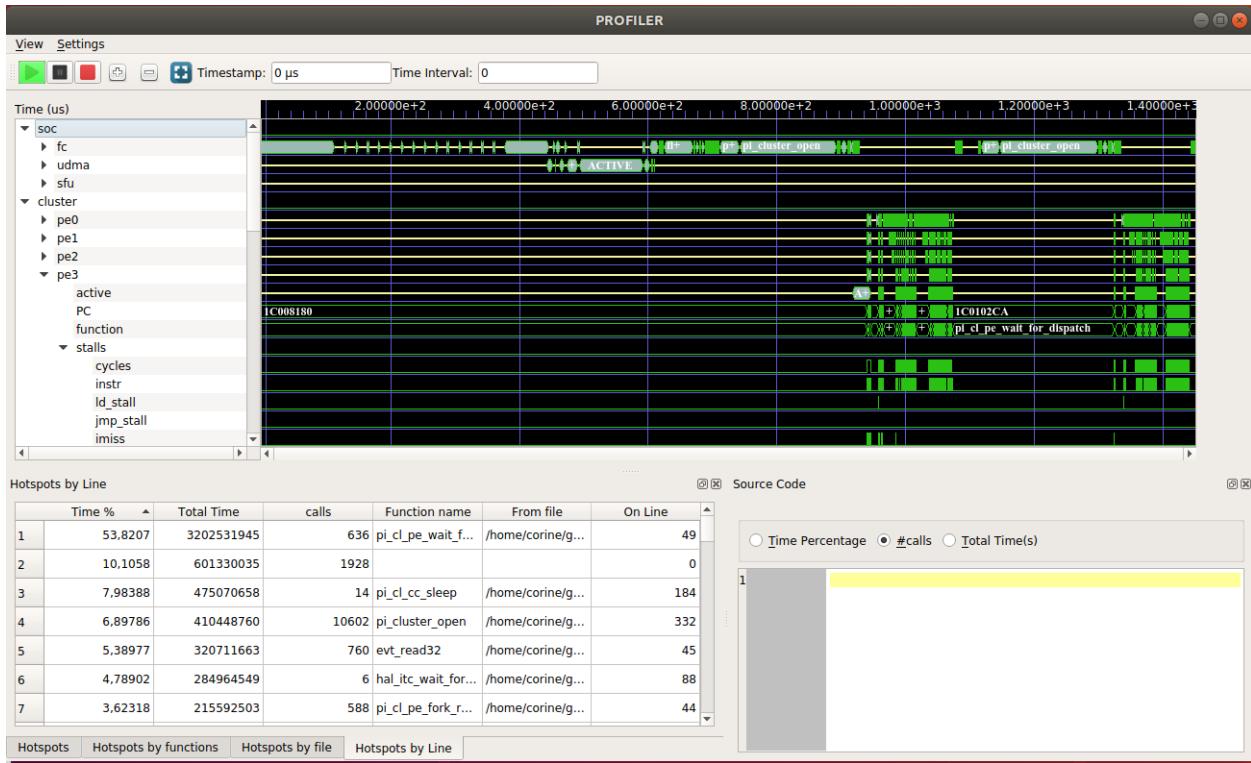
### Adding/removing Signals

On the right of the Timeline Window, where the signals are being displayed during the simulation of an application, there's a tree containing all the signals that have been previously selected through the **Settings→ Gvsoc Settings** command. The signals are hierarchically classified.

Each node of the tree can be **expanded** till we reach full expansion by **left clicking** with the mouse on the arrows of the tree. When a node is expanded, the corresponded new signals are being added in the timeline view.

Each node can be **compressed** by **left clicking** on the same arrow. Corresponding signals will then be removed from the Timeline.

If the simulation is ongoing or has been fully executed, the display of the signals in the Timeline Window is automatically in sync with the signals tree view. On the following image, the user chose to display the stall signals of the GAP core Pe3.



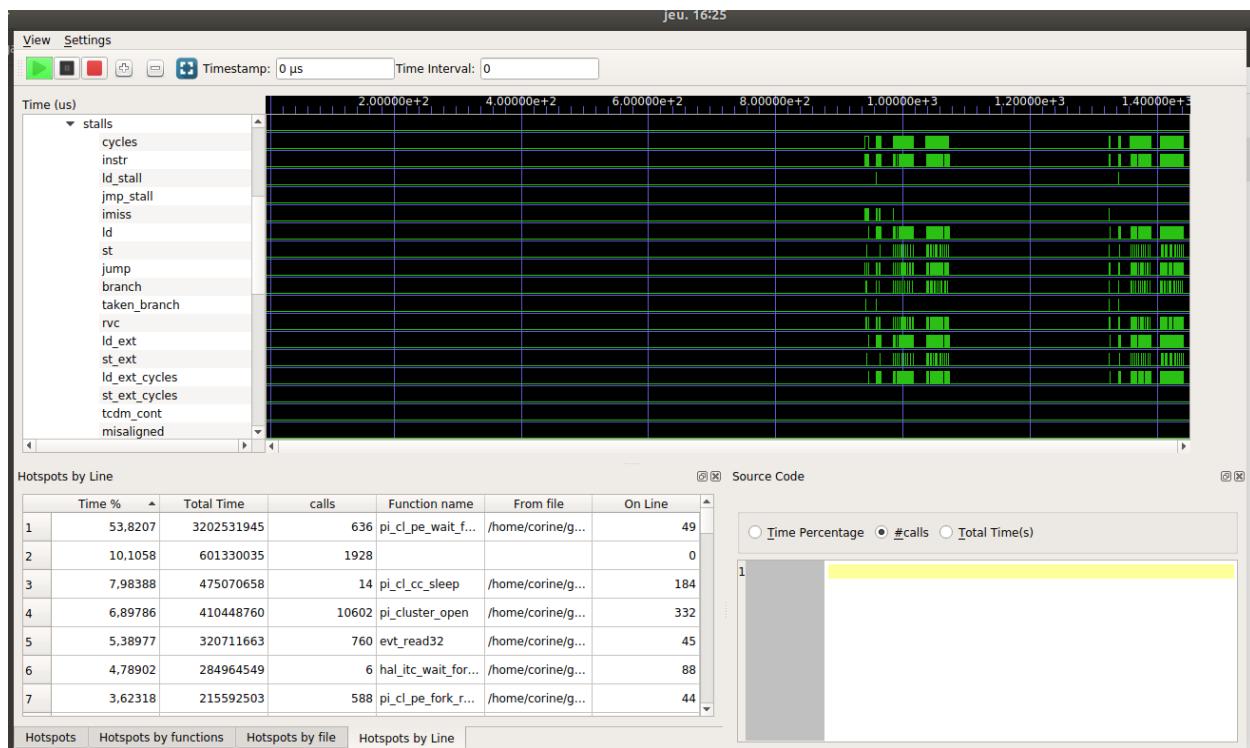
**Fig 4: Expanding the Signals Tree to display stall signals of core #3**

### Scrolling the Timeline window vertically

The signals tree vertical scroll bar synchronously pilotes the scrolling of the two following windows:

- The signals tree window (on the left)
- The Timeline Window (on the right)

In the following picture, the user has scrolled down the signal tree to watch all stall signals of core #3.



**Fig 5: Scrolling down the Signals Tree to see stall signals**

### Zooming in & out

**Zooming in & out** the Timeline view is possible by **pushing the Ctrl key** of your Keyboard while **scrolling the wheel** of your mouse respectively forward or backward.

In the following picture, the user has zoomed in to see the details of the different stall signals

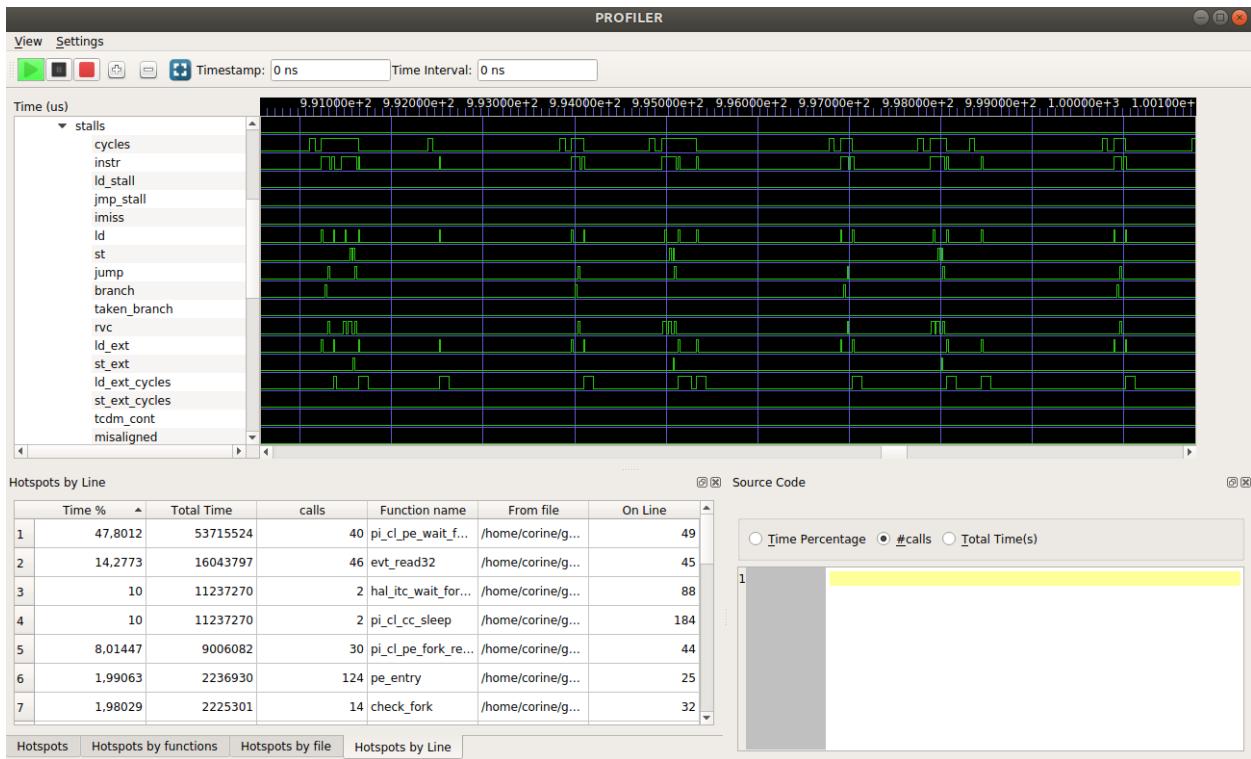


Fig 6: Zooming in & out wit Ctrl + mouse wheel

### Scrolling the Timeline Window horizontally

The Timeline Window can be scrolled horizontally with the scrollbar underneath it in three different ways:

- by **scrolling the mouse wheel** forward or backward with the mouse within the timeline window.
- by **moving the scrollbar cursor** underneath the timeline window
- by **clicking on the right or left arrows** of the timeline window scrollbar

Figure 7 shows the timeline before scrolling and figure 7 bis after scrolling to the right

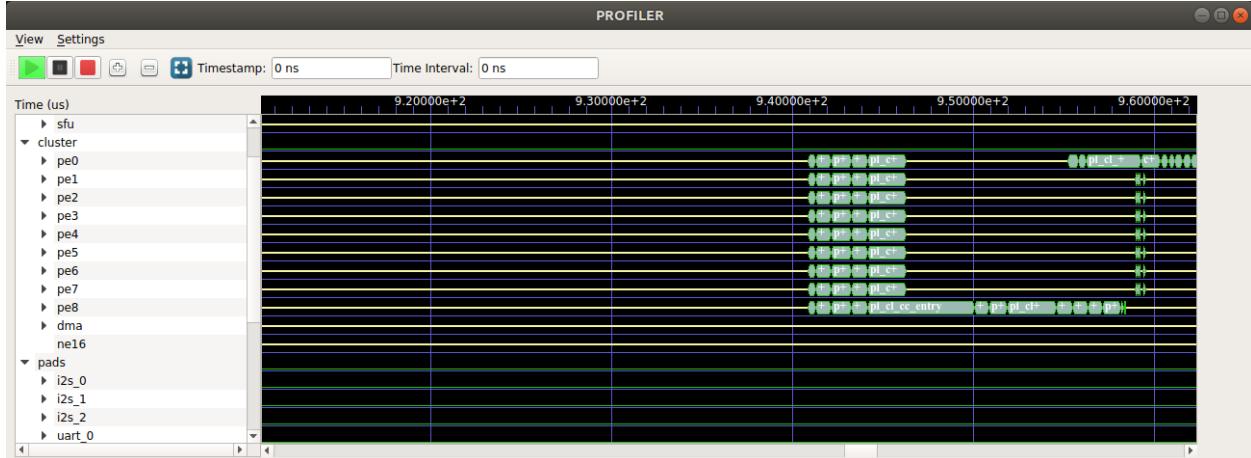
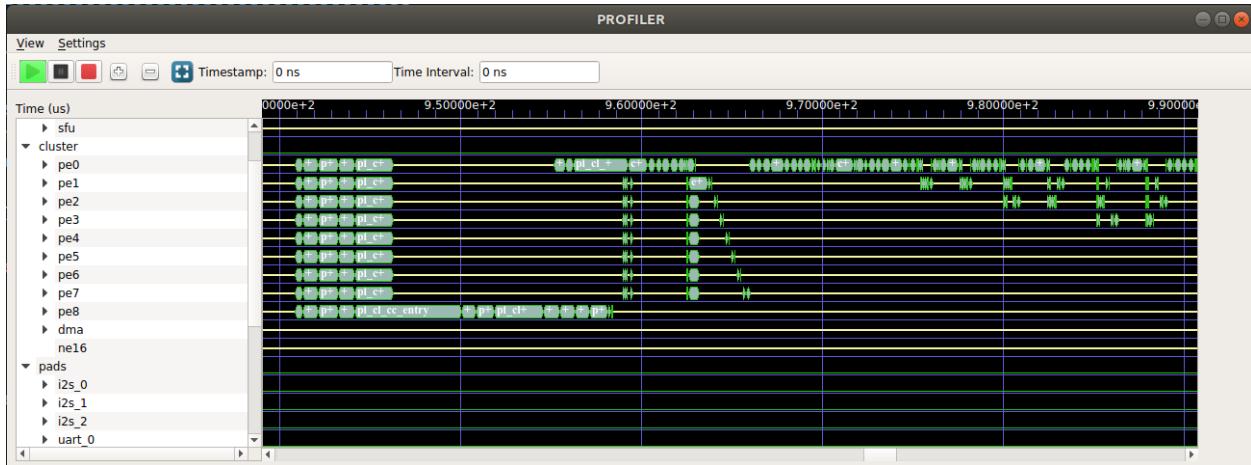


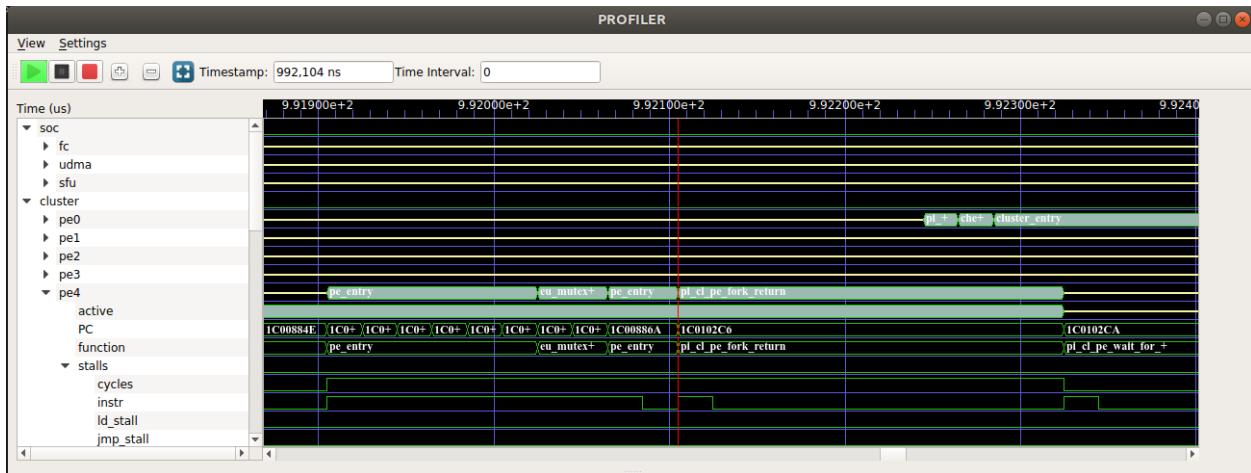
Fig 7: Timeline Window Before Scrolling



**Fig 7 bis: Timeline Window after Scrolling to the right**

### Selecting a Timestamp on the Timeline Window

A **Left Mouse Click** in the Timeline Window allows to **select a timestamp** which is materialized by a red bar in the Timeline Window. The timestamp value is displayed in the **Timestamp Text field** in the actions bar above the Timeline Window.

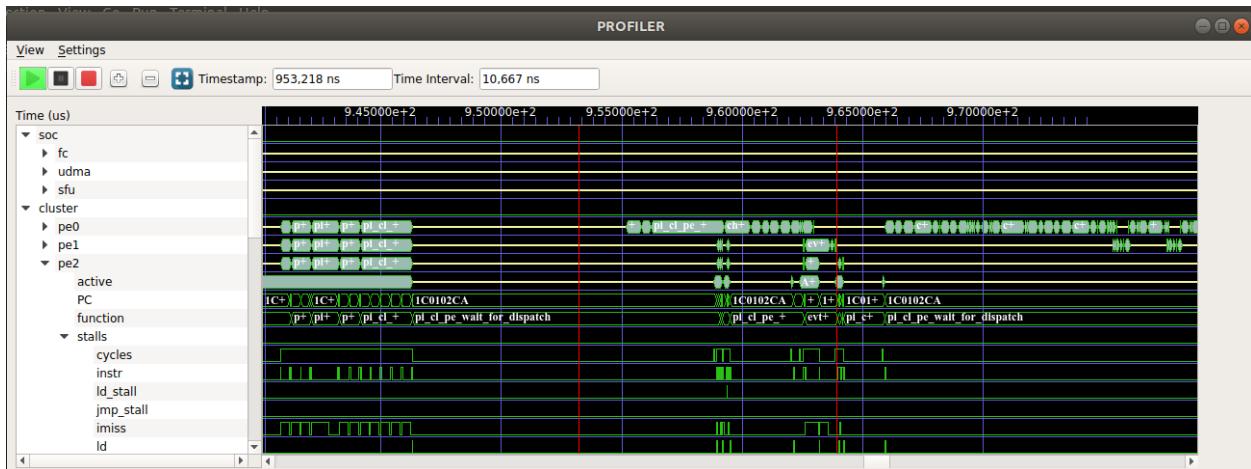


**Fig 8: Selecting a timestamp on the Timeline Window**

**Right Clicking** with the mouse in the Timeline Window **deletes the timestamp** bar in the Graphics window.

### Selecting a Time Interval on the Timeline Window

A **Right Mouse Press + Mouse Move & Release** in the Timeline Window allows to **select a time interval** which is materialized by two red bars in the Timeline Window. The first bar timestamp value is displayed in the **Timestamp Text field** in the actions bar above the Timeline Window and the time interval value is displayed in the **time interval** field.



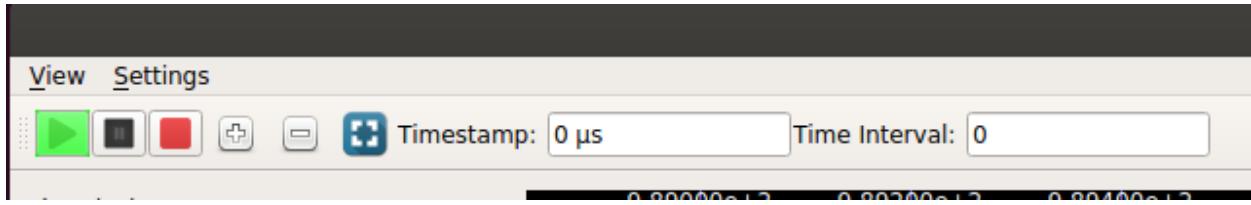
**Fig 9: Selecting a time interval on the Timeline Window**

**Right Clicking** with the mouse in the Timeline Window outside the time interval **deletes the time interval** in the Timeline window.

Note that the hotspots information is now relevant to this time interval only.

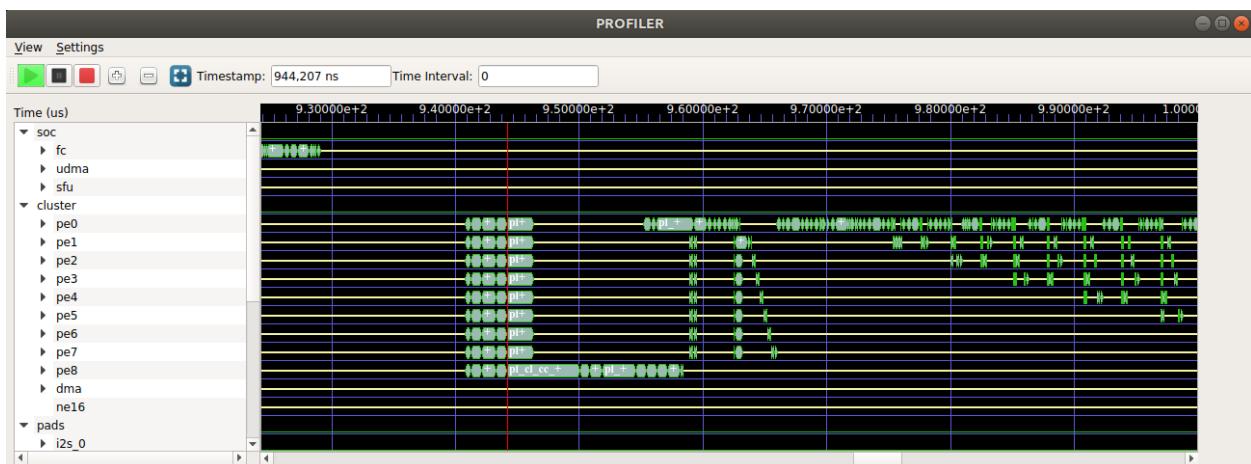
### Zooming in & out on timestamp

**Zooming in & out** the Timeline view around a timestamp previously selected is possible through the **+ and - buttons** of the timeline commands bar as shown in the following figure:



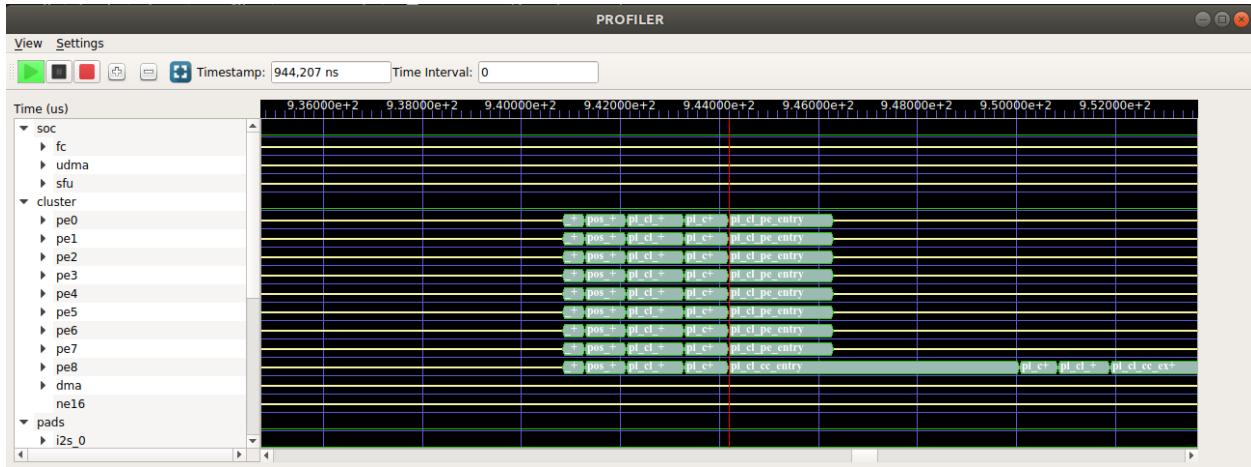
**Fig 10: + and - buttons of the commands bar**

In the following picture, the user has **selected a timestamp by clicking on the left mouse button**:



**Fig 11: Selection of a timestamp**

In the following picture, the user has zoomed in around this timestamp materialized by the vertical red line the details of the different signals by **clicking on the + button**. The timeline is automatically centered on the timestamp.



**Fig 12: Zooming in around the selected timestamp**

### 6.4.6 Other Profiler Windows

Around the central Profiler Window, which comprises the Signals Tree and the Timeline windows, other dock windows can be displayed. Whether they show up or not can be monitored by the user through the View Menu.

This menu displays the name of all those optional windows, along with a button that can be checked or not by the user, depending on whether he/she wants this window to show up or not.

- Source code
- Hotspots (by time)
- Hotspots by functions
- Hotspots by file
- Hotspots by line

Additionally, each of those dock windows is independent and can be:

- Moved in some location around the central Profiler Window by left clicking on the window bar and moving the mouse
- Moved outside the profiler window by left clicking on the window bar and moving the mouse
- Expanded by left clicking on its edges
- Closed by clicking on the “cross button” on its upper right corner
- Tabbed together as it is by default

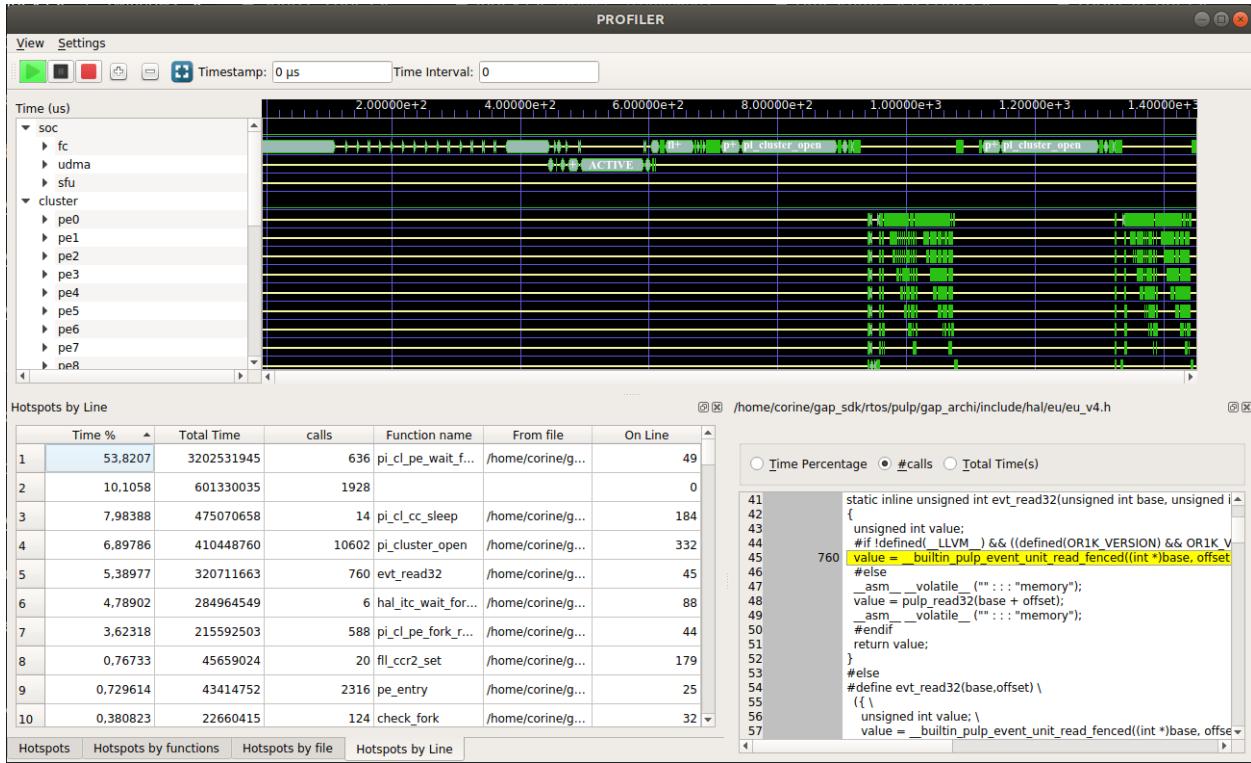


Fig 13: Dock Windows Placement within the profiler window

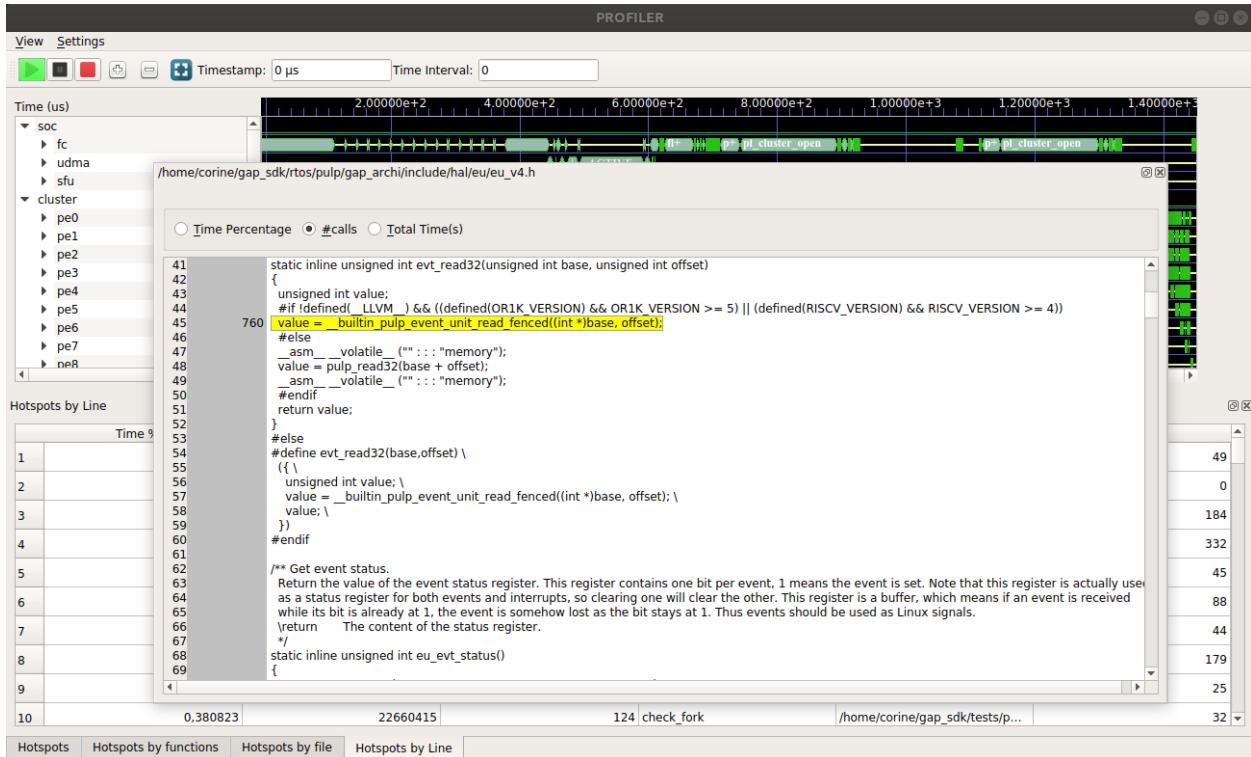


Fig 14: Source code Dock Window independent placement above the Profiler window

## The Hotspots Window

The Hotspots window lists in a table all the hotspots concerning the time interval displayes within the Timeline window. For seeing all hotspots, the user must display the full simulation interval by clicking on the blue “Display All” button.

The hotspots are sorted in this view by their percentage of time in decreasing order. The hotspot information is displayed in different columns:

- **Time %:** the percentage of time that has been spent on this particular line
- **Total time:** the total time that has been spent on this particular line
- **Calls:** the number of times the hotspot function has been cancelled
- **Function name:** the name of the function in which the hotspot occurs
- **From File:** the name of the file where the hotspot occurs
- **On Line:** the line number of the hotspot
- **PC Nb**

Fig 16 shows the Hotspots window.

Hotspots							
	Time %	Total Time	calls	Function name	From file	On Line	PC Nb
1	53.8207	3202531945	318	pi_cl_pe_wait_for_dispatch	/home/corine/gap_sdk/rtos/pu...	49	1c0102ca
2	7.98388	475070658	7	pi_cl_cc_sleep	/home/corine/gap_sdk/rtos/pu...	184	1c010184
3	5.78421	344182124	22			0	1a001540
4	4.78902	284964549	3	hal_itc_wait_for_interrupt	/home/corine/gap_sdk/rtos/pu...	88	1c00bc62
5	3.62318	215592503	294	pi_cl_pe_fork_return	/home/corine/gap_sdk/rtos/pu...	44	1c0102c6
6	3.37344	200732321	294	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c00884e
7	2.00888	119536111	2001	pi_cluster_open	/home/corine/gap_sdk/rtos/pu...	332	1c008ea2
8	1.33963	79712759	2000	pi_cluster_open	/home/corine/gap_sdk/rtos/pu...	332	1c008e9a
9	1.33957	79709220	2000	pi_cluster_open	/home/corine/gap_sdk/rtos/pu...	332	1c008ea0
10	0.756061	44988433	4	fli_ccr2_set	/home/corine/gap_sdk/rtos/pu...	179	1c00bb26
11	0.677138	40292232	2001	pi_cluster_open	/home/corine/gap_sdk/rtos/pu...	332	1c008e9e
12	0.669624	39845117	2000	pi_cluster_open	/home/corine/gap_sdk/rtos/pu...	332	1c008e9c
13	0.412562	24548972	1			0	1a001378
14	0.402793	23967671	18	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c0088ea
15	0.323603	19255571	16	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c0089be
16	0.323581	19254279	16	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c008a2e
17	0.317013	18863444	16	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c0089f4
18	0.316994	18862356	16	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c008938
19	0.225353	13409328	45			0	1a001528
20	0.22102	13151530	44	pos_init_cluster_data	/home/corine/gap_sdk/rtos/pu...	332	1c008f2a
21	0.209983	12494796	296	pi_cl_pe_fork_entry	/home/corine/gap_sdk/rtos/pu...	60	1c0102de
22	0.196809	11710868	294	pe_entry	/home/corine/gap_sdk/tests/p...	27	1c00886a
23	0.185272	11024391	18	pi_cl_entry_stub	/home/corine/gap_sdk/rtos/pu...	58	1c00bdbe
24	0.16487	9810366	294	evt_read32	/home/corine/gap_sdk/rtos/pu...	45	1c00884a
25	0.160666	9560202	310	pi_cl_pe_wait_for_dispatch	/home/corine/gap_sdk/rtos/pu...	50	1c0102ce
26	0.112917	6719003	310	pi_cl_pe_wait_for_dispatch	/home/corine/gap_sdk/rtos/pu...	54	1c0102d6
27	0.110767	6591040	23			0	1a001520

Fig 16: Hotspots Window

## The Source Code Window

The source code window displays the source code corresponding to the function hotspot that has been selected by **left clicking** in the hotspot window. The following image displays the source code of the **check\_fork** function that corresponds to hotspot #10 within the file `~/gap_sdk/tests/pmsis_tests/quick/cluster/test.c`. In the left grey column of the window, some hotspot information can be alternatively displayed:

- **The time Percentage:** the percentage of time that has been spent on this particular line
- **The calls nb:** the number of times the function has been cancelled
- **The total time:** the total time that has been spent on this particular line

A commands bar allows to check either of this information. Additionally, the hotspot line is highlighted in yellow.

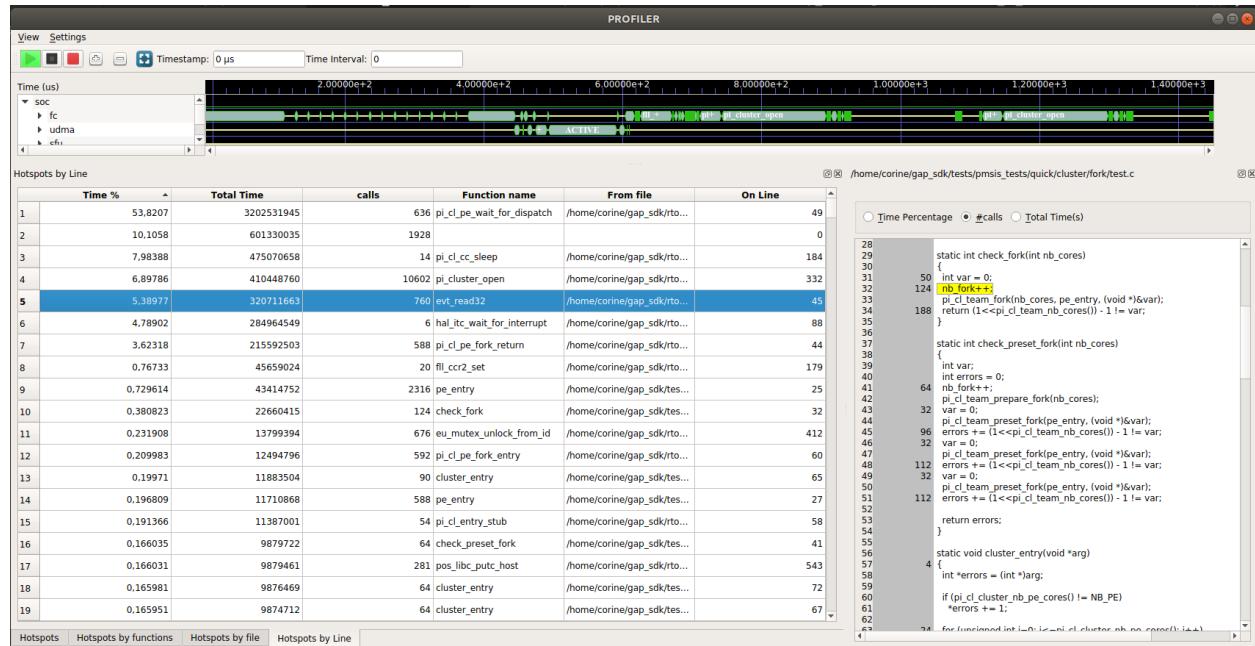


Fig 15: Source Code Window

## The Hotspots by Functions Window

The Hotspots by Functions window lists in a table all the hotspots concerning the time interval displayed within the Timeline window. For seeing all hotspots, the user must display the full simulation interval by clicking on the blue “Display All” button.

In this window, the hotspots are grouped by function name. They can be additionally sorted in alphabetic (decreasing or increasing order) by clicking in the Title field of the “Function Name” column. The hotspot information is displayed in different columns:

- **Time %:** the percentage of time that has been spent on this particular line
- **Total time:** the total time that has been spent on this particular line
- **Function name:** the name of the function in which the hotspot occurs
- **From File:** the name of the file where the hotspot occurs
- **On Line:** the line number of the hotspot

Fig 17 shows this Hotspots by Functions window.

Hotspots by functions					
	Time %	Total Time	Function name	From file	On Line
1	54,1974	3224948478	pi_cl_pe_wait_for_dispatch	/home/corine/gap_sdk/rt...	49
2	10,1058	601330035			0
3	8,00452	476299021	pi_cl_cc_sleep	/home/corine/gap_sdk/rt...	183
4	6,29324	374471352	pi_cluster_open	/home/corine/gap_sdk/rt...	332
5	5,38977	320711663	evt_read32	/home/corine/gap_sdk/rt...	45
6	4,78902	284964549	hal_itc_wait_for_interrupt	/home/corine/gap_sdk/rt...	88
7	3,62318	215592503	pi_cl_pe_fork_return	/home/corine/gap_sdk/rt...	44
8	0,926423	55125620	pe_entry	/home/corine/gap_sdk/t...	25
9	0,76733	45659024	fil_ccr2_set	/home/corine/gap_sdk/rt...	179
10	0,577185	34344643	cluster_entry	/home/corine/gap_sdk/t...	57
11	0,46793	27843601	check_fork	/home/corine/gap_sdk/t...	31
12	0,335211	19946289	pi_cl_pe_entry	/home/corine/gap_sdk/rt...	27
13	0,308829	18376462	pi_cl_pe_fork_entry	/home/corine/gap_sdk/rt...	59
14	0,308086	18332257	pos_init_cluster_data	/home/corine/gap_sdk/rt...	332
15	0,305347	18169299	check_preset_fork	/home/corine/gap_sdk/t...	41
16	0,231908	13799394	eu_mutex_unlock_from_id	/home/corine/gap_sdk/rt...	412
17	0,203554	12112221	pi_cl_entry_stub	/home/corine/gap_sdk/rt...	58
18	0,1698	10103715	pi_cl_entry	/home/corine/gap_sdk/rt...	28
19	0,153478	9132499	pos_init_entry	/home/corine/gap_sdk/rt...	30
20	0,120125	7147864	pi_cl_cc_entry	/home/corine/gap_sdk/rt...	40
21	0,10507	6252045	pi_cl_pe_team_stub	/home/corine/gap_sdk/rt...	73
22	0,0958544	5703697	pi_cl_team_nb_cores	/home/corine/gap_sdk/rt...	67

**Fig 17: Hotspots Window**

### The Hotspots by File Window

The Hotspots by File window lists in a table all the hotspots concerning the time interval displayed within the Timeline window. For seeing all hotspots, the user must display the full simulation interval by clicking on the blue “Display All” button.

The hotspots are grouped in this view by file. Additionally, they can be sorted out alphabetically by their file name by clicking on the right of the title of the “From File” column. The hotspot information is displayed in different columns:

- **Time %:** the percentage of time that has been spent on this particular line
- **Total time:** the total time that has been spent on this particular line
- **From File:** the name of the file where the hotspot occurs

Fig 18 shows the Hotspots by File window.

Hotspots by file			
	Time %	Total Time	From file
1	18,2928	18446743994372273122	
2	17,804	18446743996492201967	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/dr...
3	7,16111	18446744042651218042	/home/corine/gap_sdk/tests/pmsis_tests/quick/clu...
4	6,50705	18446744045487937065	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/dr...
5	6,45641	18446744045707590527	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/dr...
6	5,01554	18446744051956727298	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/a...
7	4,18049	18446744055578440661	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/ke...
8	3,51181	18446744058478556151	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/a...
9	3,40654	18446744058935108358	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/c...
10	2,50776	18446744062833195903	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/ke...
11	2,50698	18446744062836573755	/home/corine/gap_sdk/rtos/pulp/pulpos-2/lib/libc/...
12	2,49671	18446744062881102192	/home/corine/gap_sdk/rtos/pulp/gap_archi/include...
13	2,40783	18446744063266602016	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/ke...
14	1,87277	18446744065587211901	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/ke...
15	1,86174	18446744065635019194	/home/corine/gap_sdk/rtos/pulp/gap_archi/include...
16	1,73889	18446744066167855562	/home/corine/gap_sdk/rtos/pulp/pulpos-2/include/...
17	1,40456	18446744067617849553	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/t...
18	1,16942	18446744068637690705	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/i...
19	1,03682	18446744069212763578	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/ke...
20	0,800878	18446744070236077288	/home/corine/gap_sdk/rtos/pulp/pulpos-2/include/...
21	0,702892	18446744070661052626	/home/corine/gap_sdk/rtos/pulp/gap_archi/include...
22	0,702359	18446744070663363350	/home/corine/gap_sdk/rtos/pulp/pulpos-2/kernel/ti...
23	0,702318	18446744070663541475	/home/corine/gap_sdk/rtos/pulp/pulpos-2_gap9/in...
24	0,668985	18446744070808109514	/home/corine/gap_sdk/rtos/pulp/pulpos-2/include/...

**Fig 18: Hotspots Window**

### The Hotspots by Line Window

The Hotspots by Line window lists in a table all the hotspots concerning the time interval displayed within the Timeline window. For seeing all hotspots, the user must display the full simulation interval by clicking on the blue “Display All” button.

In this window, the hotspots are grouped by line number. They can be sorted by increasing or decreasing line number by clicking on the right of the title of the “On Line” column. The hotspot information is displayed in different columns:

- **Time %:** the percentage of time that has been spent on this particular line
- **Total time:** the total time that has been spent on this particular line
- **Calls:** the number of times the hotspot function has been cancelled
- **Function name:** the name of the function in which the hotspot occurs
- **From File:** the name of the file where the hotspot occurs
- **On Line:** the line number of the hotspot

Fig 19 shows the Hotspots by Line window.

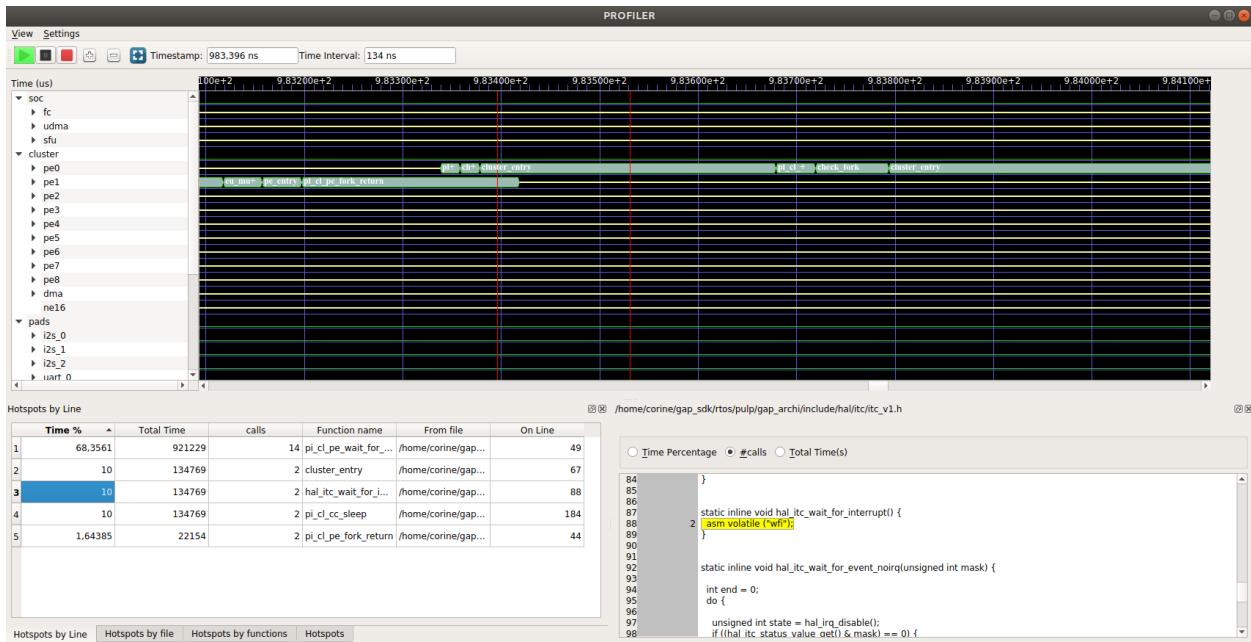
Hotspots by Line						
	Time %	Total Time	calls	Function name	From file	On Line
3	3,54507	18446744058334...	107	pos_pmu_change...	/home/corine/gap...	97
4	2,84296	18446744061379...	86	pos_allocs_init	/home/corine/gap...	142
5	2,4401	18446744063126...	282	puts	/home/corine/gap...	543
6	2,0978	18446744064611...	2316	pe_entry	/home/corine/gap...	25
7	1,60511	18446744066748...	50	pos_irq_get_fc_ve...	/home/corine/gap...	127
8	1,23685	18446744068345...	152	pos_alloc_account	/home/corine/gap...	110
9	0,8294	18446744070112...	760	evt_read32	/home/corine/gap...	45
10	0,699421	18446744070676...	676	eu_mutex_unlock...	/home/corine/gap...	412
11	0,668118	18446744070811...	57	_start	/home/corine/gap...	150
12	0,602092	18446744071098...	20	pos_alloc	/home/corine/gap...	244
13	0,602015	18446744071098...	30	pos_pmu_scu_ha...	/home/corine/gap...	63
14	0,599605	18446744071109...	54	pi_cl_entry_stub	/home/corine/gap...	58
15	0,568525	18446744071243...	52	pi_cl_entry	/home/corine/gap...	37
16	0,535195	18446744071388...	20	pos_fll_init	/home/corine/gap...	126
17	0,535188	18446744071388...	18	pi_cl_l1_malloc	/home/corine/gap...	162
18	0,535168	18446744071388...	48	pi_cl_pe_entry	/home/corine/gap...	33
19	0,535168	18446744071388...	48	pi_cl_pe_entry	/home/corine/gap...	36
20	0,535164	18446744071388...	27	pos_pmu_scu_ha...	/home/corine/gap...	25
21	0,53493	18446744071389...	20	pos_fll_get_mult_...	/home/corine/gap...	51

**Fig 19: Hotspots Window**

### Hotspots by Time Interval

There is a possibility to display the Hotspots related only to a specific time Interval. For doing this, one must select a Time Interval in the Timeline Window by **pushing Right button of the Mouse + Moving + releasing the Mouse**.

The corresponding Hotspots , and only them will automatically be displayed in the different Hotspots windows as shown in image 20.



**Fig 20: Hotspots displayed for a particulat Time Interval**

#### 6.4.7 Settings of the Profiler

Before running the profiler, some settings can be modified. The Menu bar at the upper left of the profiler window displays two Menus:

- **The View Menu:** Allows the user to display or not the different dock windows
- **The Settings Menu:** Allows the user to modify the Signals and Gvsoc settings.

The Gvsoc settings allow the user to select the groups of signals whose information he/she wants to get from the profiler backend. Allowing fewer groups of signals minimizes the profiler backend calculations and memory allocation.

The hotspots Settings allow the user to select for which cores he/she wants to have the hotspots information from the backend. It also minimizes the backend calculations and the memory footprint of the backend if fewer cores are selected.

## 6.5 GACY

### 6.5.1 Introduction

Gacy is a command-line tool used as a generic runner for all Gap boards and platforms.

For this purpose, it provides the following features:

- Generic, board-specific and platform-specific options in order to describe all what is required to launch execution on the target.
- Set of target descriptors. Each descriptor is used by Gacy to know what is allowed and how to do it on a specific target (e.g. Gap9 EVK).
- Set of utilities for building flash image, file-systems, target stimuli and so on in order to make it easy for the user to include all what is needed by his application.

- Common interface to interact with the target for flashing, running and so on, so that it is easy to switch from one board to another, or one platform to another.

Gapy is written in Python. It can be extended to support more targets. A target is described by a Python class and is selecting which features is available for this target and how to implement it.

## 6.5.2 Command-line usage

Gapy can be invoked with the following command:

```
gapy <options> <commands>
```

<commands> is a list of commands that gapy should process in the order they are specified, from left to right.

<options> is a list of options used to configure the commands to be executed.

### Options

The list of options can be obtained with the following option:

```
gapy --help
```

Some options are added only when other options are specified, for example depending on the target specified. These additional options can always be obtained with the option *-help* combined with other options.

Here are some generic options, the other ones will be explained throughout the rest of the documentation:

Option	Description
<code>-ver-</code> <code>bose=&lt;string&gt;</code>	Configure Gapy verbosity. By default nothing is printed. Can be debug, info, warning, error or critical.
<code>-work-</code> <code>dir=&lt;string&gt;</code>	Configure the working directory. This is where all the output files such as flash images will be generated. Also, any shell command will be executed from this directory.
<code>-plat-</code> <code>form=&lt;string&gt;</code>	Platform on which the target is used. This is useful to use the same target on board and various simulators.

### Commands

The list of available commands can be obtained with this command:

```
gapy commands
```

Since some commands are only available on some targets, the list of commands depends on the options specified, in particular on the target specified.

Here are the main commands:

Command Name	Description
commands	Show the list of available commands
targets	Show the list of available targets
image	Generate the target images needed to run execution
flash	Upload the flash contents to the target
flash_layout	Dump the layout of the flashes
flash_properties	Dump the properties for each flash and each flash section
run	Start execution on the target

## Choosing the target

A target is the system on which Gapy will manage the execution. This can be a board or a more complex system like a set of boards connected together.

A target is described through a Python module. Gapy needs to be specified where it should look for such modules in order to know what are the available targets. This must be specified through the option `-target_dir` and can be used several times to specify multiple directories. Gapy contains one directory with the common Gap targets such as the Gap9 EVK.

The list of available targets can be displayed with the following command:

```
gapy targets
```

Each target correspond to a python module that Gapy found from the specified target directories.

The target to be used can then be specified with the option `-target=<name>` where `<name>` corresponds to one target name shown by the `targets` command.

Not that once a target is specified, target-specific options may then be available.

## Target options

Each target or platform can have its own options. The main ones are exposed through classic command-line options. For targets or platforms having too many options, like for simulators, options can be specified as key-value properties through the command-line option `-target-opt=<name>=<value>` like in this example:

```
gapy --target-opt=**/l2/size=10000000
```

This option can be used several times to specifies several properties. The list of target options are passed as a list of key-value properties to the target, which will take care of them. See the documentation of each target and platform for more information.

### 6.5.3 Target description

#### Target module

Each target must be described by a Python module and must be accessible in one of the target directories specified on the command-line.

The target module `gap9/evk.py` which can be found in the targets inside Gapy can be looked at as an example:

```
class Target(gapylib.chips.gap.gap9_v2.target.Target):

    def __init__(self, options: list):
        super().__init__(options)

        self.register_flash(
            gapylib.chips.gap.flash.DefaultFlashRomV2(self, 'flash',
                flash_attributes={
                    "flash_type": "spi",
                    "rtl_stim_format": "slm_16",
                    "rtl_stim_name": "slm_files/flash_stim.slm",
                    "flasher": "gap_flasher-gap9_evk.elf",

```

(continues on next page)

(continued from previous page)

```

        "flasher_sector_size": 0x2000,
        "littlefs_block_size": 262144,
    },
    size=8*1024*1024
))
self.register_flash(
    gapylib.chips.gap.flash.DefaultFlashRomV2(self, 'mram',
        flash_attributes={
            "flash_type": "mram",
            "rtl_stim_format": "mram",
            "rtl_stim_name": "slm_files/mram_stim.slm",
            "flasher": "gap_flasher-gap9_evk-mram.elf",
            "flasher_sector_size": 0x2000,
            "littlefs_block_size": 262144,
        },
        size=2*1024*1024
))

def __str__(self) -> str:
    return "GAP9 evaluation kit"

```

Any target descriptor must inherit from `gapylib.target.Target`. This example inherits first from a common gap9 descriptor which itself inherits from `gapylib.target.Target`.

The descriptor should return the target name when it is converted to string, using the method `__str__`.

The descriptor can then optionnally overload some methods of the base class in order to add some custom behaviors.

The most useful one is to overload the method `gapylib.target.Target.append_args()` in order to specify additionnal command-line options, like in this example:

```

def append_args(self, parser: argparse.ArgumentParser):
    super().append_args(parser)

    parser.add_argument("--platform", dest="platform", required=True, choices=['board',
        'rtl'],
        type=str, help="specify the platform used for the target")

    [args, _] = parser.parse_known_args()

    if args.platform == 'board':
        self.runner = gapylib.chips.gap.gap9_v2.board_runner.Runner(self)

    else:
        raise RuntimeError('Unknown platform: ' + args.platform)

    self.runner.append_args(parser)

```

This method gets an argparse parser as argument, which can be used to add target-specific arguments. It can also be used to immediately parse arguments in order to check the value of the added arguments.

Another useful method to overload is `gapylib.target.Target.handle_command()` which allows defining target-specific commands, like in this example:

```
def handle_command(self, cmd: str):

    args = self.get_args()

    if cmd == 'run':
        self.runner.run()

    elif cmd == 'traces':
        if args.platform == 'rtl':
            self.runner.traces()

    elif cmd == 'flash':
        gapylib.target.Target.handle_command(self, cmd)

        # board platform needs to upload flash content
        if args.platform == 'board':
            self.runner.flash()

    else:
        gapylib.target.Target.handle_command(self, cmd)
```

As seen on this example, this can be used either to add a command or also to change the behavior of an existing command.

## Flash registration

Another goal of the target descriptor is to declare the available flashes and their layout in order to let Gapy manage them.

As we can see on the previous example, this is done by calling the method `gapylib.target.Target.register_flash()`:

```
class Target(gapylib.chips.gap.gap9_v2.target.Target):

    def __init__(self, options: list):
        super().__init__(options)

        self.register_flash(
            gapylib.chips.gap.flash.DefaultFlashRomV2(self, 'flash',
                flash_info={
                    "flash_type": "spi",
                    "rtl_stim_format": "slm_16",
                    "rtl_stim_name": "slm_files/flash_stim.slm",
                    "flasher": "gap_flasher-gap9_evk.elf",
                    "flasher_sector_size": 0x2000,
                    "littlefs_block_size": 262144,
                },
                size=8*1024*1024
        )
```

This method takes as argument the flash instance, whose class must derive from `gapylib.flash.Flash`.

This class must be instantiated with as arguments, the flash name, the flash size and optionnally some flash attributes.

The flash name is the name which is used everywhere to refer to this flash, like on the command-line to set flash section properties.

The flash size is used to check that the flash is not overflowed and can also be used by some sections like the `raw` section in `gapylib.fs.raw.RawSection` which is using it to spread until the end of the flash.

The flash attributes given to the instance can be used to store information about the flash, which can then be used by the target to understand how the flash should be handled. This is used for example on the board to get some information about the way the flasher should update the content of the flash.

The class `gapylib.chips.gap.flash.DefaultFlashRomV2` is derived from the base class `gapylib.flash.Flash` and is adding some information about the allowed sections and default flash layout. More information can be found in the sections about flash management and about gap targets.

## 6.5.4 Gap targets

### Default layout

All Gap targets are instantiating flashes which are derived from the same flash description described in `gapylib.chips.gap.flash.DefaultFlashRomV2`.

This flash description first defines all the allowed sections, and then provides the default layout.

The default layout contains first a ROM section, in case the flash is used for booting, a partition table, all kind of file-systems and a final raw partition to take the rest of the empty space in the flash.

The JSON file defining this default layout is the following:

```
{  
    "sections": [  
        {  
            "name": "rom",  
            "template": "rom",  
            "properties": {  
                "binary": null,  
                "boot": false  
            }  
        },  
        {  
            "name": "partition table",  
            "template": "partition table"  
        },  
        {  
            "name": "readfs",  
            "template": "readfs",  
            "properties": {  
                "files": []  
            }  
        },  
        {  
            "name": "hostfs",  
            "template": "hostfs",  
            "properties": {  
                "files": []  
            }  
        }  
    ]  
}
```

(continues on next page)

(continued from previous page)

```

},
{
  "name": "lfs",
  "template": "lfs",
  "properties": {
    "size": 0,
    "root_dir": null
  }
},
{
  "name": "raw",
  "template": "raw",
  "properties": {
    "size": -1
  }
}
]
}

```

## Board platform

### OpenOCD configuration

When using Gap boards, the target is managed through OpenOCD, which requires a few specific options to be set to properly configure OpenOCD.

By default, Gapy will call the executable *openocd*, which should be accessible from the environment variable PATH. If it is not the case, the path to the executable can be specified through the option *-openocd=<openocd path>*.

The OpenOCD cable must be specified through option *-openocd-cable=<cable path>*. This option is mandatory and has no default value. It can for example be set to *\$(GAP\_SDK\_HOME)/utils/openocd\_tools/tcl/gapuno\_ftdi.cfg*.

The OpenOCD script for managing the target must be specified through option *-openocd-script=<script path>*. This option is mandatory and has no default value. It can for example be set to *\$(GAP\_SDK\_HOME)/utils/openocd\_tools/tcl/gap9revb.tcl*.

The path to the OpenOCD GAP tools must be specified through option *-openocd-tools=<tools path>*. This option is mandatory and has no default value. It can for example be set to *\$(GAP\_SDK\_HOME)/utils/openocd\_tools*.

## GDB

GDB is by default disabled and can be enabled by adding option *-gdb*. The port for the connection between GDB and OpenOCD can be specified through option *-gdb-port=<port>*.

## Flash content upload

By default, to avoid unnecessary flash upload, Gapy will try to determine if the content should be uploaded. The decision is based on the flash automatic mode, which determines if the flash is considered empty or not. More information is given in [Flash management](#).

If the flash content is uploaded, Gapy will try to limit the size of the upload by removing the last empty sections.

### 6.5.5 Flash management

One of the main goal of Gapy is to help the user to produce the content of each flash.

#### Flash layout

The way the content of a flash is organized is specified through a JSON file describing all the possible sections that it can contain, and how they are organized together.

A default flash layout is provided for each target, so that by default, the user does not have to worry about this aspect. This default layout usually allows all possible kind of sections, so that all features are exposed. Please refer to the target documentation to know which is the default layout.

If there is a need to have a different layout, a custom layout can be provided for each flash. This can be useful for example to include several times the same file-system, or to include a dedicated section.

The flash layout JSON file can be specified with the following option:

```
gapy --flash-content=<filepath>@<flashname>
```

<filepath> is the path to the JSON file describing the flash layout and <flashname> is the name of the flash for which the layout should be changed. It is possible to give a different layout for each flash.

#### Flash layout format

Here is an exemple of a flash layout:

```
{
  "sections": [
    {
      "name": "rom",
      "template": "rom",
      "properties": {
        "binary": null,
        "boot": false
      }
    },
    {
      "name": "partition table",
      "template": "partition table"
    },
    {
      "name": "readfs",
      "template": "readfs",
      "properties": {
        "binary": null
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "files": []
    }
]
}

```

The top object must contain the key *section* with an associated array containing an ordered list of sections of the flash. The sections will appear in the flash in the same order.

Each section is an object which must contain at least the keys *name* and *template*.

The key *name* must be associated a string describing the name of the section. This name can be used to refer to this section at various places, for example on the command-line. It can also be used for partition tables.

The key *template* must be associated a string describing which Gapy python class will handle this section. When a target is defined, each available flash is described, and the allowed sections for each flash are specified, associating the template name to a Python class. The template name can for example be *rom*, or *readfs*.

There can also be an optional key *properties*, in order to override the value of section properties. This key must be associated an object containing the definition of key-value properties. Each property is a key with an associated value in the object. The available properties are specific to each section, please refer to the section documentation to see what is allowed. The properties will be used by the Python class handling the section in order to collect additional information, such as files to be included in a file-system. These properties are defined by the section and can be overwritten by the flash layout, and also from command-line arguments, described in the next section.

## Overwritting flash properties

Flash section properties can be overwritten from command-line arguments, like in this example:

```
gapy --flash-content=my_layout.json@flash --flash-property=file.bin@flash:readfs:files
```

As shown on this example, a new property value can be specified with the option *--flash-property=<value>@<flashname>:<sectionname>:<propertyname>*.

The type of the property value is the one of the default value specified when the section declared the property, and is used to properly convert the value specified on the command-line. If the value is a list, overwritting it will actually append the specified value to the list.

Overwritting properties can be useful to keep the flash layout as a template and then specify real values such as file-system files or ROM content on the command-line, depending on the application.

## Available flash sections

### ROM

A flash can often be used as a bootable device. In this case the ROM will expect a header in the flash describing the executable to be loaded in memory. This section can be included when the ROM is allowed to boot from this flash.

This section is chip-dependent. The only one currently available is defined in *gapylib.chips.gap.rom\_v2* and can be used for all gap9 chips. This section must be placed as the first section of the flash. It supports the following properties:

- *boot*: its value is a boolean and should be set to *true* if the chip should boot on this flash. Gapy will upload the content of the section to the target only it is *true*. This is usefull to avoid uploading the content while the target is booting through JTAG.

- *binary*: its value is the path to the executable which should be loaded by the ROM. Gapy will use it to generate the required information in the ROM section. Note that if *boot* is *false*, the section content will not be uploaded, even though *binary* contains a valid executable.

For the flash automatic mode, such a section is considered empty if property *boot* is set to *false* or if there is no executable specified.

### Partition table

This section can be placed anywhere in the flash and will build a partition table of all the sections which are after this one.

This section does not have any property.

For the flash automatic mode, such a section is considered empty if all the next sections are considered empty.

### Read FS

This section will generate a PMSIS read FS, which is a very simple read-only file-system, usefull for input stimuli.

It supports the following property:

- *files*: its value is a list of strings, each one specifying the path to a file to be embedded into the file-system.

For the flash automatic mode, such a section is considered empty if there is no file specified.

### Little FS

This section will generate a LittleFS section.

It supports the following properties:

- *size*: size in bytes of the file-system. Since the file-system is writable, the user has to specify how much space is reserved for the file-system. The specified size is the size of the whole file-system, including headers.
- *root\_dir*: string specifying the path to a directory on the workstation to be embedded inside the file-system. Only the content is included, not the directory itself. The content is copied into the section, and is available at runtime on the target.

For the flash automatic mode, such a section is considered empty if there is no root directory specified.

### Raw

This section is a placeholder for empty spaces. This is useful in order to make it visible in the partition table, for example to make the empty space at the end of the flash be visible from the partition table.

It supports the following properties:

- *size*: size in bytes of the section. The size can be set to -1 in order to cover all the space from the beginning of the section to the end of the flash.

For the flash automatic mode, such a section is always considered empty.

## Displaying the flash section properties

The list of properties for each flash and each flash section can be obtained with this command:

```
gapy flash_properties
```

This should display something similar to this example:

```
Section properties for flash: mram
+-----+
| Section name      | Section properties
+-----+
| rom               | +-----+-----+
|                   | | Property name | Property value | Property description
|                   | |             |             |
|                   | +-----+-----+
|                   | | binary       | None          | Executable to be loaded by the ROM.
|                   | | boot         | False         | True if the ROM will boot using this ROM section.
|                   | +-----+-----+
| partition table   | +-----+
| readfs            | +-----+-----+
|                   | | Property name | Property value | Property description
|                   | |             |             |
|                   | +-----+-----+
|                   | | files        | []           | List of files to be included in the ReadFS.
|                   | +-----+-----+
| hostfs            | +-----+-----+
|                   | | Property name | Property value | Property description
|                   | |             |             |
|                   | +-----+-----+
|                   | | files        | []           | List of files to be included in the HostFS.
|                   | +-----+-----+
| lfs                | +-----+-----+
|                   | | Property name | Property value | Property description
|                   | |             |             |
|                   | +-----+-----+
```

(continues on next page)

(continued from previous page)

This is also showing the current value of the properties. If some properties are overwritten from the command-line, this will show the values after being overwritten so that the command-line can be checked using this command.

## Displaying the flash layout

In order to better understand the layout of the flash, and especially in order to see the effect of overwriting section properties, the flash layout can be displayed with the following command:

gapy flash\_layout

This should display something similar to this example:

---

Chapter 6 Tools

(continued from previous page)

0x4	partition table	0xa0					
Content				Offset	Size	Name	
				0x4	0x20	header	
Offset	Name			Size	Value		
0x4	magic_number		0x2	0x2ba			
0x6	partition_table_version		0x1	0x1			
0x7	nb_entries		0x1	0x4			
0x8	crc		0x1	0x0			
0x9	padding		0xb	-			
0x14	md5		0x10	-			
				0x24	0x20	section0 header	
Offset	Name		Size	Value			
0x24	magic_number	0x2	0x1ba				
0x26	type	0x1	0x1				
0x27	subtype	0x1	0x81				
0x28	offset	0x4	0xa4				
0x2c	size	0x4	0x8046				
0x30	name	0x10	-				
0x40	flags	0x4	0x0				
				0x44	0x20	section1 header	

(continues on next page)

(continued from previous page)

Offset	Name	Size	Value								
0x44	magic_number	0x2	0x1ba								
0x46	type	0x1	0x1								
0x47	subtype	0x1	0x81								
0x48	offset	0x4	0x80ea								
0x4c	size	0x4	0x0								
0x50	name	0x10	-								
0x60	flags	0x4	0x0								
							0x64	0x20	section2 header		
0x64	magic_number	0x2	0x1ba								
0x66	type	0x1	0x1								
0x67	subtype	0x1	0x81								
0x68	offset	0x4	0x80ea								
0x6c	size	0x4	0x0								
0x70	name	0x10	-								
0x80	flags	0x4	0x0								
							0x84	0x20	section3 header		
0x84	magic_number	0x2	0x1ba								
0x86	type	0x1	0x1								

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

Offset	Name	Size	Value
0x80ea	data	0x1f7f16	-

It is also possible to tune the level of details with this option:

```
gapy --flash-layout-level=1 flash_layout
```

This will show the previous layout like this:

Section offset	Section name	Section size	Section content																		
0x0	rom	0x4	<table border="1"><tr><th>Offset</th><th>Size</th><th>Name</th></tr><tr><td>0x0</td><td>0x4</td><td>ROM header</td></tr></table>	Offset	Size	Name	0x0	0x4	ROM header												
Offset	Size	Name																			
0x0	0x4	ROM header																			
0x4	partition table	0xa0	<table border="1"><tr><th>Offset</th><th>Size</th><th>Name</th></tr><tr><td>0x4</td><td>0x20</td><td>header</td></tr><tr><td>0x24</td><td>0x20</td><td>section0 header</td></tr><tr><td>0x44</td><td>0x20</td><td>section1 header</td></tr><tr><td>0x64</td><td>0x20</td><td>section2 header</td></tr><tr><td>0x84</td><td>0x20</td><td>section3 header</td></tr></table>	Offset	Size	Name	0x4	0x20	header	0x24	0x20	section0 header	0x44	0x20	section1 header	0x64	0x20	section2 header	0x84	0x20	section3 header
Offset	Size	Name																			
0x4	0x20	header																			
0x24	0x20	section0 header																			
0x44	0x20	section1 header																			
0x64	0x20	section2 header																			
0x84	0x20	section3 header																			
0xa4	readfs	0x8046	<table border="1"><tr><th>Offset</th><th>Size</th><th>Name</th></tr><tr><td>0xa4</td><td>0xc</td><td>header</td></tr><tr><td>0xb0</td><td>0x1d</td><td>file0 header</td></tr><tr><td>0xcd</td><td>0x1d</td><td>file1 header</td></tr><tr><td>0xea</td><td>0x4000</td><td>file0</td></tr><tr><td>0x40ea</td><td>0x4000</td><td>file1</td></tr></table>	Offset	Size	Name	0xa4	0xc	header	0xb0	0x1d	file0 header	0xcd	0x1d	file1 header	0xea	0x4000	file0	0x40ea	0x4000	file1
Offset	Size	Name																			
0xa4	0xc	header																			
0xb0	0x1d	file0 header																			
0xcd	0x1d	file1 header																			
0xea	0x4000	file0																			
0x40ea	0x4000	file1																			
0x80ea	hostfs	0x0																			
0x80ea	lfs	0x0	<table border="1"><tr><th>Offset</th><th>Size</th><th>Name</th></tr></table>	Offset	Size	Name															
Offset	Size	Name																			
0x80ea	raw	0x1f7f16	<table border="1"><tr><th>Offset</th><th>Size</th><th>Name</th></tr></table>	Offset	Size	Name															
Offset	Size	Name																			

(continues on next page)

(continued from previous page)

			+-----+-----+-----+	
			0x80ea   0x1f7f16   header	
			+-----+-----+-----+	

The level can vary from 0 for minimal details to 3 for maximum details.

## Flash automatic mode

For platforms like boards where the flash content needs to be uploaded, the target will automatically decide if the flash content must be uploaded when command `flash` is executed, depending on the content of flash.

For that, the flash will check the sections and will upload the content only if at least one section is not empty. It is up to each section to decide if it is empty depending on its properties. More information is available in each section documentation.

In case the default behavior is causing trouble, the upload operation can be forced by using the option `-flash-no-auto`.

## Flash section definition

New sections can be defined by adding in Gapy a new class derived from `gapylib.flash.FlashSection`.

Here is an example of a section definition:

```
class RomFlashSection(FlashSection):

    def __init__(self, parent: Flash, name: str, section_id: int):
        super().__init__(parent, name, section_id)

        self.segments = []

        self.declare_property(name='binary', value=None,
            description="Executable to be loaded by the ROM."
    )
```

Section properties can be declared when the section is instantiated. Only the properties defined here can be overwritten by the flash layout or the command-line.

They can be used in other section methods to condition what the section is doing, by calling the method `gapylib.flash.FlashSection.get_property()`.

Adding more section properties is the best way to make the section behavior configurable from the command-line.

The section should also overload the method `gapylib.flash.FlashSection.set_content()`, which is called by the flash to give the section its content. The section should then parse it and generate the section image out of that.

Some utilities are provided in `gapylib.utils` to declare C structures, which can be filled depending on the section content, so that the flash can automatically generate the section image out of the declared C structure. A simple example can be found in `gapylib.fs.readfs`.

## 6.5.6 Guides

### How to extend Gapy with more options

There are 4 ways to extend Gapy with more options, depending on where it will impact Gapy:

- Command-line options: Their role is to configure global Gapy behavior like the target or the OpenOCD cable to be used when running on a board. The number of global options should be kept low. They can be obtained with command-line option `-help`. Adding command-line options can only be done in the target class, by overloading the method `gapylib.target.Target.append_args()`. More information can be obtained here [Target module](#).
- Flash section properties: Their role is to configure each section behavior. They are specific to each section and thus to each flash. They can be overwritten by the flash layout defined by the target and also on the command-line using the option `-flash-property`. They can be obtained with command-line command `flash_properties`. Adding flash section properties is done by registering new properties when the section is instantiated. More information can be obtained here [Flash section definition](#).
- Flash attributes: Their role is to give dedicated information about the way to handle a flash. They are specific to each flash and are described by the target when the flash is registered. Their value cannot be changed from the command-line. They are used for example to give the size of each flash. Adding flash attributes is done by adding it when the flash is registered. More information can be obtained here [Target module](#).
- Target options: Their role is to configure specific target behavior. They can be specified using the command-line option `-target-opt`. These options are for now reserved for internal usage, and only used to change the architecture simulated by GVSOC.

### How to change the layout of a flash

Each flash is having a default layout defined by the target. More information about the default layout can be found in the target documentation, here for Gap targets [Default layout](#).

It is possible to define a custom layout by providing a JSON file describing the layout on the command-line using option `-flash-content`. More information is available here [Flash layout](#).

## 6.5.7 GAPY Python API

### GAPY API Reference

#### Base classes for target description

##### `gapylib.target`

Provides target class common to all targets which brings common methods and commands.

**class** `gapylib.target.Target(options: Optional[list] = None)`

Parent class for all targets, which provides all common functions and commands.

##### `options`

A list of options for the target.

**Type** List

**append\_args(parser: argparse.ArgumentParser)**

Append target specific arguments to gapy command-line.

This should be called only by gapy executable to register some arguments handled by the default target but it can be overloaded in order to add arguments which should only be visible if the target is active.

**Parameters** `parser` (`argparse.ArgumentParser`) – The parser where to add arguments.

**get\_abspath**(`relpath: str`) → `str`

Return the absolute path depending on the working directory.

If no working directory was specified, the relpath is appended to the current directory. Otherwise it is appended to the working directory.

**Parameters** `relpath` (`str`) – Relative path of the file.

**Returns** The absolute path.

**Return type** `str`

**get\_args**() → `argparse.ArgumentParser`

Return the command-line arguments.

**Returns** The arguments.

**Return type** `argparse.ArgumentParser`

**static get\_file\_path**(`relpath: str`) → `str`

Return the absolute path of a file.

Search into the python path for this file and returns its absolute path. This can be used to find configuration files next to python modules.

**Parameters** `relpath` (`str`) – The file relative path to look for in the python path

**Returns** The path.

**Return type** `str`

**get\_working\_dir**() → `str`

Return the working directory.

All files produced by gapy will be dumped into this directory.

**Returns** The working directory.

**Return type** `str`

**handle\_command**(`cmd: str`)

Handle a command.

This should be called only by gapy executable but can be overloaded by the real targets in order to add commands or to handle existing commands differently.

**Parameters** `cmd` (`str`) – Command name to be handled.

**parse\_args**(`args: any`)

Handle arguments.

This should be called only by gapy executable to handle arguments but it can be overloaded in order to handle arguments specifically to a target.

**Parameters** `args` – The arguments.

**register\_flash**(`flash: gapylib.flash.Flash`)

Register a flash.

The flash should inherit from `gapylib.flash.Flash`. This will allow gapy to produce images for this flash.

**Parameters** `flash` (`gapylib.flash.Flash`) – The flash to be registered.

**set\_target\_dirs**(*target\_dirs*: *list*)

Set the target directories.

This sets the directories where to look for targets. This should be called only by gapy executable.

**Parameters** **target\_dirs** (*list*) – The directories.

**set\_working\_dir**(*working\_dir*: *str*)

Set the working directory.

All files produced by gapy will be dumped into this directory.

**Parameters** **working\_dir** (*str*) – The working directory.

**gapylib.target.get\_target**(*target*: *str*) → *gapylib.target.Target*

Returns the class implementing the support for the specified target.

The target is specified as a python module which is imported from python path. It returns the class ‘Target’ from the imported module.

**Parameters** **target** (*str*) – Name of the target. The name must corresponds to a python module.

**Returns** The class ‘Target’ of the imported module.

**Return type** class

**gapylib.flash**

Common definitions for flashes. In particular, it defines a common class that each flash section should inherit from. The same for each flash.

**class** **gapylib.flash.Flash**(*target*: *gapylib.target.Target*, *name*: *str*, *size*: *int*, *image\_name*: *str* = *None*, *flash\_attributes*: *dict* = *None*)

Parent class for all flashes. This provides utility functions for describing a flash.

**target**

Target containing the flash.

**Type** *gapylib.target.Target*

**name**

Name of the flash

**Type** str

**size**

Size of the flash

**Type** int

**image\_name**

Image name where the content of the flash will be dumped

**Type** str

**flash\_attributes**

Set of attributes describing the flash.

**Type** dict

**dump\_image()**

Dump the content of the flash in binary form to the specified file.

**Parameters** **fd** – File descriptor

**dump\_layout(*level*: int)**  
Dump the layout of the flash.

**Parameters** **level** (*int*) – Dumping depth

**dump\_section\_properties()**  
Dump the section properties of the flash.

**dump\_sections()**  
Dump the sections images and a description of each section  
The description is a JSON file containing the description of each section.

**get\_flash\_attribute(*name*: str) → any**  
Return the value of a flash property.  
This can be called to get the value of a property concerning the flash.

**Parameters** **name** (*str*) – Name of the property

**Returns** The property value.

**Return type** str

**get\_image(*first*: Optional[int] = None, *last*: Optional[int] = None) → bytes**  
Return the content of the flash.

**Returns**

- *bytes* – The flash content.
- **first** (*int*) – The index of the first section from which the image must be generated
- **last** (*int*) – The index of the last section until which the image must be generated

**get\_image\_name() → str**  
Return the name of the flash image.  
Gapy will dump the binary image of the flash to the image name.

**Returns** The image flash name.

**Return type** str

**get\_image\_path() → str**  
Return the path of the file containing the flash image.

**Returns** The image file path.

**Return type** str

**get\_name() → str**  
Return the name of the flash.

**Returns** The flash name.

**Return type** str

**get\_section\_by\_name(*name*: str)**  
Get a section by its name :param name: Name to search for :type name: str

**Returns** Flash section with name “name”

**Return type** *FlashSection*

**get\_sections()**  
Get all the sections of the flash.

**Returns** The flash sections.

**Return type** list

**get\_size()** → int

Get flash size.

**Returns** The flash size.

**Return type** int

**get\_target()** → *gapylib.target.Target*

Return the target containing the flash.

**Returns** The target.

**Return type** *gapylib.target.Target*

**is\_empty()** → bool

Tell if the flash is empty.

This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the flash is empty, False otherwise.

**Return type** str

**register\_section\_template(*template\_name*: str, *section\_template*: *gapylib.flash.FlashSection*)**

Register a section template.

A section template is used by a target to define the kind of sections which are allowed for the flash. The section content can then instantiates section templates.

**Parameters**

- **template\_name** (*dict*) – Name of the template
- **section\_template** (*FlashSection*) – Section template

**set\_content(*content\_dict*: dict)**

Set the content of the flash.

**Parameters** **content\_dict** (*dict*) – Flash content

**set\_properties(*properties*: dict)**

Set the section properties of this flash.

This will be used to overwrite flash section properties retrieved from its content.

**Parameters** **properties** (*dict*) – The flash properties

**class gapylib.flash.FlashSection(*parent*, *name*: str, *section\_id*: int)**

Parent class for all flash sections. This provides utility functions for describing the section. A section is a logical part of a flash, like a ROM header or a FS.

**parent**

Name of the section.

**Type** *gapylib.flash.Flash*

**name**

Name of the section.

**Type** str

**id**

Id of the section.

**Type** int

**add\_struct**(*cstruct*: any) → any  
Add a structure to the section.

**Returns** The structure.

**Return type** *CStruct*

**align\_offset**(*alignment*: int) → int  
Align the current offset.

The current offset is aligned by the specified alignment in bytes. This can be used to add some padding.

**Parameters** *size* (int) – The size to allocate.

**Returns** The size allocated to align the offset.

**Return type** int

**alloc\_offset**(*size*: int) → int  
Allocate an offset.

This returns the current offset and increases it by the specified size. This can be used to allocate some fields in the section and let the flash know the size of the section.

**Parameters** *size* (int) – The size to allocate.

**Returns** The allocated offset.

**Return type** int

**declare\_property**(*name*: str, *value*: any, *description*: str)  
Declare a section property.

Section properties are used to tell the section to configure the sections. They must be declared with this method before they can be overwritten from the flash content or command-line flash properties.

**Parameters**

- **name** (str) – Name of the property.
- **value** (any) – Value of the property.
- **description** (str) – Description of the property.

**dump\_properties**() → str  
Dump the section properties as a table.

**dump\_section\_description**() → Dict[str, Any]  
Dump the description of a section

**Returns** A description of the section parameters

**Return type** Dict[str, Any]

**dump\_table**(*level*: int) → str  
Dump the section as a table.

This should only be called by the gap executable and should be overloaded by the section to dump the content of the section into a table.

**Parameters** *level* (int) – The dump level. The content should be displayed only if it is superior or equal to zero.

**finalize()**

Finalize the section.

This can be called to set internal section fields which requires some knowledge of the offset or size of other sections. The structure of the section should not be changed in this step

**get\_current\_offset() → int**

Get the current offset.

**Returns** The current offset.

**Return type** int

**get\_flash() → gapylib.flash.Flash**

Get the sflash containing this section.

**Returns** The section name.

**Return type** gplib.flash.Flash

**get\_id() → int**

Get the section ID.

**Returns** The section ID.

**Return type** int

**get\_image() → bytes**

Dump the content of the section in bynary form to the specified file descriptor.

**Parameters** **file\_desc** – File descriptor.

**get\_image\_name() → str**

Get the section image name

**Returns** The section image name

**Return type** str

**get\_image\_path() → str**

Get the section image path

**Returns** The section image path

**Return type** str

**get\_name() → str**

Get the section name.

**Returns** The section name.

**Return type** str

**get\_next\_section() → gapylib.flash.FlashSection**

Return the next section in the flash.

**Returns** The next section.

**Return type** FlashSection

**get\_offset() → int**

Get the section offset.

**Returns** The section offset.

**Return type** int

**get\_partition\_type()** → int

Return the partition type.

This information can be used by the partition table as the subtype. This method returns an unknown type (0xff) and should be overloaded by real sections.

**Returns** The partition type.

**Return type** int

**get\_property(name: str)** → any

Return the value of a property.

This can be called to get the value of a property set from the section content.

**Parameters** name (str) – Name of the property

**Returns** The property value.

**Return type** str

**get\_size()** → int

Get the section size.

The returned size is based on the allocated offsets and the requested alignments.

**Returns** The section size.

**Return type** int

**is\_empty()** → bool

Tell if the section is empty.

This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the section is empty, False otherwise.

**Return type** str

**set\_content(offset: int, content\_dict: dict)**

Set the content of the section.

This can be called to set the default content, and may be overwritten by gapy executable if a new content has been specified. The section should in this step declare all its internal structure so that the offsets and sizes of all sections are known after this step.

**Parameters**

- **offset** (int) – Start offset of the section
- **content\_dict** (dict) – Content of the section

**set\_offset(offset: int)**

Set the start offset of the section.

The offset is used by the flash to know where the section starts. It is also used to implement a flash allocator in the section.

**Parameters** offset (int) – The section offset.

**class gapylib.flash.FlashSectionProperty(name: str, value: any, description: str)**

Placeholder for flash section properties.

**name**

Name of the property.

**Type** str

**value**

Value of the property.

**Type** any

**description**

Description of the property.

**Type** str

**gapylib.utils**

Provides utility classes

**class gapylib.utils.CStruct(name: str, parent: gapylib.utils.CStructParent)**

Class for gathering CStruct fields together into a common structure.

**name**

Name of this structure.

**Type** str

**parent**

Parent, which is aggregating all structures.

**Type** str

**add\_field(name: str, field\_format: str) → gapylib.utils.CStructField**

Add a scalar field.

The field is added to the structure. The fields are dumped in the order they are added.

**Parameters**

- **name (str)** – Name of the field
- **field\_format (str)** – Format of the field, described by struct python package

**Returns** The field.

**Return type** *CStructField*

**add\_field\_array(name: str, size: int) → gapylib.utils.CStructField**

Add an array field.

The field is added to the structure. The fields are dumped in the order they are added.

**Parameters**

- **name (str)** – Name of the field
- **size (int)** – Size of the array

**Returns** The field.

**Return type** *CStructField*

**add\_padding(name: str, align: int)**

An empty field for padding.

Can be used to add empty bytes to align the next field.

**Parameters**

- **name (str)** – Name of the field

- **align** (*int*) – Alignment of the next field.

**dump\_table**(*level*: *int*) → str

Dump the structure to a table.

This will dump all the fields of this structure to the same table.

**Parameters** **level** (*int*) – Dump level

**Returns** The table.

**Return type** str

**get\_field**(*name*: str) → *gapylib.utils.CStructField*

Get field.

Can be used for fields added with add\_field or add\_field\_array

**Parameters** **name** (str) – Name of the field

**Returns** The field.

**Return type** *CStructField*

**get\_name**() → str

Get sub-section name.

**Returns** The sub-section name.

**Return type** str

**get\_offset**() → int

Get sub-section offset.

**Returns** The sub-section offset.

**Return type** int

**get\_size**() → int

Get sub-section size.

**Returns** The sub-section size.

**Return type** int

**pack**() → bytes

Pack all fields in bynary form.

This can be used to dump all the fields to a file.

**Returns** The values of the fields packed into a byte array.

**Return type** bytes

**set\_field**(*name*: str, *value*: any)

Set a field value.

Can be used for fields added with add\_field or add\_field\_array

**Parameters**

- **name** (str) – Name of the field
- **value** (any) – Value of the field

**class** *gapylib.utils.CStructArray*(*name*: str, *size*: int, *value*: any, *offset*: int)

Class for scalar fields.

---

**dump\_table**(table: *prettytable.PrettyTable*, level: *int*)

Dump the field to a table.

This declaration is propagated towards the parents so that the runner taking care of the flash image generation knows where the flash are.

#### Parameters

- **x** (*PrettyTable*) – Table where the field should be dumped.
- **level** (*int*) – Dump level

**class** *gapylib.utils.CStructField*(name: *str*, size: *int*, value: *any*, offset: *int*)

Parent class for all kind of CStruct fields.

#### **name**

Name of this field.

**Type** *str*

#### **size**

Size of this field.

**Type** *int*

#### **value**

value of this field.

#### **offset**

Offset of this field in the flash.

**Type** *int*

**get\_offset()** → *any*

Get field offset in flash.

**Returns** The offset.

**Return type** *int*

**set**(value: *any*)

Set field value.

Set the value of the field.

**Parameters** **value** – Field value

**class** *gapylib.utils.CStructParent*(name: *str*, parent: *gapylib.flash.FlashSection*)

Class for gathering several structures together into a sub-section.

#### **name**

Name of this sub-section.

**Type** *str*

#### **parent**

Parent, which is aggregating all sub-sections.

**Type** *FlashSection*

**add\_struct**(cstruct: *gapylib.utils.CStructField*) → *gapylib.utils.CStructField*

Add a structure to the sub-section.

**Returns** The structure.

**Return type** *CStruct*

**align\_offset**(*alignment: int*) → int

Align the current offset.

The current offset is aligned by the specified alignment in bytes. This can be used to add some padding.

**Parameters** **size** (*int*) – The size to allocate.

**Returns** The size allocated to align the offset.

**Return type** int

**alloc\_offset**(*size: int*) → int

Allocate an offset.

This is used by structures to assign an offset to each field.

**Parameters** **size** (*int*) – Number of bytes to allocate

**Returns** The allocated offset.

**Return type** int

**dump\_table**(*level: int*) → str

Dump the structure to a table.

This will dump all the structures of this sub-section into a table.

**Parameters** **level** (*int*) – Dump level

**Returns** The table.

**Return type** str

**get\_current\_offset**() → int

Get the current offset.

**Returns** The current offset.

**Return type** int

**get\_image**()

Dump the content of the sub-section in binary form to the specified file.

**Parameters** **fd** – File descriptor

**pack**() → bytes

Pack all fields in binary form.

This can be used to dump all the fields to a file.

**Returns** The values of the fields packed into a byte array.

**Return type** bytes

**class** *gapylib.utils.CStructScalar*(*name: str, size: int, value: any, offset: int*)

Class for scalar fields.

**dump\_table**(*table: prettytable.PrettyTable, level: int*)

Dump the field to a table.

This declaration is propagated towards the parents so that the runner taking care of the flash image generation knows where the flash are.

**Parameters**

- **x** (*PrettyTable*) – Table where the field should be dumped.

- **level** (*int*) – Dump level

## Flash sections

### gapylib.fs.partition

Provides section template for flash partitions

**class gapylib.fs.partition.PartitionTableHeader**(*name*: str, *parent*: gapylib.utils.CStructParent)

Class for generating partition table sub-section containing the main header. This header contains global information like number of partitions.

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

**class gapylib.fs.partition.PartitionTableSection**(*parent*, *name*: str, *section\_id*: int)

Class for generating a readfs section.

**parent**

Name of the section.

**Type** gapylib.flash.Flash

**name**

Name of the section.

**Type** str

**section\_id**

Id of the section.

**Type** int

**finalize()**

Finalize the section.

This can be called to set internal section fields which requires some knowledge of the offset or size of other sections. The structure of the section should not be changed in this step

**is\_empty()**

Tell if the section is empty.

This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the section is empty, False otherwise.

**Return type** str

**set\_content**(*offset*: int, *content\_dict*: dict)

Set the content of the section.

**Parameters**

- **offset** (int) – Starting offset of the section.
- **content\_dict** (dict) – Content of the section

```
class gapylib.fs.partition.PartitionTableSectionHeader(name: str, parent:  
                                                    gapylib.utils.CStructParent)
```

Class for generating partition table sub-section containing a section header. This header contains section information like size.

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

**gapylib.fs.readfs**

Provides section template for readfs, to generate a readfs image

```
class gapylib.fs.readfs.ReadfsFile(name: str, size: int, parent: gapylib.utils.CStructParent)
```

Class for generating readfs sub-section containing a file content.

**name**

Name of this sub-section.

**Type** str

**size**

File size.

**Type** int

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

```
class gapylib.fs.readfs.ReadfsFileHeader(name: str, name_len: int, parent: gapylib.utils.CStructParent)
```

Class for generating readfs sub-section containing a file header. This header contains information about the file like its name and size.

**name**

Name of this sub-section.

**Type** str

**name\_len**

Len of the file name, including the terminating 0.

**Type** int

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

```
class gapylib.fs.readfs.ReadfsHeader(name: str, parent: gapylib.utils.CStructParent)
```

Class for generating readfs sub-section containing the main header. This header contains global information like number of files.

**name**

Name of this sub-section.

---

**Type** str

**parent**  
Parent, which is aggregating all readfs sub-sections.

**Type** str

**class** `gapylib.fs.readfs.ReadfsSection`(*parent*: gapylib.flash.Flash, *name*, *section\_id*: int)

Class for generating a readfs section.

**parent**  
Name of the section.

**Type** `gapylib.flash.Flash`

**name**  
Name of the section.

**Type** str

**section\_id**  
Id of the section.

**Type** int

**get\_partition\_type()** → int  
Return the partition type.  
This information can be used by the partition table as the subtype. This method returns an unknown type (0xff) and should be overloaded by real sections.

**Returns** The partition type.

**Return type** int

**is\_empty()**  
Tell if the section is empty.  
This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the section is empty, False otherwise.

**Return type** str

**set\_content**(*offset*: int, *content\_dict*: dict)  
Set the content of the section.

**Parameters**

- **offset** (int) – Starting offset of the section.
- **content\_dict** (dict) – Content of the section

**gapylib.fs.littlefs**

Provides section template for lfs, to generate a lfs image

**class** `gapylib.fs.littlefs.LfsHeader`(*name*: str, *parent*: gapylib.utils.CStructParent, *size*: int)

Class for generating readfs sub-section containing the main header. This header contains global information like number of files.

**name**  
Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

**class** gapylib.fs.littlefs.LfsSection(*parent*: gapylib.flash.Flash, *name*, *section\_id*: int)

Class for generating a lfs section.

**parent**

Name of the section.

**Type** gapylib.flash.Flash

**name**

Name of the section.

**Type** str

**section\_id**

Id of the section.

**Type** int

**get\_partition\_type()** → int

Return the partition type.

This information can be used by the partition table as the subtype. This method returns an unknown type (0xff) and should be overloaded by real sections.

**Returns** The partition type.

**Return type** int

**is\_empty()**

Tell if the section is empty.

This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the section is empty, False otherwise.

**Return type** str

**set\_content**(*offset*: int, *content\_dict*: dict)

Set the content of the section.

This can be called to set the default content, and may be overwritten by gapy executable if a new content has been specified. The section should in this step declare all its internal structure so that the offsets and sizes of all sections are known after this step.

**Parameters**

- **offset** (int) – Start offset of the section
- **content\_dict** (dict) – Content of the section

## gapylib.fs.hostfs

Provides section template for hostfs, to copy host files to work folder.

**class** `gapylib.fs.hostfs.HostfsSection`(*parent*: `gapylib.flash.Flash`, *name*, *section\_id*: `int`)

Class for generating a hostfs section.

**parent**

Name of the section.

**Type** `gapylib.flash.Flash`

**name**

Name of the section.

**Type** str

**section\_id**

Id of the section.

**Type** int

**set\_content**(*offset*: `int`, *content\_dict*: `dict`)

Set the content of the section.

**Parameters**

- **offset** (`int`) – Starting offset of the section.
- **content\_dict** (`dict`) – Content of the section

## gapylib.fs.raw

Provides section template for raw partition.

**class** `gapylib.fs.raw.RawHeader`(*name*: str, *parent*: `gapylib.utils.CStructParent`, *size*: `int`)

Class for generating readfs sub-section containing the main header. This header contains global information like number of files.

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all readfs sub-sections.

**Type** str

**class** `gapylib.fs.raw.RawSection`(*parent*: `gapylib.flash.Flash`, *name*, *section\_id*: `int`)

Class for generating a raw section.

**parent**

Name of the section.

**Type** `gapylib.flash.Flash`

**name**

Name of the section.

**Type** str

**section\_id**

Id of the section.

**Type** int

**get\_partition\_type()** → int

Return the partition type.

This information can be used by the partition table as the subtype. This method returns an unknown type (0xff) and should be overloaded by real sections.

**Returns** The partition type.

**Return type** int

**set\_content**(*offset*: int, *content\_dict*: dict)

Set the content of the section.

#### Parameters

- **offset** (int) – Starting offset of the section.
- **content\_dict** (dict) – Content of the section

## Targets

### gap9.evk

Provides the gapy description of the GAP9 EVK board.

**class** gap9.evk.Target(*options*: list)

GAP9 EVK board generator

This just declares to gapy the 2 supported flash (external and mram) and inherit from gap common target the supported platforms and commands.

**Parameters** **offset** (list) – List of options.

### gap9.gap9\_v2

Provides the gapy description of the GAP9 EVK board.

**class** gap9.gap9\_v2.Target(*options*: list)

GAP9 EVK board generator

This just declares to gapy the 2 supported flash (external and mram) and inherit from gap common target the supported platforms and commands.

**Parameters** **offset** (list) – List of options.

## Base classes for Gap targets

### gapylib.chips.gap.flash

Default flash description for all gap targets

**class** gapylib.chips.gap.flash.DefaultFlashRomV2(*target*: gapylib.target.Target, *name*: str, *size*: int, \**kargs*, \*\**kwargs*)

Default class for all flash for gap targets. Mostly describes the allowed section templates (rom and all FS).

**target**  
Target containing the flash.

**Type** `gapylib.target.Target`

**name**  
Name of the flash

**Type** str

**size**  
Size of the flash

**Type** int

## `gapylib.chips.gap.rom_v2`

Provides section template for gap rom v2 (starting from gap9\_v2)

**class** `gapylib.chips.gap.rom_v2.Binary`(*file\_desc: BinaryIO*)  
Class for describing an ELF binary.

**fd**  
File descriptor

**class** `gapylib.chips.gap.rom_v2.BinarySegment`(*base: int, data: bytes*)  
Class for describing an ELF segment.

**base**  
Name of this sub-section.

**Type** int

**data**  
Content of the section

**Type** bytes

**class** `gapylib.chips.gap.rom_v2.RomEmptyHeader`(*name, parent*)  
Class for generating rom sub-section containing the main header when the ROM section is empty (when there is no binary)

**name**  
Name of this sub-section.

**Type** str

**parent**  
Parent, which is aggregating all rom sub-sections.

**Type** str

**class** `gapylib.chips.gap.rom_v2.RomFlashSection`(*parent: gapylib.flash.Flash, name: str, section\_id: int*)  
Class for generating a ROM section.

**parent**  
Name of the section.

**Type** `gapylib.flash.Flash`

**name**  
Name of the section.

**Type** str

**section\_id**

Id of the section.

**Type** int

**finalize()**

Finalize the section.

This can be called to set internal section fields which requires some knowledge of the offset or size of other sections. The structure of the section should not be changed in this step

**is\_empty()**

Tell if the section is empty.

This is used by the target to know if the flash should be updated in auto mode.

**Returns** True if the section is empty, False otherwise.

**Return type** str

**set\_content(offset: int, content\_dict: dict)**

Set the content of the section.

**Parameters**

- **offset** (int) – Starting offset of the section.

- **content\_dict** (dict) – Content of the section

**class gapylib.chips.gap.rom\_v2.RomHeader(name, parent: gapylib.utils.CStructParent)**

Class for generating rom sub-section containing the main header when the ROM section is not empty (when there is a binary)

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all rom sub-sections.

**Type** str

**class gapylib.chips.gap.rom\_v2.RomSegment(name, size, parent: gapylib.utils.CStructParent)**

Class for generating rom sub-section containing a segment.

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all rom sub-sections.

**Type** str

**class gapylib.chips.gap.rom\_v2.RomSegmentHeader(name, parent: gapylib.utils.CStructParent)**

Class for generating rom sub-section containing a segment header.

**name**

Name of this sub-section.

**Type** str

**parent**

Parent, which is aggregating all rom sub-sections.

**Type** str

**class** `gapylib.chips.gap.rom_v2.Xip(flash: gapylib.flash.Flash)`

Class for handling XIP support for ROM

**flash**  
Flash where the ROM section is.

**Type** `Flash`

**fill\_header(header: gapylib.chips.gap.rom\_v2.RomHeader)**  
Fill XIP information into ROM main header.

**Parameters** `header (RomHeader)` – Name of the field

**get\_page\_size()** → int  
Get XIP page size.

**Returns** Page size.

**Return type** int

**is\_xip\_segment(base: int)** → bool  
Tell if the segment is an XIP one.

**Returns** True if the segment is an XIP one, False otherwise.

**Return type** bool

**register\_segment(offset: int, size: int)**  
Register XIP segment.

This is used to compute the size of the XIP area.

**Parameters**

- **offset (int)** – Base offset of the segment.
- **size (int)** – Size of the segment.

## `gapylib.chips.gap9_v2.gap.board_runner`

Helper for running on gap9\_v2 boards

**class** `gapylib.chips.gap.gap9_v2.board_runner.Runner(target: gapylib.target.Target)`

Helper class for running execution on gap9\_v2 boards.

**target**  
The target on which the execution is run.

**Type** `gapylib.target.Target`

**static append\_args(parser: argparse.ArgumentParser)**  
Append runner specific arguments to gapy command-line.

This is used to append arguments only if the RTL platform is selected.

**Parameters** `parser (argparse.ArgumentParser)` – The parser where to add arguments.

**flash()**  
Handle the gapy command ‘flash’ to upload the flash images.

**parse\_args(args: any)**  
Handle arguments.

This will mostly use the arguments to prepare the platform command.

**Parameters args** – The arguments.

### **run()**

Handle the gapy command ‘run’ to start execution on the platform.

## 6.6 NNTool

### 6.6.1 Introduction

The NNTool is a graph manipulation tool capable of reading a high-level computational graph description mixing DSP and NN operators and mapping it to a set of Autotiler nodes (named Generators, provided in the GAP SDK, look at the autotiler docs for more details) that it will use to generate optimized GAP C code.

NNTool is a Python library with a numpy backend to run the provided computational graph in both floating and fixed point arithmetic, bit accurate wrt GAP fixed point inference.

The process that the NNTool applies to a provided Neural Network is the following:

1. Load the model description
2. Quantize the graph (if not already, i.e. load tflite quantization) through tensor statistics collection with floating point inference over a calibration dataset
3. Map the original operators into Autotiler operators, i.e. match tensor order, insert reshape/transposes where needed, fuse nodes if possible
4. Generate the Autotiler Model

From a user perspective, NNTool can be used both interactively in command line or directly importing it as a package in a Python script. The graphs provided to NNTool must be TFLite or ONNX models.

NOTE: Not all the operands from the two formats are supported, for a complete list open the nntool command line and type:

### 6.6.2 Command Line Usage

To open NNTool in a command line interactive interface, after sourcing the GAP SDK:

```
nntool
```

#### Open a Model

As mentioned above, the model can be .tflite or .onnx, the -q option allows the user to load the quantization information already present in some tflite model (only if the tflite flatbuffer has been converted with quantization options)

```
open path/to/model.tflite [-q]
```

## Visualize the Graph

At any time, the graph can be visualized with:

```
draw          # Render the graph in a single image
show         # Display the structure of the graph in a literal way
qshow        # Display the quantization structure of the graph in a literal way
```

## Adjust and Fusions

To match the Autotiler Generators the structure of the graph might need a restructure, for example the convolutions by default are handled in CHW tensor order by the backend library of Autotiler, hence the adjust reorders the tensors to match the Autotiler backend. Also, some layer sequences can be merged together to be more efficient in the final target, i.e. Conv+Pool+ReLU can be a single node: fusions spots those structures. According to the type of quantization you want to apply, the set of fusions can change:

- scale8: for 8 bits scaled quantization (both SW or NE16) and Float16
- pow2: for 16 bits POW2 quantization

```
adjust
fusions [--scale8 or --pow2]
```

If the graph contains standalone piecewise DSP operations such as div/mul/add/pow/sqrt/log NNTool can also wrap them in fused structures and will generate specific layers for them. This last step is not automatically run, if required it will be prompted in further steps:

```
fusions -a expression_matcher
```

## NNTool Inference

In the following sections will be presented several commands that use the NNtool inference engine. Any time the user run inference on NNTool over inputs coming from files, he/she must be sure that the input shapes match the graph inputs, i.e. if the network has been adjusted to CHW input should match this order. Inputs can be numpy tensor, images (opened with Pillow), binary (.dat) or audio files (.wav).

NOTE: Regardless the type of inference (quantized or float), NNTool expects data in their floating point real representation, i.e. if a network has been trained with images normalized to [-1:1] the data must be normalized as well before feeding NNTool.

To make this input preprocessing more useful from user perspective (especially with images where they natively come in [0:255] format) there are these options:

```
INPUT_OPTIONS:
-B, --bit_shift BIT_SHIFT
    # shift input before offset and divisor. Negative for right shift.
-D, --divisor DIVISOR
    # divide all input data by this value
-O, --offset OFFSET
    # offset all input data by this value
```

(continues on next page)

(continued from previous page)

```

-H, --height HEIGHT
    # resieze image height to this value
-W, --width WIDTH
    # resieze image width this value
-T, --transpose
    # Used only on images, it swap the channels position (from CHW to HWC or viceversa)
-F, --nptype
    # Interpret pixels as this numpy type
-M, --mode {1, L, P, RGB, RGBA, CMYK, YCbCr, LAB, HSV, I, F}
    # mode to import image in
-N, --norm_func "x:x/128"
    # lambda function to apply on input in the form x: fn(x)
--rgb888_rgb565
    # convert 3 channel 8bits input into 1 channel 16bit rgb565

```

These options can be also set on the entire shell, thus they will be applied without having to specify every time:

```

set image_height HEIGHT
    # resieze image height to this value
set image_mode {1, L, P, RGB, RGBA, CMYK, YCbCr, LAB, HSV, I, F}
    # mode to import image in
set image_width WIDTH
    # resieze image width this value
set input_divisor DIVISOR
    # divide all input data by this value
set input_norm_func "x:x/128"
    # lambda function to apply on input in the form x: fn(x)
set input_offset OFFSET
    # offset all input data by this value

```

## Quantization

To quantize the Model, NNTool uses a numpy float32 backed to run inference on the graph and collect the tensors statistic (min/max). If the original graph was a TFLite quantized model, these information have been collected at loading time.

With these information NNTool uses an algorithm to propagate the quantization information through the graph, for example it handles the fact that a concat layer needs to have the same quantization in all its inputs.

At this stage the user can also choose different types of executions, this might affect the quantization types, i.e. NE16 requires asymmetric unsigned 8 bits while SW symmetric signed 8 bits. The type of execution can be selected graph-wise or layer-wise giving the graph step numbers related to the nodes the user wants to quantize in a certain way.

```

QUANT_OPTIONS:
    --scheme {SQ8, POW2, FLOAT}
        # quantize with scaling factors
        # (TFLite quantization-like) [default] or POW2
    --float_type {bfloat16, float16, ieee16, float32, ieee32}
        # If scheme=FLOAT it selects the type of datatype
    --hwc
        # Use HWC SW Kernel instead of CHW (only with SQ8)

```

(continues on next page)

(continued from previous page)

```
--use_ne16
    # Enable use of NE16 kernels (if supported)
--weight_bits {2, 3, 4, 5, 6, 7, 8}
    # (Only available on NE16 layers)
--force_external_size {8, 16}
    # number of bits to use for outputs features and RNN states
    # (only available in NE16)
--force_input_size {8, 16}
    # number of bits to use for input features
    # (only available in NE16)
--softmax_out_8bits
    # make the output scale8 8 bits (default 16bits)
--clip_type
    # Clipping method for any node that modifies its input:
    # none (default) - the minimum and maximum observed values are used.
    # laplace, gaus - Values chosen based on laplace or gaussian distribution
    # mix - MSE is used to estimate if distribution is laplace or gaussian
    # whiskers - 1.5 * inter quartile range
    # std3, std5 - 3 or 5 times standard deviation from mean
```

```
aquant path/to/calibration/samples* [INPUT_OPTIONS] [QUANT_OPTIONS]
    --json JSON_PATH
        # Quant options can be stored in a json dictionary and provided here
    --stats STATS_PATH
        # Instead of running calibration in nntool, statistics can be
        # passed as a pickle

# If no calibration data are available or the target quantization is float (no statistic
# needed)
fquant [QUANT_OPTIONS] --json JSON_PATH
```

If the model is already quantized, i.e. the statistics are already in the graph, the user can change the quantization options with:

```
qtune --step STEP KEY1=VALUE1 KEY2=VALUE2 [...]
    # STEP: single step, a set of steps or a range to which apply
    # following QUANT_OPTIONS:
    # --step 1 use_ne16=True force_external_size=16
    # --step 2,3 hwc=True
    # --step 4:-1 use_ne16=True
    # or to apply to the whole graph:
    # --step * use_ne16=True
    --json JSON_PATH
        # Quant options can be stored in a json dictionary and provided here
```

To save the current quantization options in a dictionary and later manually edit and pass to qtune or aquant --json:

```
qtunesave --json JSON_PATH
```

NOTE: after running qtune/aquant might be necessary to run adjust/fusions again (the tensor order might be changed)

## Run Inference and Compare

Once the model has been quantized the user can run inference with a bit accurate quantized backend with respect to the platform execution. In this way the user can assert any accuracy drop in the quantization process.

```
dump path/to/files* [INPUT_OPTIONS]
    -q
    # Quantized mode (without -q it runs float32)
    -d
    # if quantized mode: dequantize the results such that
    # they are represented in the real space
    -S TENSOR_NAME
    # save the results in the workspace with TENSOR_NAME
    -P TENSOR_PICKLE
    # save the results in a pickle file

qerror path/to/files* [INPUT_OPTIONS]
    # Automatically run in float and quantized and
    # compares layer by layer statistics

# To compare two set of dump runs manually (i.e. float vs quantized)
tensors -t TENSOR_NAME1 TENSOR_NAME2
    -s 10
    # To compare outputs of step 10, otherwise all
    -Q
    # QSNR
    -E
    # Absolute error
```

If the model has been generated with the Autotiler graph option DUMP\_TENSOR, the log of the run will contain the outputs of the layers. NNTool can load the so dumped tensors with:

```
tensors --gap_load path/to/logfile.txt
```

## Attach Special Nodes in front of the Model

Sometimes you want to apply standard operations to the input of a network which prepare the data coming from a sensor to the input of the NN. NNTool supports few of these possible input preprocessors: imageformat: attach a layer that can do image transposition HWC -> CHW on the cluster with offset -128 in int8 or automatically output 16bits, useful in POW2 16bits models resize: attach a resizer to the input of the network, useful for example if the network is trained with HxW images but the sensor uses IN\_HxIN\_W

```
imageformat INPUT_NODE
    # Node to attach the formatter
    IN_FORMAT {bw16, bw8, rgb16, rgb565_rgb888, rgb888}
    # Data Type coming in
    NORMALIZATION {offset_int8, out_int16, shift_int8}
        # Operation applied to each op:
        #     - offset_int8: subtracts 128 to every pixel [0:255]->[-128:127]
        #     - out_int16: outputs int16 data
        #     - shift_int8: shift >>1 every pixel: [0:255]->[0:127]

input_resizer INPUT_NODE
```

(continues on next page)

(continued from previous page)

```

# Node to attach the resizer
RESIZER_TYPE {bilinear, nearest}
    # Algo used by the resizer
    --from_w IN_W
    --from_h IN_H
        # Width and Height of the original input resized to the INPUT_NODE shape

```

AUDIO DSP: dsp\_preprocessing: attach a DSP preprocessing which can be of type Mfcc or RFFT. It computes the operation to overlapped windows of the inputs. NOTE: The Mfcc can do melspectrogram/logmelspectrogram/mfcc but no normalization is applied to the output of the DSP node. If needed it should be included as part of the model after the input.

```

logmel_config.json:
{
    "window": "hanning",           # Window numpy function
    "pad_type": "center",          # How to apply padding if frame_size < n_fft
    "n_fft": 512,                 # Number of FFT bins
    "n_frames": 401,               # Number of frames (it must match the graph input size)
    "frame_size": 512,              # Number of samples in each frame
    "window_size": 400,             # Number of samples for the window function (if < frame_
    ↪size the window will be center padded)
    "frame_step": 160,              # Number of samples between two consecutive frames
    "sample_rate": 16000,            # Samplerate
    "magsquared": True,             # If True, the spectrogram is calculated as np.abs(rfft)**
    ↪** 2, else np.abs(rfft)
    "n_fbanks": 128,                # Number of Mel Filterbanks (ignored when dsp_node_type==
    ↪"fft")
    "fbank_type": "librosa",         # Algorithm used to generate the Mel Filters
    "mel_filterbanks_file": None,   # Path to melfilters .npy array provided by user, in_
    ↪this case fbank_type will be ignored
    "n_dct": 0,                     # Number of DCT bins in case of Mfcc preprocessing
    "dct_type": 2,                  # DCT algorithm type
    "fmin": 0.0,                   # Fmin used to calculate the Mel Filters
    "fmax": None,                  # Fmax used to calculate the Mel Filters
    "log_type": "natural",          # Logarithm type ("None", "db" or "natural"), None will_
    ↪output melspectrogram
    "log_offset": 1e-6,              # Offset applied before log function
}

dsp_preprocessing INPUT_NODE
    # Node to Attach the DSP node
    {MFCCPreprocessingParameters, RFFT2DPreprocessingParameters}
        # Type of preprocessing
        --config_json logmel_config.json
            # Config file with all the parameters

```

The code above emulates the librosa equivalent of:

```

import librosa
melspect = librosa.feature.melspectrogram(y=audio,
                                            sr=16000,
                                            n_fft=512,

```

(continues on next page)

(continued from previous page)

```

        hop_length=160,
        win_length=400,
        window="hann",
        center=False # This is important to not pad
    ↵the whole audio input
)
logmelspect = np.log(melspect + 1e-6)

```

HINT: If using GAP9 the preprocessing in float16 is equally fast in terms execution time as the fixed point, hence we suggest to qtune this layer to be float16 to get better accuracy:

```
qtune --step MfccPreprocessing scheme=float float_type=float32
```

NOTE: This feature is advanced and might not work in every possible configuration. To have a more flexible feature we suggest to use directly the autotiler Generator for MFCC/RFFT. You can find examples in the sdk/examples

## Save NNTool State

After several commands are run, you can save the nntool state in a json file. Save the state of the transforms and quantization of the graph. This state file can be used to generate the model file as part of a build script. If no argument is given then the state files will be saved in the same directory as the graph. If a directory is given then the state files will be saved in it with the graph basename. If a filename is given, its basename will be used to save the state files.

```
save_state FILE
```

It basically saves the sequence of commands run and the quantization statistics.

## Generate Autotiler Code

Once the model has been quantized and fused, the Autotiler model can be generated:

```

gen AT_MODEL
    -t, --output_tensors
    # write constants (weights, biases)
    -T, --tensor_directory TENSOR_DIRECTORY
    # path to tensor directory. full path will be created
    # if it doesn't exist. If this parameter is given it
    # will update the settings saved with the graph state.
    -M, --model_directory MODEL_DIRECTORY
    # path to model directory. full path will be created
    # if it doesn't exist. If this parameter is given it
    # will update the settings saved with the graph state.
    --basic_kernel_source_file BASIC_KERNEL_SOURCE_FILE
    # file to write to, otherwise output to terminal
    --basic_kernel_header_file BASIC_KERNEL_HEADER_FILE
    # file to write to, otherwise output to terminal

```

## Scripting Mode

All the commands to prepare a graph for Autotiler can be stored in a script file and run with:

```
nntool -s nntool_script
```

To generate the code directly from an nntool state:

```
nntool -g AT_MODEL_PATH
    -M, --model_directory MODEL_DIRECTORY
        # path to model directory. full path will be created
        # if it doesn't exist. If this parameter is given it
        # will update the settings saved with the graph state.
    -m AT_MODEL_NAME
        # file name for autotiler model
    -T TENSOR_DIRECTORY
        # path to tensor directory. full path will be created
        # if it doesn't exist.
```

After having prepared the graph with some commands, the NNTool can also generate a sample project, with all files dependencies and a simple main application which runs on the platform the model with unset input data. The model used in the project generated will be prepared with a script written from the history of the current NNTool shell:

```
gen_project PROJECT_FOLDER
    -o, --overwrite
        # overwrite existing files
```

Similarly the performance command will generate a project and run the application from the python interface, getting performance results out of gvsoc run:

```
performance
```

## Autotiler Optimization options

NOTE: All the following options can be manually overwritten in the generated Autotiler Model. They don't affect anyhow the NNTool execution.

Node Options:

```
nodeoption STEP OPTION=VAL
```

OPTION:

- PARALLELFEATURES: In CHW mode convolution can be selected in with parallelization done in the features (default) or spatial dimension
- ALLOCATE: input/output can be allocated in memory by the autotiler. This will spare L2 memory for other data in the application
- TILEORIENTATION: tile computed vertically or horizontally (default)

Graph Options:

```
set graph_const_exec_from_flash true/false
    # If on, constant symbols executes from home location
set graph_dump_one_node NODE_NAME
```

(continues on next page)

(continued from previous page)

```

# Trace one specific graph node (name of the autotiler not NNTool)
set graph_dump_tensor NUM
    # {4: output, 6: input and output, 7: input, output, constants}
    # Trace selected tensors arguments at inference time, either all nodes or selected
    ↵node
set graph_monitor_cvar_name NAME
    # When monitor cycles is on name of the C var array to receive results
set graph_monitor_cycles true/false
    # Enable automatic cycle capture for each node of the graph
set graph_name NAME
    # name of the graph used for code generation
set graph_noinline_node true/false
    # If on, all user kernel function are marked as noinline
set graph_pref_l3_exec {AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_L3_QSPIRAM, AT_MEM_L3_
    ↵DEFAULTRAM}
    # In case a symbol must be allocated in L3 for execution this is the prefered memory
set graph_pref_l3_home {AT_MEM_L3_HRAM, AT_MEM_L3_OSPIFLASH, AT_MEM_L3_QSPIFLASH, AT_MEM_
    ↵L3_HFLASH, AT_MEM_L3_MRAMEFLASH, AT_MEM_L3_OSPIRAM, AT_MEM_L3_QSPIRAM, AT_MEM_L3_
    ↵DEFAULTRAM, AT_MEM_L3_DEFAULTFLASH}
    # For constant symbols which L3 flash prefered memory
set graph_produce_node_cvar_name true/false
    # When producing node names is on name of the C array receiving the names as strings
set graph_produce_node_names true/false
    # Enable production of an array containing the name of each graph node
set graph_produce_operinfos true/false
    # Enable production of number of macs for each layer
set graph_produce_operinfos_cvar_name true/false
    # When Number of oper Infos is on name of the C array receiving mac infos for each
    ↵node
set graph_reorder_constant_in true/false
    # Enable reordering of constant inputs in order to transform 2D accesses into 1D_
    ↵accesses
set graph_size_opt OPT
    # 0: Default make opt, 1: 02 for all layers, 0s for xtor,dxtor,runner, 2: 0s for
    ↵layers and xtor,dxtor,runner
set graph_trace_exec true/false
    # Enable trace of activity
set graph_warm_construct true/false
    # Generate construct/destruct functions with the Warm option to only allocate/
    ↵deallocate L1 buffer

```

Memory Options:

```

set l1_size BYTE_SIZE
set l2_size BYTE_SIZE
set l3_size BYTE_SIZE
set l3_flash_mb MBYTE_SIZE
    # Set the different memory levels size, used to generate the default Autotiler model
    ↵Memories.
    # All of them can be overriden when calling the Autotiler
set l2_ram_ext_managed true/false
    # Whether to let the autotiler allocate the L2 buffer by itself, in the model
    ↵constructor (default: true)

```

(continues on next page)

(continued from previous page)

```

set l3_flash_device {AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_L3_QSPIRAM}
    # Which type of device will be used for the read/write memory level 3 device in the
    ↵model. It will change the used driver to open the L3 RAM (default: AT_MEM_L3_HRAM)
set l3_flash_ext_managed true/false
    # Whether to let the autotiler open the external L3 Flash and allocate the L3 Flash
    ↵buffer by itself or the user will have to do it in the application, in the model
    ↵constructor (default: true)
set l3_ram_device {AT_MEM_L3_HFLASH, AT_MEM_L3_OSPIFLASH, AT_MEM_L3_QSPIFLASH, AT_MEM_L3_
    ↵MRAMFLASH, AT_MEM_L3_DEFULTRAM}
    # Which type of device will be used for the read-only memory level 3 device in the
    ↵model. It will change the used driver to open the L3 FLASH (default: AT_MEM_L3_HFLASH)
set l3_ram_ext_managed true/false
    # Whether to let the autotiler open the external L3 and allocate the L3 buffer by
    ↵itself or the user will have to do it in the application, in the model constructor
    ↵(default: true)
set default_input_exec_location {AT_MEM_L2, AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_L3_
    ↵QSPIRAM, AT_MEM_L3_DEFULTRAM}
    # Where the autotiler should assume the input to be executed from (default: AT_MEM_
    ↵L2)
set default_input_home_location {AT_MEM_L2, AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_L3_
    ↵QSPIRAM, AT_MEM_L3_DEFULTRAM}
    # Where the autotiler should assume the input to be located to (default: AT_MEM_L2)
set default_output_exec_location {AT_MEM_L2, AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_
    ↵L3_QSPIRAM, AT_MEM_L3_DEFULTRAM}
    # Where the autotiler should assume the output to be executed from (default: AT_MEM_
    ↵L2)
set default_output_home_location {AT_MEM_L2, AT_MEM_L3_HRAM, AT_MEM_L3_OSPIRAM, AT_MEM_
    ↵L3_QSPIRAM, AT_MEM_L3_DEFULTRAM}
    # Where the autotiler should assume the output to be located to (default: AT_MEM_L2)

```

### 6.6.3 NNTool Python API

#### NNTool Python API Introduction

The NNTool Python API exposes all of the capabilities in the command line interface and more. It is very useful for more complicated experiments and graph validation scenarios.

The main interface to the NNTool API is the `nntool.api.NNGraph` class. The `NNGraph.load_graph()` static method is used to load a graph. It returns an instance of `NNGraph`.

If a method on a class is not documented here then you should consider it private and subject to change.

Here are some of the relationships between the command line interface and the `NNGraph` methods.

- `open` - `load_graph()`
- `adjust` - `adjust_order()`
- `fusions` - `fusions()`
- `aquant` and `qtune` - `collect_statistics()` and `quantize()`
- `dump` - `execute()`

## Selecting Nodes

The API refers to operators in the graph as nodes. All nodes inherit from the `nntool.api.types.NNNodeBase` class.

The `NNGraph.nodes()` method can be used to select nodes by node classes. The `NNGraph` instance can be indexed by step index or node name to provide the instance of a particular node in the graph.

e.g.

```
from nntool.api import NNGraph
from nntool.api.types import Conv2DNode

G = NNGraph.load_graph('mygraph.tflite')

# get node by step index
print(G[12].name)

# get node by name
print(G["my_conv"].step_idx)

# get nodes by class. node_classes can be a single class or a tuple of classes
for node in G.nodes(node_classes=Conv2DNode):
    print(node.name)
```

## Setting Node Options

Most node options are set on nodes through the `at_options`. The options are identical to those that you can set using the `nodeoptions` command and are exposed as properties of the `NodeOptions` class returned from the `at_option` property.

e.g.

```
from nntool.api import NNGraph

G = NNGraph.load_graph('mygraph.tflite')

G[12].at_options.dump_tensors = 1
```

They can also be set and read via the `nntool.api.NNGraph.set_node_option()` and `nntool.api.NNGraph.get_node_option()` methods.

## Collecting Statistics

To quantize a graph NNTool needs to have statistics on the dynamic of each activation. If the graph is imported with the `load_quantization` parameter set to True (i.e. a quantized TFLITE graph or NNCF quantized ONNX graph) then these statistics will already be present. If not you need to collect them.

The `nntool.api.NNGraph.collect_statistics()` method is used for this. It needs a dataloader which is just an iterable that returns either normalized input tensors (as numpy float arrays) if there is one input to the graph or a sequence of tensors if there is more than one input array. `nntool.api.FileImporter` is an example of a dataloader for sounds and images.

Sometimes you just want to see what the performance of a network will be and are not worried about accuracy. In this case the `nntool.api.RandomIter` class can be used.

e.g.

```

from nntool.api import NNGraph
from nntool.api.utils import RandomIter

G = NNGraph.load_graph('mygraph.tflite')

stats = G.collect_statistics(RandomIter.fake(G))

```

Here is an example of a custom datacollector:

```

from PIL import Image
import numpy as np

class MyDataLoader():
    def __init__(self, image_files, max_idx=None, transpose_to_chw=True):
        self._file_list = image_files
        self._idx = 0
        self._max_idx = max_idx if max_idx is not None else len(image_files)
        self._transpose_to_chw = transpose_to_chw

    def __iter__(self):
        self._idx = 0
        return self

    def __next__(self):
        if self._idx > self._max_idx:
            raise StopIteration()
        filename = self._file_list[self._idx]

        # Here we read the image and make it a numpy array
        image = Image.open(filename)
        img_array = np.array(image)

        # Apply some preprocessing
        img_array = img_array / 128 - 1.0

        # Transpose to CHW
        if self._transpose_to_chw:
            img_array = img_array.transpose(2, 0, 1)

        self._idx += 1
        return img_array

```

If you don't want to redo the statistics cache them to disk with `numpy.save` and `numpy.load`.

## Quantization Options

Quantization options can be set at graph or node level using the `node_options` or `graph_options` optional arguments to the `quantize()` method. Graph options are just a dict of option name and value. Node options are a dictionary of node name to dictionary of options. There is a helper function to create the option dictionary `nntool.api.utils.quantization_options()`.

e.g.

```
from nntool.api import NNGraph
from nntool.api.utils import quantization_options

G = NNGraph.load_graph('mygraph.tflite')

# ... generate stats

G.quantize(
    stats,
    node_options={
        'conv1': node_options(weight_bits=2)
    },
    schemes=['scaled'],
    graph_options=node_options(use_ne16=True)
)
```

## Executing a graph on the simulator or chip

A loaded graph can be used to build a project and execute the graph on the GVSOC SoC simulator or actually on the chip directly from the API. The performance data from the execution of the graph can be returned as can all of the tensors from the execution of the graph. All this functionality is provided through the `NNGraph execute_on_target()` method.

## Longer example of NNTool Python API

Here is a longer example of some of the functionality in the NNTool Python API.

```
from nntool.api import NNGraph

model = NNGraph.load_graph(
    'face_detection_front.tflite',
    load_quantization=False # Whether tflite quant should be loaded or not
    ↵(default: False)
)

# Model show returns a table of information on the Graph
print(model.show())

# Model draw can open or save a PDF with a visual representation of the graph
model.draw()

# The equivalent of the adjust command
```

(continues on next page)

(continued from previous page)

```

model.adjust_order()

# The equivalent of the fusions --scale8 command. The fusions method can be given a
→series of fusions to apply
# fusions('name1', 'name2', etc)
model.fusions('scaled_match_group')

# draw the model here again to see the adjusted and fused graph
model.draw()

# The executer returns all the layer output. Each layer output is an array of the
→outputs from each output of a layer
# Generally layers have one output but some (like a split for example) can have multiple
→outputs
# Here we select the first output of the last layer which in a graph with one output
→will always be the the
# graph output
data_loader = MyDataLoader(glob("input_images/*"))
layer_outputs = model.execute(data_loader)
last_layer = layer_outputs[-1][0]

# Now let's quantize the graph
statistics = model.collect_statistics(data_loader)
# The resulting statistics contain detailed information for each layer
statistics['input_1']

name_layer_2 = model[2].name
# quantize the model. quantization options can be supplied for a layer or for the whole
→model
model.quantize(
    statistics,
    schemes=['scaled'], # Schemes present in the graph
    graph_options={
        "use_ne16": True,
        "hwc": False,
        "force_output_size": 16,
    }, # QUANT_OPTIONS applied graph-wise
    node_options={
        name_layer_2: {
            "use_ne16": False,
            "hwc": True
        }
    }, # QUANT_OPTIONS applied layer-wise
)
test_image = next(data_loader)

# Now execute the quantized graph outputting quantized values
print("execute model without dequantizing data")
print(model.execute(test_image, quantize=True)[-1][0])

# Now execute the graph twice with float and quantized versions and compare the results
print("execute model comparing float and quantized execution and showing Cosine
→Similarity")

```

(continues on next page)

(continued from previous page)

```

cos_sim = model.cos_sim(model.execute(test_image), model.execute(test_image, ↴
    ↴quantize=True, dequantize=True))
print(cos_sim)

# the step idx can be used to index the model to find the layer with the worst cos_sim
model[np.argmin(cos_sim)]

```

## DSP Preprocessing

To attach a DSP preprocessing to one of the network inputs you can provide the NNTool API a config file equal to the one described in section *Attach Special Nodes in front of the Model* and attach it with:

```

config = {
    "window": "hanning",
    "pad_type": "center",
    "n_fft": 512,
    "n_frames": 401,
    "frame_size": 512,
    "window_size": 400,
    "frame_step": 160,
    "sample_rate": 16000,
    "magsquared": True,
    "n_fbanks": 128,
    "fbank_type": "librosa",
    "n_dct": 0,
    "fmin": 0.0,
    "fmax": None,
    "log_type": "natural",
    "log_offset": 1e-6
}
model.insert_DSP_preprocessing(input_node=model[0], dsp_node_type="mfcc", dsp_node_name= ↴
    ↴"dsp_node", config_dict=config)
model.adjust_order()

# Then we quantize to float16 by forcing the statistics for that node
# (since it's going to be float16 it doesn't matter if the numbers are correct)
statistics["dsp_node"] = {
    "range_in": [{"min": -1.0, "max": 1.0}] * len(model["dsp_node"].in_dims),
    "range_out": [{"min": -1.0, "max": 1.0}]
}
model.quantize(
    statistics,
    graph_options={
        'bits': 8,
        'quantized_dimension': 'channel',
        'use_ne16': use_ne16,
        'softmax_out_8bits': True
    },
    node_options={
        'input_1': {
            'scheme': 'FLOAT', 'float_type': 'bfloating16'

```

(continues on next page)

(continued from previous page)

```

    },
    'dsp_node': {
        'scheme': 'FLOAT', 'float_type': 'bfloating16'
    }
)

```

## NNTool API Reference

### [nntool.api](#)

**class nntool.api.NNGraph(model=None, name=None, filename=None, build\_settings=None)**

**add\_constant(dim: Optional[Union[nntool.graph.dim.Dim, Tuple[int]]] = None, name: Optional[str] = None, value: Optional[np.ndarray] = None, is\_mutated=False, is\_intermediate=False, short\_name: Optional[str] = None) → nntool.graph.types.base.NNNodeRef**

Creates a constant node

#### Parameters

- **dim (Union[Dim, Tuple[int]], optional)** – Dimension of constant if not supplied then a value must be. Defaults to None.
- **name (str, optional)** – Optional name. A unique one will be created if None. Defaults to None.
- **value (np.ndarray, optional)** – Numpy array with value. Defaults to None.
- **is\_mutated (bool, optional)** – Constant is both an input and an output. Defaults to False.
- **is\_intermediate (bool, optional)** – Constant is marked as intermediate at import. Defaults to False.
- **short\_name (str, optional)** – Preferred short name for model generation. Defaults to None.

**Returns** A reference to the Node in the Graph

**Return type** NNNodeRef

**add\_dimensions(quiet=False)**

Add dimensions to the graph and calculate execution order and liveness

**Parameters quiet (bool, optional)** – Do not log progress. Defaults to False.

**add\_input(dim: Union[nntool.graph.dim.Dim, Tuple[int]], name: Optional[str] = None, \*\*kwargs) → nntool.graph.types.base.NNNodeRef**

Create an input node. If a name is not supplied then one will be automatically chosen.

#### Parameters

- **dim (Union[Dim, Tuple[int]])** – Input dimension
- **name (str, optional)** – Node name. Defaults to None.

**Returns** Reference to created node in graph

**Return type** NNNodeRef

**add\_output**(*name=None, not\_used=False*) → nntool.graph.types.input\_output.OutputNode  
Create an output node. If a name is not supplied then one will be automatically chosen.

**Parameters** **name** (*str, optional*) – Node name. Defaults to None.

**Returns** Created node

**Return type** OutputParameters

**adjust\_order**(*reshape\_weights=True, no\_postprocess=False, debug\_function: Optional[Callable] = None, steps: Optional[int] = None, single\_step=False*)  
Adjusts tensor order to match selected kernels

**Parameters**

- **reshape\_weights** (*bool, optional*) – Whether weights should be modified to remove transposes. Defaults to True.
- **no\_postprocess** (*bool, optional*) – Whether post processing such as transpose elimination is run. Defaults to False.
- **debug\_function** (*Callable, optional*) – Function to be called after each transpose elimination step. Defaults to None.
- **steps** (*int, optional*) – Number of elimination steps to run. Defaults to None.
- **single\_step** (*bool, optional*) – Execute only one transpose elimination step in each cycle. Defaults to False.

**property all\_constants:**

**Sequence[nntool.graph.types.constant\_input.ConstantInputNode]**

All the constant nodes in the graph

**Returns** List of nodes

**Return type** Sequence[ConstantInputParameters]

**property all\_expressions:**

**Sequence[nntool.graph.types.expression\_fusion.ExpressionFusionNode]**

All the expression nodes in the graph

**Returns** List of nodes

**Return type** Sequence[ExpressionFusionParameters]

**balance\_filters**(*step\_idx: Optional[int] = None, precision\_threshold=0.2*)

Experimental filter balancing routines

**Parameters**

- **step\_idx** (*int, optional*) – Step to balance. Defaults to None.
- **precision\_threshold** (*float, optional*) – Precision threshold. Defaults to 0.20.

**Raises**

- **ValueError** – Bad parameters
- **NotImplementedError** – Bad graph structure

**collect\_statistics**(*input\_tensors\_iterator: Union[Sequence[Sequence[numpy.ndarray]], Sequence[numpy.ndarray]]*) → Mapping[Union[str, Tuple[str, str]], Mapping]

Collect tensor statistics for quantization

**Parameters** **input\_tensors\_iterator** (*Union[Sequence[Sequence[np.ndarray]], Sequence[np.ndarray]]*) – If the graph has a single input this can just be an iterator over

numpy arrays. If the graph has multiple inputs then it should be an iterator over sequences of numpy arrays.

**Returns** Mapping of statistics for each node's inputs and outputs

**Return type** Mapping[Union[str, Tuple[str, str]], Mapping]

**compress\_constant**(node: nntool.graph.types.constant\_input.ConstantInputNode, bits: int, force\_sparse=False, allow\_sparse=True)

Sets a constant node to be compressed

Derives a lookup table for a constant node using bits bits. e.g. G.compress\_constant(G['const1'], 5) will set const1 as compressed with  $2^5$  LUT entries.

#### Parameters

- **node** (*ConstantInputNode*) – Node to compress
- **bits** (*int*) – Number of bits to use for LUT entry
- **force\_sparse** (*bool, optional*) – Force use of sparse bit. Defaults to False.
- **allow\_sparse** (*bool, optional*) – Allow use of sparse bit. Defaults to True.

**Raises** **ValueError** – Input is not a constant

**static cos\_sim**(tensors1: Sequence[Sequence[numpy.ndarray]], tensors2: Sequence[Sequence[numpy.ndarray]], idx: int = 0) → Sequence[float]

**Convenience method to calculate the Cosine similarity between two sets of output tensors from the execute method**

#### Parameters

- **tensors1** (*Sequence[Sequence[np.ndarray]]*) – Output from execute
- **tensors2** (*Sequence[Sequence[np.ndarray]]*) – Output from execute
- **idx** (*int, optional*) – Output index. Defaults to 0.

**Returns** Cosine similarity

**Return type** Sequence[float]

**draw**(all\_dims=False, filepath: Optional[str] = None, view=True, expressions: Optional[Literal[quantized, unquantized]] = None, quant\_labels=False, fusions=False, nodes: Optional[Sequence[nntool.graph.types.base.NNNodeBase]] = None)

Draw the graph in a PDF file

#### Parameters

- **all\_dims** (*bool, optional*) – Show dimensions of constant inputs. Defaults to False.
- **filepath** (*str, optional*) – Filename to save PDF to. If None a temporary file will be used. Defaults to None.
- **view** (*bool, optional*) – Launch the default PDF viewer to view file. Defaults to True.
- **expressions** (*Literal["quantized", "unquantized"], optional*) – Show fused piecewise expressions. If the graph is quantized and “quantized” is selected then the quantized version of the compiled piecewise expression will be shown. Defaults to False.
- **quant\_labels** (*bool, optional*) – Instead of showing tensor dimensions on edges show the quantization information for the edge. Defaults to False.

- **fusions** (*bool, optional*) – Show all fusion internals (except expressions). Defaults to False.
- **nodes** (*Sequence[NNNodeBase], optional*) – List of nodes to draw. If None all nodes except constants to reduce the complexity of the displayed graph. Constants can be included by calling with nodes=graph.nodes().

**execute**(*input\_tensors: Union[np.ndarray, Sequence[np.ndarray]]*, *quantize=False*,  
*dequantize=False*, *output\_fusion\_tensors=False*, *check\_quantization=True*) →  
Sequence[Sequence[np.ndarray]]

Runs inference on the graph

#### Parameters

- **input\_tensors** (*Union[np.ndarray, Sequence[np.ndarray]]*) – Numpy arrays containing inputs (which should be normalized and in float) If there is only one input it can be specified without a sequence.
- **quantize** (*bool, optional*) – Run the graph using quantization parameters. Defaults to False.
- **dequantize** (*bool, optional*) – Dequantize outputs. Implies quantize. Defaults to False.
- **output\_fusion\_tensors** (*bool, optional*) – Output outputs from nodes that have been fused. Defaults to False.
- **check\_quantization** (*bool, optional*) – Run a check that quantization is consistent before executing. Defaults to True.

**Raises** **ValueError** – Incorrect parameters

**Returns** List of lists of outputs of each node in the graph. If output\_fusion\_tensors is True this will also include the output of nodes contained inside fusions (except fused expressions)

**Return type** Sequence[Sequence[np.ndarray]]

**execute\_on\_target**(*input\_tensors: Optional[Sequence[np.ndarray]] = None*, *directory: Optional[str] = None*, *settings: Optional[Mapping[str, Any]] = None*, *script: Optional[Sequence[str]] = None*, *output\_tensors=False*, *performance=True*,  
*pretty=False*, *dont\_run=False*, *at\_log=True*, *jobs: Optional[int] = None*,  
*profile=False*, *at\_loglevel: int = 0*, *print\_output=False*, *pmsis\_os='freertos'*,  
*source='gap9\_v2'*, *platform='gvsoc'*, *do\_clean=True*, *cmake: bool = True*,  
*write\_out\_to\_file=False*, *check\_on\_target=False*, *tolerance=0.0*, *verbose=True*) →  
nntool.graph.nngraph.CompletedProcess

Execute the model using SDK. Builds a project in a directory and executes it.

#### Parameters

- **input\_tensors** (*Sequence[np.ndarray], optional*) – Optional sequence of tensors to pass to function. Defaults to None.
- **directory** (*str, optional*) – Directory to generate project in. Defaults to None.
- **settings** (*Mapping[str, Any], optional*) – Generator settings. Defaults to None. If None is inherited from nntool.graph.settings.
- **script** (*Sequence[str], optional*) – Optional nntool script. If not provided AutoTiler model will be directly generated. Defaults to None.
- **output\_tensors** (*bool, optional*) – Return output tensors in CompletedProcess instance. Defaults to False.

- **performance** (*bool, optional*) – Return layer performance information table as list in CompletedProcess instance. Defaults to True.
- **at\_log** (*bool, optional*) – Return extracted autotiler log in CompletedProcess instance. Defaults to False.
- **pretty** (*bool, optional*) – Return performance table in CompletedProcess instance formatted nicely as string rather than a list. Defaults to False.
- **dont\_run** (*bool, optional*) – Only build the project. Defaults to False.
- **jobs** (*int, optional*) – Number of CPUs to use for build. Defaults to None. If None the amount will be determined using a system call.
- **at\_loglevel** (*int, optional*) – AutoTiler log level. Defaults to 0.
- **pmsis\_os** (*str, optional*) – OS to use for PMSIS. Usually pulpos or freertos. Defaults to ‘freertos’.
- **platform** (*str, optional*) – platform to run on gvsoc, board, emul, etc. Defaults to ‘gvsoc’.
- **source** (*str, optional*) – Target to source i.e. one of the scripts from GAP\_SDK/configs. For the GAP9 evk use gap9\_evk\_audio. Defaults to ‘gap9\_v2’.
- **do\_clean** (*bool, optional*) – Do not clean the project when run a second time. Default: False
- **cmake** (*bool, optional*) – Use cmake GAP project structure (this is now the default and False remains only for legacy builds)
- **write\_out\_to\_file** (*bool, optional*) – If True writes the outputs of the network (not all of the layers) to files.
- **check\_on\_target** (*bool, optional*) – Produce main.c with for loop that compares results with ground truth, i.e. checks directly on chip
- **tolerance** (*float, optional*) – If check\_on\_target, this is the tolerance used to compare the results
- **verbose** (*bool, optional*) – If check\_on\_target, this enables print of all the errors present in the execution: index, actual and expected values (Default: True)

**Raises** `ValueError` – SDK not sourced

**Returns** Information on process that ran

**Return type** CompletedProcess

#### **property filename: str**

Filename that graph was loaded from (if any)

**Returns** Filename

**Return type** str

#### **fusions(\*match\_names, no\_postprocess: bool = False, exclude\_match\_names: Optional[Sequence[str]] = None)**

Run matchers on the graph

The scale8 match group has the name scaled\_match\_group The pow2 match group has the name pow2\_match\_group get\_fusions() can be used to retrieve a dictionary of available fusions

#### **Parameters**

- **match\_names** (*str*) – Names of matches to apply

- **no\_postprocess** (*bool, optional*) – Do not execute postprocessing such as transpose elimination. Defaults to False.

- **exclude\_match\_names** (*Sequence[str], optional*) – Do no run these match names if in a match group or added by other matches

**gen\_project** (*input\_tensors: Optional[Sequence[numpy.ndarray]] = None, directory: Optional[str] = None, settings: Optional[Mapping[str, Any]] = None, script: Optional[Sequence[str]] = None, at\_loglevel: int = 0, platform='gvsoc', cmake: bool = False, output\_tensors=False*)

Builds a GAP template project in a directory.

#### Parameters

- **input\_tensors** (*Sequence[np.ndarray], optional*) – Optional sequence of tensors to pass to function. Defaults to None.
- **directory** (*str, optional*) – Directory to generate project in. Defaults to None.
- **settings** (*Mapping[str, Any], optional*) – Generator settings. Defaults to None. If none is inherited from nntool.graph.settings.
- **script** (*Sequence[str], optional*) – Optional nntool script. If not provided AutoTiler model will be directly generated. Defaults to None.
- **at\_loglevel** (*int, optional*) – AutoTiler log level. Defaults to 0.
- **cmake** (*bool, optional*) – Use cmake GAP project structure (this is now the default and False remains only for legacy builds)
- **output\_tensors** (*bool, optional*) – Return output tensors in CompletedProcess instance. Defaults to False.

**Raises** **ValueError** – SDK not sourced

**static get\_fusions()**

Returns a dictionary of all the fusion/graph optimization pass names and descriptions

**Returns** Names and descriptions of graph optimisation passes

**Return type** Dict[str, str]

**get\_node\_option** (*name: str, option\_name: str*) → Any

Shortcut to get an option on a node

#### Parameters

- **name** (*str*) – Name of the node to get
- **option\_name** (*str*) – Node option name

#### Raises

- **KeyError** – Node not found
- **ValueError** – Option not valid for node

**Returns** Value of option

**Return type** Any

**property has\_dsp: bool**

Graph has DSP nodes

**Returns** True if present

**Return type** bool

**property has\_expressions: bool**

Graph has compiled expressions

**Returns** True if present

**Return type** bool

**has\_node\_type(node\_type: nntool.graph.types.base.NNNNodeBase) → bool**

Returns True if graph contains node type

**Parameters** **node\_type** (*Parameters*) – Node class

**Returns** True if present

**Return type** bool

**property has\_quantized\_parameters: bool**

Graph was imported with quantized parameters

**Returns** quantized parameters or not

**Return type** bool

**property has\_resizer: bool**

Graph has resizer nodes

**Returns** True if present

**Return type** bool

**has\_rnn(ktype: Optional[str] = None, ne16: bool = False) → bool**

Graph has RNN nodes

**Parameters**

- **ktype** (*str, optional*) – kernel type to match or all. Defaults to None.
- **ne16** (*bool, optional*) – match nodes that will map to ne16 kernels. Defaults to False.

**Returns** True if present

**Return type** bool

**property has\_ssd\_postprocess: bool**

Graph has SSD detector nodes

**Returns** True if present

**Return type** bool

**static hist\_compare\_tensors(tensors1: Sequence[Sequence[numpy.ndarray]], tensors2:**

*Sequence[Sequence[numpy.ndarray]]]) → None*

Convenience method to plot the histogram of the two sets of tensors overlapped

**Parameters**

- **tensors1** (*Sequence[Sequence[np.ndarray]]*) – Output from execute
- **tensors2** (*Sequence[Sequence[np.ndarray]]*) – Output from execute

**input\_nodes() → Generator[nntool.utils.graph.Node, None, None]**

Iterate over all inputs

**Returns** a generator for all nodes

**Return type** Generator[Node]

**inputs\_and\_constants()** → Generator[nntool.utils.graph.Node, None, None]  
 Iterate over all inputs and constants

**Returns** a generator for all nodes

**Return type** Generator[Node]

**insert\_dsp\_preprocessing**(*input\_node*: nntool.utils.graph.Node, *dsp\_node\_type*: str, *dsp\_node\_name*: str = 'dsp\_node', *config\_dict*: Optional[Mapping[str, Any]] = None)

Inserts a DSP preprocessing node before one of the *input\_node* provided

**Parameters**

- **input\_node** (NNNNodeBase) – Graph Node before which nntool will insert the DSP pre-processing
- **dsp\_node\_type** (str) – Type of DSP operation (“mfcc” or “fft”)
- **dsp\_node\_name** (str) – Name of the node insrted (default: “dsp\_node”)
- **config\_dict** (Mapping[str, Any]) – Dictionary with configuration for LUT generation, i.e.:

```
config = {
    "window": "hanning",           # Window numpy function
    "pad_type": "center",          # How to apply padding if frame_
    ↪size < n_fft
    "n_fft": 512,                 # Number of FFT bins
    "n_frames": 401,               # Number of frames (it must match_
    ↪the graph input size)
    "frame_size": 512,             # Number of samples in each frame
    "window_size": 400,             # Number of samples for the window_
    ↪function (if < frame_size the window will be center padded)
    "frame_step": 160,              # Number of samples between two_
    ↪consecutive frames
    "sample_rate": 16000,            # Samplerate
    "magsquared": True,             # If True, the spectrogram is_
    ↪calculated as np.abs(rfft) ** 2, else np.abs(rfft)
    "n_fbanks": 128,                # Number of Mel Filterbanks_
    ↪(ignored when dsp_node_type=="fft")
    "fbank_type": "librosa",        # Algorithm used to generate the_
    ↪Mel Filters
    "mel_filterbanks_file": None,   # Path to melfilters .npy array_
    ↪provided by user, in this case fbank_type will be ignored
    "n_dct": 0,                     # Number of DCT bins in case of_
    ↪Mfcc preprocessing
    "dct_type": 2,                  # DCT algorithm type
    "fmin": 0.0,                    # Fmin used to calculate the Mel_
    ↪Filters
    "fmax": None,                   # Fmax used to calculate the Mel_
    ↪Filters
    "log_type": "natural",          # Logarithm type ("None", "db" or_
    ↪"natural"), None will output melspectrogram
    "log_offset": 1e-6               # Offset applied before log function
}
```

```
static load_graph(filepath_or_model: Union[str, Any], load_quantization=False,
                    remove_quantize_ops=True, use_hard_sigmoid=False, use_hard_tanh=False,
                    fold_batchnorm=True, rescale_perchannel=True, **opts) →
                    nntool.graph.nngraph.NNGraph
```

Load a graph into NNTool returning an NNGraph instance.

#### Parameters

- **filepath\_or\_model** (Union[str, Any]) – Absolute path to file or ONNX ModelProto
- **load\_quantization** (bool, optional) – Load quantization information in graph. In TFLite graphs this loads the tensor quantization information. In ONNX graphs exported from the NNCF compression and quantization framework this enables interpretation of the NNCF LinearQuantize operators. Defaults to False.
- **remove\_quantize\_ops** (bool, optional) – Remove quantize operators from graph. Defaults to True.
- **use\_hard\_sigmoid** (bool, optional) – Force use of hard sigmoids for sigmoid operations. Defaults to False.
- **use\_hard\_tanh** (bool, optional) – Force use of hard tanh for tanh operations. Defaults to False.
- **fold\_batchnorm** (bool, optional) – Fold batchnorm during import. Defaults to True.
- **rescale\_perchannel** (bool, optional) – Rescale all filter tensors to per channel quantized weights when importing TFLITE quantized graphs. Defaults to True.

**Returns** Loaded graph

**Return type** *NNGraph*

#### property model

The original model that generated the NNTool graph

**Returns** The model file (TFLite or ONNX graph descriptor)

**Return type** Any

#### property name: str

Returns the name of the graph potentially modified to be a valid C identifier

**Returns** The graph name

**Return type** str

#### needs\_adjust()

Checks if all nodes' tensor order matches an available kernel

**Returns** Graph requires adjust order to be run

**Return type** bool

#### nodes(node\_classes: Optional[nntool.utils.graph.Node] = None, sort=False) →

Sequence[nntool.utils.graph.Node]

Select nodes in the graph filtering by node class.

#### Parameters

- **node\_classes** (Node, optional) – Node classes to filter. Defaults to None.
- **sort** (bool, optional) – Sort nodes by step idx. Defaults to False.

**Returns** List of nodes.

**Return type** Sequence[Node]

**property nodes\_by\_step\_idx:** Sequence[nntool.graph.types.base.NNNodeBase]  
All the nodes in the graph ordered by execution order

**Returns** List of nodes

**Return type** Sequence[Parameters]

**property nodes\_by\_step\_idx\_with\_fusions:**  
Sequence[nntool.graph.types.base.NNNodeBase]  
Nodes ordered by execution order but also including internal nodes for fusions

**Returns** List of nodes

**Return type** Sequence[Parameters]

**nodes\_iterator(yield\_fusions=True)**  
Yields a tuple of length 4 with the step idx and parameters of each node. Optionally when in a fusion yields tuples containing the fusion internal step id and node for each internal node.

**Parameters** **yield\_fusions** (bool, optional) – Whether to yield fusion nodes. Defaults to True.

**Yields** [Tuple[int, Parameters, Optional[int], Optional[Parameters]]] –  
**Tuple containing node\_idx**, node, fusion\_idx, fusion\_node

**property num\_constants:** int  
Current number of constant inputs

**Returns** Number of constant inputs

**Return type** int

**property num\_inputs:** int  
Current number of inputs

**Returns** Number of inputs

**Return type** int

**property num\_outputs:** int  
Current number of outputs

**Returns** Number of outputs

**Return type** int

**output\_nodes()** → Generator[nntool.utils.graph.Node, None, None]  
Iterate over all outputs

**Returns** a generator for all nodes

**Return type** Generator[Node]

**plot\_mem\_usage(nodes=None, normalized=False, tot\_mem\_usage=False, l2\_size=None)**  
Show the quantization of the graph

**Parameters**

- **nodes** (list, optional) – List of graph nodes of which you want a quantization report, i.e. G.qshow([G[10]])
- **width** (int, optional) – Width in characters of the report. Defaults to 150 characters.

**Returns** A table of the node quantification

**Return type** str

**qshow**(*nodes=None*, *width=150*) → str  
Show the quantization of the graph

**Parameters**

- **nodes** (*list, optional*) – List of graph nodes of which you want a quantization report, i.e. G.qshow([G[10]])
- **width** (*int, optional*) – Width in characters of the report. Defaults to 150 characters.

**Returns** A table of the node quantification

**Return type** str

**static qsnrs**(*tensors1: Sequence[Sequence[numpy.ndarray]]*, *tensors2: Sequence[Sequence[numpy.ndarray]]*, *idx: int = 0*) → Sequence[float]  
Convenience method to calculate the QSNR between two sets of output tensors from the execute method

**Parameters**

- **tensors1** (*Sequence[Sequence[np.ndarray]]*) – Output from execute
- **tensors2** (*Sequence[Sequence[np.ndarray]]*) – Output from execute
- **idx** (*int, optional*) – Output index. Defaults to 0.

**Returns** QSNRs

**Return type** Sequence[float]

**property quantization:**

**Optional[nntool.quantization.quantization\_set.QuantizationSet]**

Current graph Quantization

**Returns** quantization set

**Return type** Union[QuantizationSet, None]

**quantize**(*statistics: Optional[Mapping[Union[str, Tuple[str, str]], Mapping]] = None*, *schemes: Optional[Sequence[str]] = None*, *graph\_options: Optional[Mapping[str, Any]] = None*, *node\_options: Optional[Mapping[Union[str, Tuple[str, str]], Mapping[str, Any]]] = None*, *read\_existing\_options=True*, *no\_postprocess=False*) → None  
Quantize the graph

**Parameters**

- **statistics** (*Mapping[Union[str, Tuple[str, str]], Mapping], optional*) – Statistics collected by the NNGraph.collect\_statistics method.
- **schemes** (*Sequence[], optional*) – Sequence of schemes “scaled”, “pow2”, or “float” to use in priority order. If None use scaled. Defaults to None.
- **graph\_options** (*Mapping[str, Any], optional*) – Quantization options to set for the whole graph. Defaults to None.
- **node\_options** (*Mapping[Union[str, Tuple[str, str]], Mapping[str, Any]], optional*) – Quantization options to set for specific nodes. The map key should be the node name or if the node is inside a fusion then a tuple of the fusion name and the node name. Defaults to None.
- **read\_existing\_options** (*bool, optional*) – Incorporate existing quantization options and schemes in the graph (if quantization is present). Leaving this as True and just

supplying graph\_option, node\_options and/or schemes is the equivalent of the nntool qtune command

- **no\_postprocess** (*bool, optional*) – If True skip adjust after quantize

**static read\_graph\_state**(*graphpath: str*) → *nntool.graph.nngraph.NNGraph*

Read a graph zip module created by write\_graph\_state.

**Parameters** **graphpath** (*str*) – path to archive file

**Returns** Created graph

**Return type** *NNGraph*

**remove\_nodes**(*node\_from, node\_to=None, up=True, leave=True, no\_check=False*)

**Removes all the edges and nodes between node\_from and node\_to. Will only work if nodes do not affect shape of tensors.**

If node\_to is None, this will remove all nodes following/preceding (up argument) node\_from and the node specified (leave argument). New output nodes will be created on all the inputs to the node specified. Specifying up does the remove upwards and creates inputs.

**Parameters**

- **node\_from** (*Node, optional*) – Node that remove operation starts from.
- **node\_to** (*str, optional*) – Node that remove operation ends to. Default to None: i.e. remove everything above/under node\_from.
- **up** (*bool, optional*) – When node\_to=None, remove it and everything above it. Defaults to True.
- **leave** (*bool, optional*) – When node\_to=None, only remove what is above or below and not the node itself. Defaults to True.
- **no\_check** (*bool, optional*) – Don't check that the graph structure is damaged. Default to False

**set\_node\_option**(*name: str, option\_name: str, value: Optional[Any] = None*)

Shortcut to set nodeoptions. Equivalent to the node option command.

**Parameters**

- **name** (*str*) – Name of the node to set
- **option\_name** (*str*) – Name of the node option
- **value** (*Any, optional*) – The value of the option. If None the option will be cleared. Defaults to None.

**Raises**

- **KeyError** – Node name is not found
- **ValueError** – Option is not valid for the node
- **ValueError** – Value is not of correct type

**property settings: dict**

NNTool shell settings relating to autotiler model creation

**Returns** Map of settings

**Return type** dict

**show**(*nodes=None*, *show\_constants=False*, *width=150*) → str

Show the structure of the graph

#### Parameters

- **show\_constants** (*bool*, *optional*) – Include constants in report. Defaults to False.
- **nodes** (*list*, *optional*) – List of graph nodes of which you want a report, i.e. G.show([G[10]])
- **width** (*int*, *optional*) – Width in characters of the report. Defaults to 150 characters.

**Returns** A table of the nodes in the graph

**Return type** str

**property total\_memory\_usage:** int

Estimated total number of parameters in the graph

**Returns** Number of operations

**Return type** int

**property total\_ops:** int

Estimated total operations in the graph

**Returns** Number of operations

**Return type** int

**use\_compressed**(*on\_off=True*)

Switch on and off compression on constant nodes that have compressed values

**write\_graph\_state**(*graphpath: Optional[str] = None*, *build\_settings=None*) → Tuple[Any]

Save a graph along with all changes made and quantization.

The zip file created by this method can be loaded with NNGraph.read\_graph\_state(*path\_to\_file*).

The file is actually a zip module with a single module in it with the name of the basename of the supplied path without a .zip extension if any. The module contains a single element with the same name containing the imported graph.

It can be imported like:

```
import sys
sys.path.insert(0, "/path_to_zip/mygraph.zip")
from mygraph import mygraph
```

mygraph will now contain the imported NNGraph

#### Parameters

- **graphpath** (*str*, *optional*) – If provided a zip module of the graph along with its tensors is saved. Defaults to None.
- **build\_settings** (*dict*, *optional*) – Override build settings in graph.settings.

**Returns** The python function used to create the graph and dictionary of the tensors.

**Return type** Tuple[Any]

**nntool.api.utils**

```
class nntool.api.utils.FileImporter(files: Union[str, Sequence[str]], transpose=False, norm_func:  
                                      Optional[Union[str, callable]] = None, rgb888_rgb565=False, width:  
                                      Optional[int] = None, height: Optional[int] = None, mode:  
                                      Optional[str] = None, npdtype: Optional[str] = None)
```

```
class nntool.api.utils.RandomIter(count: int, shapes: Sequence[Sequence[int]], ranges:  
                                    Sequence[Sequence[float]], gen:  
                                    Optional[numumpy.random._generator.Generator] = None)
```

Value generator using a random number source

**classmethod fake(G: NNGraph, gen: numpy.random.\_generator.Generator = None) → RandomIter**  
Create instance that matches graph G inputs

**Parameters**

- **G (NNGraph)** – Graph to create input values for
- **gen (np.random.Generator, optional)** – Numpy random number generator to use.  
Defaults to None.

**Returns** A random input generator that produces inputs for G

**Return type** *RandomIter*

```
nntool.api.utils.cos_sims(tensors1: Sequence[Sequence[numumpy.ndarray]], tensors2:  
                           Sequence[Sequence[numumpy.ndarray]], idx: int = 0) → Sequence[float]
```

**Convenience method to calculate the Cosine similarity between two sets of output tensors from the execute method**

**Parameters**

- **tensors1 (Sequence[Sequence[np.ndarray]])** – Output from execute
- **tensors2 (Sequence[Sequence[np.ndarray]])** – Output from execute
- **idx (int, optional)** – Output index. Defaults to 0.

**Returns** Cosine similarity

**Return type** Sequence[float]

```
nntool.api.utils.import_data(filepath: str, transpose=False, norm_func: Optional[Union[str, callable]] =  
                               None, rgb888_rgb565=False, width: Optional[int] = None, height:  
                               Optional[int] = None, slices: Optional[Sequence[Sequence[int]]] = None,  
                               zero_pad: Optional[Union[Sequence[int], int]] = None, mode: Optional[str] =  
                               None, npdtype: Optional[str] = None) → numpy.ndarray
```

Imports data from image, sound or data files such as numpy npy files.

File type is detected by file extension. Valid extensions are:

- Images: .pgm, .png, .ppm, .jpg, jpeg
- Sounds: .wav, .raw, .pcm
- Data: .npy, .dat

**Parameters**

- **filepath (str)** – Full path to file

- **transpose** (*bool, optional*) – Transpose images from HWC to CHW. Defaults to False.
- **norm\_func** (*Union[str, callable], optional*) – Normalization function to apply. This can be a function or a string as in the NNTool command interfreter. Defaults to None.
- **rgb888\_rgb565** (*bool, optional*) – Reformat RGB565 images to RGB888. Defaults to False.
- **width** (*int, optional*) – Width for image. Defaults to None.
- **height** (*int, optional*) – Height for image. Defaults to None.
- **slices** (*Sequence[Sequence[int]]*) – Slices to take of input. As ((start, stop, step), ...). Defaults to None.
- **zero\_pad** – (*Sequence[int], int*): Amount of zero padding to add. As numpy.pad. Defaults to None.
- **mode** (*str, optional*) – Modes for image import. Can be one of:

```
'1':           1-bit pixels, black and white, stored with one pixel per ↵byte
'L':           8-bit pixels, black and white
'P':           8-bit pixels, mapped to any other mode using a color ↵palette
'RGB':         3x8-bit pixels, true color
'RGBA':        4x8-bit pixels, true color with transparency mask
'CMYK':        4x8-bit pixels, color separation
'YCbCr':      3x8-bit pixels, color video format
'LAB':         3x8-bit pixels, the L*a*b color space
'HSV':         3x8-bit pixels, Hue, Saturation, Value color space
'I':           32-bit signed integer pixels
'F':           32-bit floating point pixels
```

Defaults to None.

- **nptype** (*str, optional*) – Set imput tensor as named numpy dtype for example “int16”. Defaults to None.

**Returns** Returns imported tensor

**Return type** np.ndarray

```
nntool.api.utils.model_settings(default_input_home_location: str = 'AT_MEM_L2',
                                 default_input_exec_location: str = 'AT_MEM_L2',
                                 default_output_home_location: str = 'AT_MEM_L2',
                                 default_output_exec_location: str = 'AT_MEM_L2',
                                 default_global_home_location: str = 'AT_MEM_L3_HFLASH',
                                 default_global_exec_location: str = 'AT_MEM_UNDEF',
                                 default_local_location: str = 'AT_MEM_UNDEF', l2_ram_ext_managed:
                                 bool = True, l3_ram_ext_managed: bool = False, l3_flash_ext_managed:
                                 bool = False, include_project_header: bool = False, tensor_directory: str
                                 = '.', model_directory: str = '.', model_file: str = 'model.c',
                                 basic_kernel_source_file: str = 'Expression_Kernels.c',
                                 basic_kernel_header_file: str = 'Expression_Kernels.h', at_ver: int = 3,
                                 l3_ram_device: str = 'AT_MEM_L3_HRAM', l3_flash_device: str =
                                 'AT_MEM_L3_HFLASH', AT_force_relu: bool = True, l1_size: int =
                                 64000, l2_size: int = 300000, l3_size: int = 8000000, l3_flash_mb: int =
                                 64, fc_freq: int = 250000000, cl_freq: int = 175000000,
                                 cluster_stack_size: int = 4096, cluster_slave_stack_size: int = 1024,
                                 cluster_num_cores: int = 8, anonymise: bool = False,
                                 graph_monitor_cycles: bool = False, graph_monitor_cvar_name: str =
                                 'AT_GraphPerf', graph_produce_node_names: bool = False,
                                 graph_produce_node_cvar_name: str = 'AT_GraphNodeNames',
                                 graph_produce_operinfos: bool = False,
                                 graph_produce_operinfos_cvar_name: str = 'AT_GraphOperInfosNames',
                                 graph_reorder_constant_in: bool = True, graph_trace_exec: bool = False,
                                 graph_noinline_node: bool = False, graph_pref_l3_exec: str =
                                 'AT_MEM_L3_HRAM', graph_const_exec_from_flash: bool = False,
                                 graph_pref_l3_home: str = 'AT_MEM_L3_HFLASH',
                                 graph_dump_tensor: int = 0, graph_checksum: int = 0,
                                 graph_dump_one_node: Optional[str] = None, graph_arg2struct: bool =
                                 False, graph_size_opt: int = 0, graph_warm_construct: int = 0,
                                 graph_group_weights: bool = False, graph_async_fork: bool = False,
                                 graph_dump_graph_outputs: bool = False, graph_runner_re_entra nt:
                                 bool = False)
```

Create execute on target model settings

Utility method to create execute on target settings settings dictionary.

#### Parameters

- **default\_input\_home\_location (str)** – default home location for inputs for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMEFLASH’,  
‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFAULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_L2’.

- **default\_input\_exec\_location (str)** – default exec location for inputs for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMEFLASH’,  
‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFAULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_L2’.

- **default\_output\_home\_location (str)** – default home location for outputs for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
 ‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
 ‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
 ‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_L2’.

- **default\_output\_exec\_location (str)** – default exec location for outputs for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
 ‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
 ‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
 ‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_L2’.

- **default\_global\_home\_location (str)** – default home location for globals for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
 ‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
 ‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
 ‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_L3\_HFLASH’.

- **default\_global\_exec\_location (str)** – default exec location for globals for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
 ‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
 ‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
 ‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_UNDEF’.

- **default\_local\_location (str)** – default location for locals for code generation.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
 ‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
 ‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
 ‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.

Default is ‘AT\_MEM\_UNDEF’.

- **l2\_ram\_ext\_managed (bool)** – Externally manage L2 RAM.

Choices are: True, False.

Default is True.

- **l3\_ram\_ext\_managed (bool)** – Externally manage L3 RAM.

Choices are: True, False.

Default is False.

- **l3\_flash\_ext\_managed (bool)** – Externally manage L3 flash.

Choices are: True, False.

Default is False.

- **include\_project\_header** (*bool*) – Include a header file called “*GraphName.h*” in generated code.

Choices are: True, False.

Default is False.

- **tensor\_directory** (*str*) – directory to dump tensors to.

Default is ‘.’.

- **model\_directory** (*str*) – directory to dump model to.

Default is ‘.’.

- **model\_file** (*str*) – filename for model.

Default is ‘model.c’.

- **basic\_kernel\_source\_file** (*str*) – filename for generated basic kernels.

Default is ‘Expression\_Kernels.c’.

- **basic\_kernel\_header\_file** (*str*) – filename for generated basic kernel headers.

Default is ‘Expression\_Kernels.h’.

- **at\_ver** (*int*) – AutoTiler version.

Default is 3.

- **l3\_ram\_device** (*str*) – L3 RAM device.

Choices are: ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFAULTRAM’.

Default is ‘AT\_MEM\_L3\_HRAM’.

- **l3\_flash\_device** (*str*) – L3 FLASH device.

Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
‘AT\_MEM\_L3\_DEFAULTFLASH’.

Default is ‘AT\_MEM\_L3\_HFLASH’.

- **AT\_force\_relu** (*bool*) – Replace reluN with relu in the AT model.

Default is True.

- **l1\_size** (*int*) – Amount of L1 memory to use in target (including cluster stack).

Default is 64000.

- **l2\_size** (*int*) – Amount of L2 memory to use in target.

Default is 300000.

- **l3\_size** (*int*) – Amount of L3 memory to use in target.

Default is 8000000.

- **l3\_flash\_mb** (*int*) – Amount of FLASH L3 memory MB to use in target.

Default is 64.

- **fc\_freq (int)** – Fabric Controller frequency in Hz.  
Default is 250000000.
- **cl\_freq (int)** – Cluster frequency in Hz.  
Default is 175000000.
- **cluster\_stack\_size (int)** – Size of stack for cluster master core.  
Default is 4096.
- **cluster\_slave\_stack\_size (int)** – Size of stack for slave core.  
Default is 1024.
- **cluster\_num\_cores (int)** – Number of cores in cluster.  
Default is 8.
- **anonymise (bool)** – Try to anonymise names.  
Default is False.
- **graph\_monitor\_cycles (bool)** – Enable automatic cycle capture for each node of the graph.  
Choices are: True, False.  
Default is False.
- **graph\_monitor\_cvar\_name (str)** – When GRAPH\_MONITOR\_CYCLES is enabled, set the name of the C var array to receive results.  
Default is ‘AT\_GraphPerf’.
- **graph\_produce\_node\_names (bool)** – Enable production of an array containing the name of each graph node.  
Choices are: True, False.  
Default is False.
- **graph\_produce\_node\_cvar\_name (str)** – When GRAPH\_PRODUCE\_NODE\_NAMES is enabled, set the name of the C array receiving the names as strings.  
Default is ‘AT\_GraphNodeNames’.
- **graph\_produce\_operinfos (bool)** – Enable production of number of operations/mac for each layer.  
Choices are: True, False.  
Default is False.
- **graph\_produce\_operinfos\_cvar\_name (str)** – When GRAPH\_PRODUCE\_OPERINFOS is enabled, set the name of the C array receiving mac infos for each node.  
Default is ‘AT\_GraphOperInfosNames’.
- **graph\_reorder\_constant\_in (bool)** – Enable reordering of constant inputs in order to transform 2D accesses into 1D accesses.  
Choices are: True, False.  
Default is True.

- **graph\_trace\_exec** (*bool*) – Enable trace of activity.  
Choices are: True, False.  
Default is False.
- **graph\_noinline\_node** (*bool*) – If on, all user kernel function are marked as noinline.  
Choices are: True, False.  
Default is False.
- **graph\_pref\_l3\_exec** (*str*) – In case a symbol must be allocated in L3 for execution this is the prefered memory.  
Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.  
Default is ‘AT\_MEM\_L3\_HRAM’.
- **graph\_const\_exec\_from\_flash** (*bool*) – If on, constant symbols executes from home location.  
Choices are: True, False.  
Default is False.
- **graph\_pref\_l3\_home** (*str*) – L3 flash type preferrer for constant symbols.  
Choices are: ‘AT\_MEM\_L3\_HFLASH’, ‘AT\_MEM\_L3\_QSPIFLASH’,  
‘AT\_MEM\_L3\_OSPIFLASH’, ‘AT\_MEM\_L3\_MRAMFLASH’,  
‘AT\_MEM\_L3\_DEFAULTFLASH’, ‘AT\_MEM\_L3\_HRAM’, ‘AT\_MEM\_L3\_QSPIRAM’,  
‘AT\_MEM\_L3\_OSPIRAM’, ‘AT\_MEM\_L3\_DEFULTRAM’, ‘AT\_MEM\_L2’.  
Default is ‘AT\_MEM\_L3\_HFLASH’.
- **graph\_dump\_tensor** (*int*) – Trace selected tensors arguments at inference time, either all nodes or selected node.  
Choices are: 0, 1.  
Default is 0.
- **graph\_checksum** (*int*) – Generate the checksum functions call in the C code and print the checksum values.  
Choices are: 0, 1.  
Default is 0.
- **graph\_dump\_one\_node** (*str*) – Trace one specific graph node.  
Default is None.
- **graph\_arg2struct** (*bool*) – Kernel C arguments are promoted to struct.  
Choices are: True, False.  
Default is False.
- **graph\_size\_opt** (*int*) – 0: Default make opt, 1: O2 for all layers, Os for xtor,dxtor,runner,  
2: Os for layers and xtor,dxtor,runner.  
Choices are: 0, 1, 2.  
Default is 0.

- **graph\_warm\_construct (int)** – Generate construct/destruct functions with the Warm option to only 1. allocate/deallocate L1 buffer, 2. separate alloc/dealloc of L2 dynamic memory, 3. fully flexible construct/destruct with separate allocation functionalities.

Choices are: 0, 1, 2, 3.

Default is 0.

- **graph\_group\_weights (bool)** – Group together KerArg of the same type/size in the context of the single layer. For example weights and biases in RNN.

Choices are: True, False.

Default is False.

- **graph\_async\_fork (bool)** – ONLY AVAILABLE IN GAP9: Generate user kernel code with async fork option.

Choices are: True, False.

Default is False.

- **graph\_dump\_graph\_outputs (bool)** – Trace graph output tensors at inference time.

Choices are: True, False.

Default is False.

- **graph\_runner\_re\_entrant (bool)** – Generate re-entrant graph runner.

Choices are: True, False.

Default is False.

`nntool.api.utils.qsnr(orig, quant, axis=None)`

Calculate the QSNR between two tensors

`nntool.api.utils.qsnrs(tensors1: Sequence[Sequence[numpy.ndarray]], tensors2:`

`Sequence[Sequence[numpy.ndarray]], idx: int = 0) → Sequence[float]`

Convenience method to calculate the QSNR between two sets of output tensors from the execute method

#### Parameters

- **tensors1 (Sequence[Sequence[np.ndarray]])** – Output from execute
- **tensors2 (Sequence[Sequence[np.ndarray]])** – Output from execute
- **idx (int, optional)** – Output index. Defaults to 0.

#### Returns QSNRs

**Return type** Sequence[float]

`nntool.api.utils.quantization_options(clip_type: str = 'none', float_type: str = 'float32', kernel_type: str = 'fastfloat', hwc: bool = False, sq_bits: int = 8, force_output_size: int = 8, weight_bits: int = 8, force_external_size: int = 8, narrow_weights: bool = True, use_ne16: bool = False, narrow_state: bool = True, max_precision_limit: int = 2, force_rnn_1_minus_1_out: bool = False, quantized_dimension: str = 'channel', force_ne16: bool = False, allow_asymmetric_out: bool = True, force_input_size: int = 8, softmax_out_8bits: bool = False, bits: int = 16, pow2_biases: int = 0)`

Create quantization options

Utility method to create quantization options dictionary.

### Parameters

- **clip\_type (str)** – Clipping method for any node that modifies its input: none - the minimum and maximum observed values are used. laplace, gaus - Values chosen based on laplace or gaussian distribution mix - MSE is used to estimate if distribution is laplace or gaussian std3, std5 - 3 or 5 times standard deviation from mean  
Default is ‘none’.
- **float\_type (str)** – float type to use for quantization  
Default is ‘float32’.
- **kernel\_type (str)** – Kernel to use for activation function  
Default is ‘fastfloat’.
- **hwc (bool)** – Use HWC kernel  
Default is False.
- **sq\_bits (int)** – bits for inputs and outputs of scaled kernels  
Default is 8.
- **force\_output\_size (int)** – number of bits to use for output features  
Default is 8.
- **weight\_bits (int)** – how many bits to use in weights  
Default is 8.
- **force\_external\_size (int)** – bits to use for features and state  
Default is 8.
- **narrow\_weights (bool)** – scales filter weights with a representation of both 1 and -1 (i.e. -127 - 127 in 8 bits)  
Default is True.
- **use\_ne16 (bool)** – enable use of NE16 kernels (if supported) on this layer  
Default is False.
- **narrow\_state (bool)** – scales state in Q14 so that 1 <-> -1 can be represented  
Default is True.
- **max\_precision\_limit (int)** – maximum number of bits to degrade input scale precision by to stop overflow of accumulator.  
Default is 2.
- **force\_rnn\_1\_minus\_1\_out (bool)** – forces RNNs to have narrow, symmetric 1 - - 1 output  
Default is False.
- **quantized\_dimension (str)** – scales filter weights by channel or tensor  
Default is ‘channel’.
- **force\_ne16 (bool)** – force use of NE16 kernels on this layer - may not be supported for model generation  
Default is False.

- **allow\_asymmetric\_out** (*bool*) – Allow soft kernels to use asymmetric quantization where possible. This option defaults to False on inputs and outputs except for inputs if their minimum value in the statistics is 0 in which case it is always True.  
Default is True.
- **force\_input\_size** (*int*) – number of bits to use for input features  
Default is 8.
- **softmax\_out\_8bits** (*bool*) – make the output scale8 8 bits  
Default is False.
- **bits** (*int*) – bits for inputs and outputs  
Default is 16.
- **pow2\_biases** (*int*) – bits for filter biases - if set to 0 the same size as the output type will be used  
Default is 0.

## nntool.api.compression

```
class nntool.api.compression.AutoCompress(graph: NNGraph, labels_and_inputs: Sequence[Tuple[Any,
Sequence[numpy.ndarray]]], validation:
nntool.utils.validation_utils.ValidateBase, start_qsnr=30,
min_step=0.5, base_inputs=None)

nntool.api.compression.print_progress(msg, newline)
```

## nntool.api.quantization

```
class nntool.api.quantization.MultMulBiasScaleQType(*args, scale=None, dtype=<class
'numpy.uint8'>, available_bits=None,
calc_dtype=<class 'numpy.int32'>,
float_scale=False, **kwargs)

class nntool.api.quantization.QRec(in_qs=None, out_qs=None, ktype=None, auto_quantize_inputs=False,
auto_dequantize_inputs=False, auto_quantize_outputs=False,
auto_dequantize_outputs=False, **kwargs)

nntool.api.quantization.QSet
alias of nntool.quantization.quantization_set.QuantizationSet

class nntool.api.quantization.QType(*args, q=None, bits=None, signed=None, zero_point=0, scale=None,
min_val=None, max_val=None, quantized_dimension=None,
dtype=None, offset=None, narrow_range=None, forced=False,
asymmetric=None, dont_copy_attr=None, **kwargs)

static precision_key()
Returns a key function that compares precision
```

## nntool.api.types

### Base classes

```
class nntool.api.types.NNNodeBase(name, in_dims_hint=None, out_dims_hint=None, in_dims=None,  
                                   out_dims=None, step_idx=-1, ker_in_order=None,  
                                   ker_out_order=None, meta=None, at_options=None, **kwargs)
```

**property at\_options: nntool.graph.types.base.NodeOptions**

AutoTiler node options

**Returns** AutoTiler node options

**Return type** NodeOptions

### Filter classes

```
class nntool.api.types.Conv2DNode(name, dilation=None, groups=None, multiplier=1, in_dims_hint=None,  
                                   out_dims_hint=None, batch=None, ker_in_order=None,  
                                   ker_out_order=None, custom=None, tf_depthwise=False,  
                                   has_bias=True, **kwargs)
```

```
class nntool.api.types.LinearNode(*args, filter_dim=None, batch_size=1, keep_dims=False,  
                                   has_bias=True, _batch_minor=False, ker_in_order=None,  
                                   ker_out_order=None, **kwargs)
```

```
class nntool.api.types.PoolingNodeBase(*args, pool_type='max', **kwargs)
```

```
class nntool.api.types.MaxPoolNode(*args, pool_type='max', **kwargs)
```

```
class nntool.api.types.AveragePoolNode(*args, pool_type='max', **kwargs)
```

```
class nntool.api.types.GlobalPoolingNodeBase(*args, pool_type='max', axis=None, keep_dims=None,  
                                             **kwargs)
```

```
class nntool.api.types.GlobalMaxPoolNode(*args, pool_type='max', axis=None, keep_dims=None,  
                                         **kwargs)
```

```
class nntool.api.types.GlobalMinPoolNode(*args, pool_type='max', axis=None, keep_dims=None,  
                                         **kwargs)
```

```
class nntool.api.types.GlobalSumPoolNode(*args, pool_type='max', axis=None, keep_dims=None,  
                                         **kwargs)
```

```
class nntool.api.types.GlobalAveragePoolNode(*args, pool_type='max', axis=None, keep_dims=None,  
                                              **kwargs)
```

## RNN Filter classes

```
class nnntool.api.types.LSTMNode(name, *args, cell_clip=0.0, proj_clip=0.0, **kwargs)
class nnntool.api.types.GRUJNode(*args, linear_before_reset=False, activation_zr=None, **kwargs)
class nnntool.api.types.RNNNode(name, *args, n_cells=None, n_states=None, n_inputs=None,
                                n_input_cells=None, n_output_cells=None, activation='tanh', revert=False,
                                output_directions=False, **kwargs)
```

## nnntool.api.validation

**class nnntool.api.validation.ValidateBase**

**validate**(*input\_tensors*: Sequence[numpy.ndarray], *output\_tensors*: Sequence[Sequence[numpy.ndarray]],  
*input\_name*: Optional[str] = None) → nnntool.utils.validation\_utils.ValidationResultBase  
 Validate an execution result

**Parameters**

- **input\_tensors** (Sequence[np.ndarray]) – Input tensors provided to execute
- **output\_tensors** (Sequence[Sequence[np.ndarray]]) – Output tensors provided by execute
- **input\_name** (str, optional) – Name of input file. Defaults to None.

**Returns** Result of validation

**Return type** ValidationResultBase

**class nnntool.api.validation.ValidateFromClass**(*class\_number*, *type\_of\_prediction*='classification',  
\*\*kargs)

**class nnntool.api.validation.ValidateFromJSON**(*json\_file*, \*\*kargs)

**class nnntool.api.validation.ValidateFromName**(*class\_thr*=0, *binary\_classification*=False,  
*type\_of\_prediction*='classification', *out\_idx*=-1)

**class nnntool.api.validation.ValidateFromVWIInstances**(*instances\_file*, \*\*kargs)

**class nnntool.api.validation.ValidationResultBase**

Base class for validation results

**property margin: float**

**What is the margin of the result. Value between 0 and 1.** Can be overriden in derived class. Return 0.0 if not used.

**Returns** Margin of result

**Return type** float

**abstract property validated: bool**

Is result valid. Overriden in derived class.

**Returns** True if result is valid

**Return type** bool

## 6.7 SSBL

**Warning:** This is only for GAP9

**Warning:** This is **experimental/work-in-progress** work. **DO NOT USE.** Many features are not present. It's not yet to be used in production.

The SSBL (Second-Stage BootLoader) is executed just after the ROM. It includes the following boot modes:

- boot from application in MRAM (with CRC verification)
- boot from JTAG
- boot from recovery mode (JTAG mram flasher)

These boot modes are controlled via pins 60 and 62. if pin 60 is 0, SSBL boots from app. Else if pin 62 is 0, boots from JTAG. Else it boots from recovery mode.

This SSBL does not include the automatic recovery process if the boot from application failed.

It only supports applications that do not use extra partitions such as filesystems. It does not yet support applications using the XIP.

### 6.7.1 Usage

First, compile and flash the SSBL:

```
cd $GAP_SDK_HOME/utils/ssbl  
make all CONFIG_BOOT_DEVICE=mram
```

After the SSBL has been flashed, put it in boot from recovery mode by putting both pins 60 and 62 to 1, and resetting the board.

Then flash any application such as the helloworld by calling this script:

```
$GAP_SDK_HOME/utils/ssbl/ota_scripts/flash.sh <tmp_directory> <path_to_the_application_  
elf>
```

This require to compile the application you want to flash first. You can also put *io=uart* when compiling to have access to *printf* via UART.

Once it's done, put the SSBL in application mode by putting the pin 60 to 0, and reset the board. Your application should now be executed.

## API REFERENCE

This sections contains documentation extracted from source code.

### 7.1 PMSIS API

#### 7.1.1 Introduction

The PMSIS API is a set of low-level drivers which any operating system can implement to provide a common layer to upper layers. Together with the PMSIS BSP, it provides a full stack of drivers, allowing the development of applications portable across a wide range of operating systems.

#### 7.1.2 Conventions

All functions prefixed by `pi_` can only be called from fabric-controller side while the ones prefixed by `pi_cl_` can only be called from cluster side. Any exception to these rules is documented where it applies.

All functions on fabric-controller side are by default synchronous and are blocking the caller until the operation is done. All the functions suffixed by `_async` are asynchronous and are not blocking the caller. The termination of such operations is managed with a `pi_evt_t` object, see PMSIS API documentation for more information.

Functions on cluster-side are by default synchronous but can also be asynchronous if the documentation of the function mentions it.

#### 7.1.3 Contents

##### RTOS

###### Task

*group* **Task**

OS tasks management.

This part details tasks management along with semaphores and mutexes.

## TypeDefs

```
typedef void (*func_t)(void *arg)
Function type.
Function type taking a pointer to void and returning void.
```

## Functions

```
int32_t pi_os_open(struct pi_device *device)
Open OS.
```

### Parameters

- **device** – Pointer to pi\_device.

### Returns

```
void pi_os_close(struct pi_device *device)
Close OS.
```

### Parameters

- **device** – Pointer to pi\_device.

```
static inline int32_t pmsis_kickoff(void *arg)
Kickoff the system.
```

### *Deprecated:*

This function is no longer useful. It's called automatically now. You just have to put your code in the 'main' function.

This function starts the system, prepares the event kernel, IRQ,... Completely OS dependant might do anything from a function call to main task creation.

---

**Note:** This function must be called in the main in order to launch the event kernel, enable IRQ, create the main task and start the scheduler.

---

### Parameters

- **arg** – Parameter given to main task/thread.

### Returns

**0** If operation is successful.

**ERRNO** An error code otherwise.

```
static inline void pmsis_exit(int32_t err)
Stop runtime and exit platform.
```

This function stops runtime and exits platform. Once it is called, there is no return.

### Parameters

- **err** – Exit status.

```
static inline void *pi_thread_create(pi_task_entry_t func, void *arg, char *name, uint32_t stack_size,
int32_t priority)
```

Create a thread.

This function creates an OS thread.

#### Parameters

- **func** – Thread entry function.
- **arg** – Arguments to thread's entry function.
- **name** – Name of the thread.
- **stack\_size** – Stack size given to the thread.
- **priority** – Thread priority.

**Returns thread** Pointer to created thread.

**Returns NULL** If thread has not been created.

```
static inline void *pi_user_thread_create(pi_task_entry_t func, void *arg, char *name, uint32_t
                                         stack_size, int32_t priority)
```

Create a usermode thread.

This function creates a usermode thread.

#### Parameters

- **func** – Thread entry function.
- **arg** – Arguments to thread's entry function.
- **name** – Name of the thread.
- **stack\_size** – Stack size given to the thread.
- **priority** – Thread priority.

**Returns thread** Pointer to created thread.

**Returns NULL** If thread has not been created.

```
static inline char *pi_thread_get_name(void *thread)
```

Get thread's name.

This function returns the name of a thread.

#### Parameters

- **thread** – Pointer to the thread to get the name.

**Returns** Pointer to a char sequence, with the name of the thread.

```
static inline void pi_thread_suspend(void *thread)
```

Suspend a thread.

Calling this function suspends a thread. If given parameter thread is NULL, then the calling thread is suspended, and only an other thread can resume it.

#### Parameters

- **thread** – Pointer to the thread to suspend.

```
static inline void pi_thread_resume(void *thread)
```

Resume a thread.

Calling this function resumes a suspended thread.

#### Parameters

- **thread** – Pointer to the thread to suspend.

```
static inline void pi_thread_delete(void *thread)
Delete a thread.

Calling this function terminate a thread, and delete it.
```

#### Parameters

- **thread** – Pointer to the thread to delete.

```
static inline void pi_yield()
Call OS to switch context.
```

This function can be called when a task should let another task run on CPU. In case of waiting for the end of some transfers, synchronisation,...

## **Event\_Task**

*group* **Event\_Task**  
Event\_Task management.

The asynchronous interactions between the fabric controller, the cluster and the peripherals are managed with events on the fabric controller side.

A task is either a function callback which can be pushed for a deferred execution when it is triggered, or notification event, which can used to block the caller execution until it is triggered.

#### Defines

```
pi_task_block(event)
```

```
pi_task_callback(evt, function, arg)
```

```
pi_task_irq_callback(evt, function, arg)
```

```
pi_task_wait_on(evt)
```

```
pi_evt_wait_on(evt)
```

```
pi_task_push(evt)
```

```
pi_task_push_irq_safe(evt)
```

```
pi_task_push_delayed_us(evt, delay)
```

```
pi_task_cancel_delayed_us(evt)
```

```
pi_task_timeout_set(evt, timeout_us)
```

```
pi_task_status_get(evt)
```

---

**pi\_task\_status\_set**(evt, status)

## Functions

static inline pi\_evt\_t \***pi\_evt\_sig\_init**(pi\_evt\_t \*event)

Prepare a notification event.

This initializes a notification event so that it is ready to be triggered. A notification event can be used to block the execution of the caller (using pi\_evt\_wait) until a certain action occurs, e.g. an end of transfer.

---

**Note:** This structure is allocated by the caller and must be kept alive until the pi\_evt\_wait returns.

---

**Note:** If the same notification is re-used several times, it must be reinitialized everytime by calling this function or another variant.

### Parameters

- **event** – Pointer to notification event.

**Returns** event The notification event initialized.

static inline pi\_evt\_t \***pi\_evt\_callback\_no\_irq\_init**(pi\_evt\_t \*event, pi\_callback\_func\_t function, void \*arg)

Prepare a notification callback.

This initializes a notification callback so that it is ready to be triggered. A notification callback can be used to trigger a function execution when a certain action occurs, e.g. an end of transfer.

---

**Note:** This structure is allocated by the caller and must be kept alive until the pi\_evt\_wait returns.

---

**Note:** If the same notification is re-used several times, it must be reinitialized everytime by calling this function or another variant.

---

**Note:** A notification callback can not be used to block the caller execution with pi\_evt\_wait.

### Parameters

- **event** – Pointer to notification event.
- **function** – Callback function to execute when the notification is triggered.
- **arg** – Callback function argument.

**Returns** event The notification event initialized.

static inline pi\_evt\_t \***pi\_evt\_callback\_irq\_init**(pi\_evt\_t \*event, pi\_callback\_func\_t function, void \*arg)

Prepare an IRQ notification callback.

This initializes an IRQ notification callback so that it is ready to be triggered. An IRQ notification callback can be used to trigger a function execution when a certain action occurs, e.g. an end of transfer. Compared to a normal callback, an IRQ callback will be called directly from the interrupt handler and will not go through the scheduler.

---

**Note:** This structure is allocated by the caller and must be kept alive until the pi\_evt\_wait returns.

---

**Note:** If the same notification is re-used several times, it must be reinitialized everytime by calling this function or another variant.

---

**Note:** A notification callback can not be used to block the caller execution with pi\_evt\_wait.

---

#### Parameters

- **event** – Pointer to notification event.
- **function** – Callback function to execute when the notification is triggered.
- **arg** – Callback function argument.

**Returns** event The notification event initialized.

```
static inline void pi_evt_wait(pi_evt_t *event)  
Wait until a notification event is triggered.
```

This can be called to block the caller until the specified notification event (created with pi\_evt\_sig\_init) has been triggered.

---

**Note:** The notification event is released just before returning from this call and must be reinitialized before it can be re-used.

---

#### Parameters

- **event** – Pointer to notification event.

```
static inline void pi_evt_push(pi_evt_t *event)  
Trigger a notification.
```

This can be used to trigger the specified notification. If the notification is a callback, this will schedule the callback execution. If the notification is an event, this will trigger the event.

#### Parameters

- **event** – Pointer to notification event.

```
static inline void pi_evt_push_from_irq(pi_evt_t *event)  
Trigger a notification.
```

This can be used to trigger the specified notification. If the notification is a callback, this will schedule the callback execution. If the notification is an event, this will trigger the event.

#### Parameters

- **event** – Pointer to notification event.

---

```
void pi_evt_push_delayed_us(pi_evt_t *event, uint32_t delay)
Trigger a notification.
```

This can be used to trigger the specified notification after the specified delay, given in micro-seconds. If the notification is a callback, this will schedule the callback execution. If the notification is an event, this will trigger the event.

#### Parameters

- **event** – Pointer to notification event.
- **delay** – The number of micro-seconds after which the notification must be triggered.

```
void pi_evt_cancel_delayed_us(pi_evt_t *event)
Cancel a scheduled notification.
```

This can be cancel a scheduled notification. If the notification has already been canceled, this has no effect.

#### Parameters

- **event** – Pointer to notification event.

```
static inline pi_callback_t *pi_callback_init(pi_callback_t *callback, pi_callback_func_t function, void
*arg)
```

Init callback.

Initialize a simple callback with the function to call and its arg.

#### Parameters

- **callback** – Pointer to callback to initialize.
- **function** – Callback function.
- **arg** – Callback function arg.

```
static inline void pi_evt_timeout_set(pi_evt_t *event, uint32_t timeout_us)
```

Init timeout feature for transfers.

Initialize timeout value of the event before starting transfers.

---

**Note:** This event will hold transfer result.

---

#### Parameters

- **event** – Pointer to event.
- **timeout\_us** – Timeout value in us.

```
static inline int32_t pi_evt_status_get(pi_evt_t *event)
```

Get event status.

This function can be used to check if an event completed successfully.

**Returns** ERRNO Value corresponding to event status.

```
static inline void pi_evt_status_set(pi_evt_t *event, int32_t status)
```

Set event status.

This function can be used to tell if an event completed successfully.

#### Parameters

- **status** – Value corresponding to event status.

## Memory allocation

### group **MemAlloc**

Memory management.

This provides support for memory allocation in internal memories of GAP8/GAP9.

There are 3 types of memory :

- L2 memory : L2 memory used by both FC and cluster cores(using delegation).
- FC TCDM memory or L2 Private : L1 memory close to Fabric Controller.
- Cluster L1 memory : L1 memory used by the cluster cores and NE16.

## Functions

`int32_t pi_alloc_fail(pi_allocator_name_e allocator, int32_t size)`  
Called if an allocation failed.

---

**Note:** Function is **weak** and can be overriden default implementation will dump the allocator state

---

### Parameters

- **a** – allocator structure
- **size** – size of the failed allocation

`void pi_malloc_init(void *fc_heap_start, uint32_t fc_heap_size, void *l2_heap_start, uint32_t l2_heap_size)`

Initialize the memory allocators.

This function initializes the FC and L2 memory allocators to allocate from the FC or L2 memory heap.

---

**Note:** If this function is called first, those subfunctions do not need to be called.

---

### Parameters

- **fc\_heap\_start** – FC memory heap base address.
- **fc\_heap\_size** – FC memory heap size.
- **l2\_heap\_start** – L2 memory heap base address.
- **l2\_heap\_size** – L2 memory heap size.

`void *pi_malloc(size_t size)`

Allocate memory from FC or L2 memory allocator.

This function allocates a memory chunk in FC if there is enough memory to allocate otherwise in L2.  
BEWARE, in GAP9, it will allocate memory from L2 ONLY.

### Parameters

- **size** – Size of the memory to be allocated.

**Returns** Pointer to an allocated memory chunk or NULL if there is not enough memory to allocate.

**void \*pi\_malloc\_align(size\_t size, uint32\_t align)**  
Allocate memory from FC or L2 memory allocator with aligned address.

This function allocates an address aligned memory chunk in FC if there is enough memory to allocate required chunk of memory otherwise in L2.

#### Parameters

- **size** – Size of the memory to be allocated.
- **align** – Memory alignment size.

**Returns** Pointer to an allocated memory chunk or NULL if there is not enough memory to allocate.

**void pi\_free(void \*\_chunk)**  
Free an allocated memory chunk.

This function frees an allocated memory chunk.

#### Parameters

- **\_chunk** – Start address of an allocated memory chunk.

**void pi\_malloc\_dump(void)**  
Display free blocks.

This function displays free blocks available in either FC or L2 memory.

### group L2\_Malloc

Memory allocation in L2 memory from FC.

## Functions

**void \*pi\_l2\_malloc(int32\_t size)**  
Allocate in L2 memory.

The allocated memory is 4-bytes aligned. The caller has to provide back the size of the allocated chunk when freeing it.

#### Parameters

- **size** – Size in bytes of the memory to be allocated.

**Returns** The allocated chunk or NULL if there was not enough memory available.

**void pi\_l2\_free(void \*chunk, int32\_t size)**  
Free L2 memory.

#### Parameters

- **chunk** – Chunk to be freed.
- **size** – Size in bytes of the memory to be freed.

**void \*pi\_l2\_malloc\_align(int32\_t size, int32\_t align)**  
Allocate in L2 memory.

The allocated memory is aligned on the specified number of bytes. The caller has to provide back the size of the allocated chunk when freeing it.

#### Parameters

- **size** – Size in bytes of the memory to be allocated.
- **align** – Alignment in bytes.

**Returns** The allocated chunk or NULL if there was not enough memory available.

void **pi\_l2\_malloc\_dump**(void)

Display free blocks.

This function can be used to display free blocks available from the L2 allocator.

void **pi\_l2\_available\_get**(uint32\_t \*totalMemAvailable, uint32\_t \*largestMemAllocatable)

Get available blocks in L2.

This function can be used to get free blocks available from the L2 allocator.

#### Parameters

- **totalMemAvailable** – Size in bytes of total available L2.
- **largestMemAllocatable** – Size in bytes of largest free block.

void **pi\_fc\_l1\_available\_get**(uint32\_t \*totalMemAvailable, uint32\_t \*largestMemAllocatable)

Get available blocks in L2 Private.

This function can be used to get free blocks available from the L2 private.

#### Parameters

- **totalMemAvailable** – Size in bytes of total available L2.
- **largestMemAllocatable** – Size in bytes of largest free block.

void **pi\_cl\_l1\_available\_get**(uint32\_t \*totalMemAvailable, uint32\_t \*largestMemAllocatable)

Get available blocks in cluster L1.

This function can be used to get free blocks available from the L1 in cluster.

#### Parameters

- **totalMemAvailable** – Size in bytes of total available L2.
- **largestMemAllocatable** – Size in bytes of largest free block.

### group CL\_L1\_Malloc

Memory allocation in Cluster L1 memory.

#### Functions

void \***pi\_cl\_l1\_malloc**(struct pi\_device \*device, uint32\_t size)

Allocate in Cluster L1 memory.

The allocated memory is 4-bytes aligned. The caller has to provide back the size of the allocated chunk when freeing it. This can be called only when the specified cluster is opened.

#### Parameters

- **device** – Cluster device where to allocate memory.
- **size** – Size in bytes of the memory to be allocated.

**Returns** The allocated chunk or NULL if there was not enough memory available.

---

```
void pi_cl_l1_free(struct pi_device *device, void *chunk, int32_t size)
    Free Cluster L1 memory.
```

This can be called only when the specified cluster is opened.

#### Parameters

- **device** – Cluster device where to free memory.
- **chunk** – Chunk to be freed.
- **size** – Size in bytes of the memory to be freed.

```
void *pi_cl_l1_malloc_align(struct pi_device *device, int32_t size, int32_t align)
    Allocate in Cluster L1 memory.
```

The allocated memory is aligned on the specified number of bytes. The caller has to provide back the size of the allocated chunk when freeing it. This can be called only when the specified cluster is opened.

#### Parameters

- **device** – Cluster device where to allocate memory.
- **size** – Size in bytes of the memory to be allocated.
- **align** – Alignment in bytes.

**Returns** The allocated chunk or NULL if there was not enough memory available.

```
void pi_cl_l1_malloc_dump(struct pi_device *device)
    Display free blocks.
```

This function can be used to display free blocks available from the CL L1 allocator.

### group **FC\_L1\_Malloc**

Memory allocation in FC TCDM memory(GAP8) or L2(GAP9).

#### Functions

```
void *pi_fc_l1_malloc(int32_t size)
    Allocate in FC L1 memory.
```

The allocated memory is 4-bytes aligned. The caller has to provide back the size of the allocated chunk when freeing it. This will allocate in the closest memory for the FC, which can vary depending on the chip. Check the chip-specific section for more information.

#### Parameters

- **size** – Size in bytes of the memory to be allocated.

**Returns** The allocated chunk or NULL if there was not enough memory available.

```
void pi_fc_l1_free(void *chunk, int32_t size)
    Free FC L1 memory.
```

#### Parameters

- **chunk** – Chunk to be freed.
- **size** – Size in bytes of the memory to be freed.

```
void *pi_fc_l1_malloc_align(int32_t size, int32_t align)
    Allocate in FC L1 memory.
```

The allocated memory is aligned on the specified number of bytes. The caller has to provide back the size of the allocated chunk when freeing it.

#### Parameters

- **size** – Size in bytes of the memory to be allocated.
- **align** – Alignment in bytes.

**Returns** The allocated chunk or NULL if there was not enough memory available.

```
void pi_fc_11_malloc_dump(void)  
    Display free blocks.
```

This function can be used to display free blocks available from the FC allocator.

## Errors Definition

*group* **Erro\_Def**

### Enums

enum **pi\_err\_t**

RTOS Errors' definition.

*Values:*

enumerator **PI\_OK** = 0x00

indicating success (no error)

enumerator **PI\_FAIL** = 0x01

Generic code indicating failure

enumerator **PI\_ERR\_INVALID\_ARG** = 0x02

Invalid argument

enumerator **PI\_ERR\_INVALID\_STATE** = 0x03

Invalid state

enumerator **PI\_ERR\_INVALID\_SIZE** = 0x04

Invalid size

enumerator **PI\_ERR\_NOT\_FOUND** = 0x05

Requested resource not found

enumerator **PI\_ERR\_NOT\_SUPPORTED** = 0x06

Operation or feature not supported

enumerator **PI\_ERR\_TIMEOUT** = 0x07

Operation timed out

enumerator **PI\_ERR\_INVALID\_CRC** = 0x08

CRC or checksum was invalid

---

enumerator **PI\_ERR\_INVALID\_VERSION** = 0x09  
 Version was invalid

enumerator **PI\_ERR\_INVALID\_APP** = 0x0A  
 App binary is not compliant with GAP.

enumerator **PI\_ERR\_INVALID\_MAGIC\_CODE** = 0x0B  
 Magic code does not match.

enumerator **PI\_ERR\_ALREADY\_EXISTS** = 0x0C

enumerator **PI\_ERR\_I2C\_NACK** = 0x100  
 An item already exists.

enumerator **PI\_ERR\_NO\_MEM** = 0x200  
 I2C request ended with a NACK Generic out of memory

enumerator **PI\_ERR\_L2\_NO\_MEM** = 0x201  
 L2 out of memory

## Drivers

### AES

*group AES*  
 AES driver.

The AES API provides support for encryption and decryption using hardware AES module.

To use a uDMA FIFO with the AES, give the FIFO ID as source and/or destination address. If you use a FIFO as destination, you will need to manually stop the FIFO mode when you are done using it with an ioctl command.

### TypeDefs

typedef struct *pi\_aes\_conf* **pi\_aes\_conf\_t**

### Enums

enum **pi\_aes\_mode\_e**  
 AES block cipher modes.

*Values:*

enumerator **PI\_AES\_MODE\_ECB** = 0  
 enumerator **PI\_AES\_MODE\_CBC** = 1  
 enumerator **PI\_AES\_MODE\_CTR** = 2

**enum pi\_aes\_key\_len\_e**

AES key lengths.

*Values:*

enumerator **PI\_AES\_KEY\_128** = 0

enumerator **PI\_AES\_KEY\_256** = 1

**enum pi\_aes\_ioctl\_e**

Commands for pi\_aes\_ioctl.

This is used to tell which command to execute through pi\_aes\_ioctl.

*Values:*

enumerator **PI\_AES\_IOCTL\_STOP\_FIFO\_MODE**

Exit FIFO mode and resume operation in normal mode.

After a FIFO has been used as output of the AES, this command must be sent to return to the normal mode.

## Functions

**void pi\_aes\_conf\_init(struct pi\_aes\_conf \*conf)**

Initialize configuration structure with default values.

**Parameters**

- **conf** – Configuration structure to initialize

**int32\_t pi\_aes\_open(struct pi\_device \*device)**

Allocate and initialize AES device software structures with previously provided configuration contained in **device**. Reference to the configuration should be previously set into **device** by **pi\_open\_from\_conf()** call.

Multiple devices can be opened at the same time on the same AES hardware module. If **pi\_aes\_open()** is called for the first time on an AES hardware module, it will also power-up the AES module and allocate uDMA linear tx and rx channels.

**Parameters**

- **device** – AES device descriptor

**Returns 0** Success

**Returns <errno>** Error code

**void pi\_aes\_close(struct pi\_device \*device)**

Deallocate configuration/context held by this **device** descriptor. If this was the only device opened on an AES hardware module, it will also power down the module and release related uDMA channels.

**Parameters**

- **device** – AES device descriptor

**void pi\_aes\_ioctl(pi\_device\_t \*device, uint32\_t cmd, void \*arg)**

AES IOCTL function.

**Parameters**

- **device** – AES device descriptor
- **cmd** – ioctl number. See [pi\\_aes\\_ioctl\\_e](#) for the list of available ioctl identifiers.
- **arg** – Argument specific to the given cmd

`int32_t pi_aes_encrypt(struct pi_device *device, void *src, void *dest, uint16_t len)`  
Encrypt data (synchronous)

The call initiates encryption of data from **src**, blocks until the processing is complete and returns encrypted data via **dest**.

**Warning:** Source data size (in bytes) must be a multiple of 16. User must handle the padding.

#### Parameters

- **device** – AES device descriptor
- **src** – Data to encrypt
- **dest** – Encrypted data
- **len** – Data length as number of 32-bit words. Must be multiple of 4.

**Returns 0** Success

**Returns <errno>** Error code

`int32_t pi_aes_encrypt_async(struct pi_device *device, void *src, void *dest, uint16_t len, pi_evt_t *evt)`  
Encrypt data (asynchronous)

The call initiates encryption of data given by **src**, registers the event **task** for the caller to be notified on once the processing is complete, and returns.

**Warning:** Source data size (in bytes) should be a multiple of 16. User must handle the padding if needed.

#### Parameters

- **device** – AES device descriptor
- **src** – Data to encrypt
- **dest** – Encrypted data
- **len** – Data length as number of 32-bit words. Must be multiple of 4.
- **evt** – Event to be scheduled after encryption is finished

**Returns 0** Success

**Returns <errno>** Error code

`int32_t pi_aes_decrypt(struct pi_device *device, void *src, void *dest, uint16_t len)`  
Decrypt data (synchronous)

The call initiates decryption of data from **src**, blocks until the processing is complete and returns encrypted data via **dest**.

**Warning:** Source data size (in bytes) should be a multiple of 16. User must handle the padding if needed.

#### Parameters

- **device** – AES device descriptor
- **src** – Data to decrypt
- **dest** – Decrypted data
- **len** – Data length as number of 32-bit words. Must be multiple of 4.

**Returns 0** Success

**Returns <errno>** Error code

```
int32_t pi_aes_decrypt_async(struct pi_device *device, void *src, void *dest, uint16_t len, pi_evt_t *evt)
```

Decrypt data (asynchronous)

The call initiates decryption of data given by **src**, registers the event **task** for the caller to be notified on once the processing is complete, and returns.

**Warning:** Source data size (in bytes) should be a multiple of 16. User must handle the padding if needed.

#### Parameters

- **device** – AES device descriptor
- **src** – Data to decrypt
- **dest** – Decrypted data
- **len** – Data length as number of 32-bit words. Must be multiple of 4.
- **evt** – Event executed after the decryption

**Returns 0** Success

**Returns <errno>** Error code

```
int32_t pi_aes_ctr_set(pi_device_t *device, uint32_t low, uint32_t high)
```

For AES module with CTR support, allows to manually set counter value for the next transfer.

**Warning:** It is relevant only if AES hardware module has CTR support.

#### Parameters

- **device** – AES device descriptor
- **low** – Lower 32 bits of the counter
- **high** – Upper 32 bits of the counter

**Returns 0** Success

**Returns <errno>** Error code

---

```
struct pi_aes_conf
#include <aes.h> Structure describing the configuration of an AES device.
```

### Public Members

**uint8\_t `itf`**  
AES device ID.

*pi\_aes\_mode\_e `mode`*  
AES mode

*pi\_aes\_key\_len\_e `key_len`*  
AES key length: 128 or 256 bits

**uint32\_t \*`key`**  
AES key

**uint32\_t \*`iv`**  
Initialization vector (used in CBC and CTR modes)

## Cluster

### FC/cluster synchronization

#### group **FcClusterSync**

This set of functions provide support for controlling clusters from fabric-controller side.

### Enums

enum **pi\_cluster\_flags\_e**  
Cluster configuration flags.

*Values:*

enumerator **PI\_CLUSTER\_FLAGS\_FORK\_BASED** = (0 << 0)  
Start the cluster with a fork-based execution model.

enumerator **PI\_CLUSTER\_FLAGS\_TASK\_BASED** = (1 << 0)  
Start the cluster with a task-based execution model.

## Functions

```
void pi_cl_send_task_to_fc(pi_evt_t *task)
    Enqueue a task to fabric-controller side.
```

This enqueues the specified task into the fabric-controller task scheduler for execution. The task must have been initialized from fabric-controller side.

### Parameters

- **task** – Pointer to the fabric-controller task to be enqueued.

```
static inline void pi_cl_send_callback_to_fc(pi_callback_t *callback)
    Send a callback to Fabric Controller.
```

This function is used to send a simple callback to FC.

---

**Note:** This is an alternative to *pi\_cl\_send\_task\_to\_fc()*.

---

### Parameters

- **callback** – Pointer to callback(with function and arg).

```
void pi_cluster_conf_init(struct pi_cluster_conf *conf)
    Initialize a cluster configuration with default values.
```

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the SPI device is opened.

### Parameters

- **conf** – A pointer to the SPI master configuration.

```
int pi_cluster_open(struct pi_device *device)
    Open and power-up the cluster.
```

This function must be called before the cluster device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. By default the cluster is powered down and cannot be used. Calling this function will power it up. At the end of the call, the cluster is ready to execute a task. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
int pi_cluster_close(struct pi_device *device)
    Close an opened cluster device.
```

This function can be called to close an opened cluster device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used. This will power-down the cluster. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the structure describing the device.

---

```
static inline struct pi_cluster_task *pi_cluster_task(struct pi_cluster_task *task, void (*entry)(void*), void *arg)
```

Prepare a cluster task for execution.

This initializes a cluster task before it can be sent to the cluster side for execution. If the same task is re-used for several executions, it must be reinitialized everytime by calling this function.

#### Parameters

- **task** – A pointer to the structure describing the task. This structure is allocated by the caller and must be kept alive until the task has finished execution.
- **entry** – The task entry point that the cluster controller will execute.
- **arg** – The argument to the entry point.

```
static inline pi_cluster_task_t *pi_cluster_task_stacks(pi_cluster_task_t *task, void *stacks, int stack_size)
```

Specify cluster task stack information.

This can be called to configure the size of the stacks and to specify stacks allocated by the caller.

#### Parameters

- **task** – The task for which the stack information is being specified.
- **stacks** – Pointer to the memory which should be used for the stacks.
- **arg** – The argument to the entry point.

```
static inline void pi_cluster_task_priority(pi_cluster_task_t *task, uint8_t priority)
```

Specify the cluster task priority.

This sets the priority of the specified cluster task. Only priorities 0 and 1 are currently supported. The cluster driver provides on cluster side a cooperative priority-based scheduler. A currently running task is in charge of periodically checking if it must release the cluster to let a higher priority task execute.

#### Parameters

- **task** – The cluster task
- **priority** – The cluster task priority, can be 0 or 1.

```
static inline int pi_cluster_send_task(struct pi_device *device, struct pi_cluster_task *task)
```

Enqueue a task for execution on the cluster.

This will enqueue the task at the end of the queue of tasks, ready to be executed by the specified cluster. Once the task gets scheduled, the cluster-controller core is waken-up and starts executing the task entry point. This function is intended to be used for coarse-grain job delegation to the cluster side, and thus the stack used to execute this function can be specified. When the function starts executing on the cluster, the other cores of the cluster are also available for parallel computation. Thus the stacks for the other cores (called slave cores) can also be specified, as well as the number of cores which can be used by the function on the cluster (including the cluster controller). The caller is blocked until the task has finished execution. This function only supports task with priority 0. *pi\_cluster\_enqueue* should be used instead to use priority 1.

Note that this enqueues a function execution. To allow cluster executions to be pipelined, several tasks can be enqueued at the same time. If more than two tasks are enqueued, as soon as the first is finished, the cluster-controller core immediately continues with the next one, while the fabric controller receives the termination notification and can enqueue a new execution, in order to keep the cluster busy.

#### Parameters

- **device** – A pointer to the structure describing the device.

- **task** – Cluster task structure containing task and its parameters.

```
static inline int pi_cluster_send_task_async(struct pi_device *device, struct pi_cluster_task  
                                         *cluster_task, pi_evt_t *task)
```

Enqueue asynchronously a task for execution on the cluster.

This will enqueue the task at the end of the queue of tasks, ready to be executed by the specified cluster. Once the task gets scheduled, the cluster-controller core is waken-up and starts executing the task entry point. This function is intended to be used for coarse-grain job delegation to the cluster side, and thus the stack used to execute this function can be specified. When the function starts executing on the cluster, the other cores of the cluster are also available for parallel computation. Thus the stacks for the other cores (called slave cores) can also be specified, as well as the number of cores which can be used by the function on the cluster (including the cluster controller). The task is just enqueued and the caller continues execution. A task must be specified in order to specify how the caller should be notified when the task has finished execution. This function only supports task with priority 0. `pi_cluster_enqueue_task_async` should be used instead to use priority 1.

Note that this enqueues a function execution. To allow cluster executions to be pipelined, several tasks can be enqueued at the same time. If more than two tasks are enqueued, as soon as the first is finished, the cluster-controller core immediately continues with the next one, while the fabric controller receives the termination notification and can enqueue a new execution, in order to keep the cluster busy.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **cluster\_task** – Cluster task structure containing task and its parameters.
- **task** – The task used to notify the end of execution.

```
int pi_cluster_enqueue_task(struct pi_device *device, struct pi_cluster_task *task)
```

Enqueue a task for execution on the cluster.

This function is similar to `pi_cluster_send_task` but supports priority 0 and 1 and do not support automatic stack allocation. Stacks but always be allocated by the caller. This will enqueue the task at the end of the queue of tasks, ready to be executed by the specified cluster. Once the task gets scheduled, the cluster-controller core is waken-up and starts executing the task entry point. This function is intended to be used for coarse-grain job delegation to the cluster side, and thus the stack used to execute this function can be specified. When the function starts executing on the cluster, the other cores of the cluster are also available for parallel computation. Thus the stacks for the other cores (called slave cores) can also be specified, as well as the number of cores which can be used by the function on the cluster (including the cluster controller). The caller is blocked until the task has finished execution.

Note that this enqueues a function execution. To allow cluster executions to be pipelined, several tasks can be enqueued at the same time. If more than two tasks are enqueued, as soon as the first is finished, the cluster-controller core immediately continues with the next one, while the fabric controller receives the termination notification and can enqueue a new execution, in order to keep the cluster busy.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **task** – Cluster task structure containing task and its parameters.

```
int pi_cluster_enqueue_task_async(struct pi_device *device, struct pi_cluster_task *task, pi_evt_t  
                                 *async_task)
```

Enqueue asynchronously a task for execution on the cluster.

This function is similar to `pi_cluster_send_task_async` but supports priority 0 and 1 and do not support automatic stack allocation. Stacks but always be allocated by the caller. This will enqueue the task at the end of the queue of tasks, ready to be executed by the specified cluster. Once the task gets scheduled, the

cluster-controller core is waken-up and starts executing the task entry point. This function is intended to be used for coarse-grain job delegation to the cluster side, and thus the stack used to execute this function can be specified. When the function starts executing on the cluster, the other cores of the cluster are also available for parallel computation. Thus the stacks for the other cores (called slave cores) can also be specified, as well as the number of cores which can be used by the function on the cluster (including the cluster controller). The task is just enqueued and the caller continues execution. A task must be specified in order to specify how the caller should be notified when the task has finished execution.

Note that this enqueues a function execution. To allow cluster executions to be pipelined, several tasks can be enqueued at the same time. If more than two tasks are enqueued, as soon as the first is finished, the cluster-controller core immediately continues with the next one, while the fabric controller receives the termination notification and can enqueue a new execution, in order to keep the cluster busy.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **cluster\_task** – Cluster task structure containing task and its parameters.
- **task** – The task used to notify the end of execution.

`static inline int pi_cl_task_yield()`

Check if the current task should release the cluster.

Since there is no preemption on cluster side, a low priority running task must periodically check if the cluster should be released by calling this function. If so the task should return, and the entry point will be called again later on to resume execution. Note that if this function tells that the task should return, this will force the entry point to be called again later on.

**Returns** 1 if the current task should release the cluster or 0 if it can keep executing.

`void *pi_cl_l1_scratch_alloc(pi_device_t *device, pi_cluster_task_t *task, int size)`

Allocate L1 memory from the task scratch area.

The scratch area is an area within the L1 memory reserved during cluster configuration for data which does not need to be kept when a cluster task ends or is suspended to let another higher priority task execute. Each task is having its own linear scratch allocator so that such memory area can be reused from one task to another. Calling this function will allocate scratch data from the specified task in a linear way, which means the allocated data must be freed in reverse order. Allocated data do not need to be freed when the task is over since it is just a pointer being increased or decreased. This allocator is reset every time the task is reset, for example by calling `pi_cluster_task`. The amount of scratch data allocated for the specified task can not exceed the size of the scratch area specified when the cluster was opened (0 by default).

#### Parameters

- **device** – The cluster device where the data is being allocated.
- **task** – The cluster task on which the scratch data must be allocated.
- **size** – Size of the allocated data.

**Returns** pointer The allocated area if the allocation succeeded.

**Returns** NULL In case the allocation failed.

`void pi_cl_l1_scratch_free(pi_device_t *device, pi_cluster_task_t *task, int size)`

Free L1 memory from the task scratch area.

Calling this function will free the specified amount of data in the scratch allocator. No data needs to be provided since the allocator works linearly.

#### Parameters

- **device** – The cluster device where the data is being freed.

- **task** – The cluster task on which the scratch data must be freed.
- **size** – Size of the freed data.

```
struct pi_cluster_conf
#include <cl_pmsis_types.h> Cluster configuration structure.
```

This structure is used to pass the desired cluster configuration to the runtime when opening a cluster.

### Public Members

pi\_device\_e **device\_type**

Device type.

int **id**

Cluster ID, starting from 0.

uint32\_t **cc\_stack\_size**

Cluster controller stack size (0x800)

uint32\_t **scratch\_size**

Size of the L1 reserved for scratch data.

uint32\_t **icache\_conf**

Reserved for cluster icache configuration: b'0: icache enable b'1: master core icache enable b'2-10: prefetch enable, bit i => core[i]

### Cluster team synchronization

#### group **ClusterTeam**

Once a cluster entry point has been entered by cluster controller core, all the following primitives can be used to do multi-core processing, in a fork-join manner (like OMP parallel primitive).

The execution can first be forked in order to activate more cores, and then synchronized together using barriers.

### Functions

PI\_INLINE\_CL\_TEAM\_0 int **pi\_cl\_cluster\_nb\_cores()**

Return the number of cores of the cluster.

This will return the number of worker cores present in the cluster, available to share/fork a task.

**Returns** Number of cores.

PI\_INLINE\_CL\_TEAM\_0 int **pi\_cl\_team\_nb\_cores()**

Return the number of cores in the team.

This will return the number of cores involved in the team created by the active fork operation.

**Returns** The number of cores of the team.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_fork**(int nb\_cores, void (\*entry)(void\*), void \*arg)

Fork the execution of the calling core.

Calling this function will create a team of workers and call the specified entry point on each core of the team to start multi-core processing. The team parameters (number of cores and stacks) are by default the ones configured when sending a task to cluster from the fabric controller. It is possible to use different parameters when doing a new fork. If this is done the new parameters will become the new default ones.

---

**Note:** If `nb_cores` is zero, fork is done reusing the `cores_mask` of the previous fork or the default.

---



---

**Note:** If the number of cores is not provided (i.e. is zero), the number of cores of the previous fork will be reused. Doing this has less runtime overhead.

---

### Parameters

- **nb\_cores** – Number of cores to execute the task/entry point..
- **entry** – Function entry point to be executed by team of workers.
- **arg** – Argument of the function entry point.

```
PI_INLINE_CL_TEAM_0 void pi_cl_team_prepare_fork(int nb_cores)
Prepare a team for a task.
```

This function is called to set up a team of workers. This function has the same behaviour as [`pi\_cl\_team\_fork\(\)`](#), but here the fork is not done, thus a call to [`pi\_cl\_team\_preset\_fork\(\)`](#) is needed in order to dispatch the task to the team of workers.

### Parameters

- **nb\_cores** – Number of cores to execute the entry point

```
PI_INLINE_CL_TEAM_0 void pi_cl_team_preset_fork(void (*entry)(void*), void *arg)
Fork a the execution of the calling core.
```

This function is to be called after [`pi\_cl\_team\_prepare\_fork\(\)`](#). A call to this function will fork the function entry point between a preset team of workers.

---

**Note:** Calling [`pi\_cl\_team\_prepare\_fork\(\)`](#) then [`pi\_cl\_team\_preset\_fork\(\)`](#) is no different than calling [`pi\_cl\_team\_fork\(\)`](#) once. But when forking multiple times with the same team of workers, calling these two functions prevent from having an overhead due to barrier setups, ie first call [`pi\_cl\_team\_prepare\_fork\(\)`](#) to set the team then multiple calls to [`pi\_cl\_team\_preset\_fork\(\)`](#).

---



---

**Note:** Workers are exclusively slave cores.

---

### Parameters

- **entry** – Function entry point to be executed by team of workers.
- **arg** – Argument of the function entry point.

```
PI_INLINE_CL_TEAM_0 void pi_cl_team_fork_task(struct pi_cl_team_task *fork_task)
Fork the execution of the calling core using task.
```

This function is similar to pi\_cl\_team\_fork but takes a task as parameter, which allows setting more parameters. Calling this function will create a team of workers and call the specified entry point on each core of the team to start multi-core processing with the team parameters specified in the task.

#### Parameters

- **fork\_task** – Task to be forked on slave cores.

PI\_INLINE\_CL\_TEAM\_0 uint32\_t **pi\_cl\_team\_barrier\_nb\_available**(void)

Return number of available barriers.

This function returns the number of barriers available for the user to alloc.

#### Returns NUMBER Number of barriers.

PI\_INLINE\_CL\_TEAM\_0 uint32\_t **pi\_cl\_team\_barrier\_id**(uint32\_t barrier)

Get barrier ID.

This function returns the ID of given barrier address.

---

**Note:** The barrier ID goes from 0 to number of physical barriers minus 1.

---

#### Parameters

- **barrier** – Address of the barrier.

#### Returns BAR\_ID ID of barrier.

PI\_INLINE\_CL\_TEAM\_0 uint32\_t **pi\_cl\_team\_barrier\_alloc**(void)

Allocate a barrier.

This function will allocate a barrier and return its address if available.

---

**Note:** A call to *pi\_cl\_team\_barrier\_set()* is needed in order to set number of cores that will use the barrier and wait on it.

---

**Returns 0** If no barrier available.

**Returns BAR\_ADDR** Address of the barrier allocated.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_barrier\_free**(uint32\_t barrier)

Free a barrier.

This function will free a barrier and make it available for allocation.

#### Parameters

- **barrier** – Address of the barrier to free.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_barrier\_set**(uint32\_t barrier, uint32\_t team\_mask)

Set up a barrier.

This function must be used after allocating a barrier in order to set up the barrier.

#### Parameters

- **barrier** – Address of the barrier to configure.
- **team\_mask** – Mask of the cores using the barrier.

---

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_barrier\_wait**(uint32\_t barrier)

Execute a barrier between all cores of the team.

This will block the execution of each calling core until all cores have reached the barrier. The set of cores participating in the barrier is the one created with the last fork. Each core of the team must execute the barrier exactly once for all cores to be able to go through the barrier.

#### Parameters

- **barrier** – Address of the barrier to wait on.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_barrier()**

Execute a barrier between all cores of the team.

This will block the execution of each calling core until all cores have reached the barrier. The set of cores participating in the barrier is the one created with the last fork. Each core of the team must execute the barrier exactly once for all cores to be able to go through the barrier.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_critical\_enter**(void)

Enter a critical section.

This will block the execution of the calling core until it can execute the following section of code alone. This will also prevent all other cores of the team to execute the following code until *pi\_cl\_team\_critical\_exit()* is called.

---

**Note:** No runtime functions should be called from within the critical section, only application code is allowed.

PI\_INLINE\_CL\_TEAM\_0 void **pi\_cl\_team\_critical\_exit**(void)

Exit a critical section.

This will exit the critical code and let other cores executing it.

## Cluster DMA

### group ClusterDMA

This set of functions provides support for controlling the cluster DMA. The cluster has its own local memory for fast access from the cluster cores while the other memories are relatively slow if accessed by the cluster. To keep all the cores available for computation each cluster contains a DMA unit whose role is to asynchronously transfer data between a remote memory and the cluster memory.

The DMA is using HW counters to track the termination of transfers. Each transfer is by default allocating one counter, which is freed when the wait function is called and returns, which is limiting the maximum number of transfers which can be done at the same time. You can check the chip-specific section to know the number of HW counters.

## TypeDefs

```
typedef struct pi_cl_dma_cmd_s pi_cl_dma_cmd_t
    Structure for DMA commands.
```

This structure is used by the runtime to manage a DMA command. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through the memory allocator.

```
typedef struct pi_cl_dma_copy_s pi_cl_dma_copy_t
    Structure for 1D DMA copy structure.
```

This structure is used by the runtime to manage a 1D DMA copy. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through the memory allocator.

```
typedef pi_cl_dma_copy_t pi_cl_dma_copy_2d_t
    Structure for 2D DMA copy structure.
```

This structure is used by the runtime to manage a 2D DMA copy. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through the memory allocator.

## Enums

```
enum pi_cl_dma_dir_e
    DMA transfer direction.
```

Describes the direction for a DMA transfer.

*Values:*

```
enumerator PI_CL_DMA_DIR_LOC2EXT = 0
    Transfer from cluster memory to external memory.
```

```
enumerator PI_CL_DMA_DIR_EXT2LOC = 1
    Transfer from external memory to cluster memory.
```

## Functions

```
static inline void pi_cl_dma_cmd(uint32_t ext, uint32_t loc, uint32_t size, pi_cl_dma_dir_e dir,
                                pi_cl_dma_cmd_t *cmd)
```

1D DMA memory transfer.

This enqueues a 1D DMA memory transfer (i.e. classic memory copy) with simple completion based on transfer identifier.

### Parameters

- **ext** – Address in the external memory where to access the data.
- **loc** – Address in the cluster memory where to access the data.
- **size** – Number of bytes to be transferred.

- **dir** – Direction of the transfer. If it is PI\_CL\_DMA\_DIR\_EXT2LOC, the transfer is loading data from external memory and storing to cluster memory. If it is PI\_CL\_DMA\_DIR\_LOC2EXT, it is the opposite.
- **cmd** – A pointer to the structure for the copy. This can be used with pi\_cl\_dma\_wait to wait for the completion of this transfer.

```
static inline void pi_cl_dma_cmd_2d(uint32_t ext, uint32_t loc, uint32_t size, uint32_t stride, uint32_t length, pi_cl_dma_dir_e dir, pi_cl_dma_cmd_t *cmd)
```

2D DMA memory transfer.

This enqueues a 2D DMA memory transfer (rectangle) with simple completion based on transfer identifier.

#### Parameters

- **ext** – Address in the external memory where to access the data.
- **loc** – Address in the cluster memory where to access the data.
- **size** – Number of bytes to be transferred.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the DMA will switch to the next line.
- **dir** – Direction of the transfer. If it is PI\_CL\_DMA\_DIR\_EXT2LOC, the transfer is loading data from external memory and storing to cluster memory. If it is PI\_CL\_DMA\_DIR\_LOC2EXT, it is the opposite.
- **cmd** – A pointer to the structure for the copy. This can be used with pi\_cl\_dma\_wait to wait for the completion of this transfer.

```
static inline void pi_cl_dma_cmd_wait(pi_cl_dma_cmd_t *cmd)
```

Simple DMA transfer completion wait.

This blocks the core until the specified transfer is finished. The transfer must be described through the identifier given to the copy function.

#### Parameters

- **cmd** – The copy structure (1d or 2d).

```
static inline void pi_cl_dma_flush()
```

Simple DMA transfer completion flush.

This blocks the core until the DMA does not have any pending transfer.

```
static inline void pi_cl_dma_memcpy(pi_cl_dma_copy_t *copy)
```

1D DMA memory transfer.

This enqueues a 1D DMA memory transfer (i.e. classic memory copy) with simple completion based on transfer identifier.

#### Parameters

- **copy** – A pointer to the structure describing the transfer. The same structure can be used with pi\_cl\_dma\_wait to wait for the completion of this transfer.

```
static inline void pi_cl_dma_memcpy_2d(pi_cl_dma_copy_2d_t *copy)
```

2D DMA memory transfer.

This enqueues a 2D DMA memory transfer (rectangle area) with simple completion based on transfer identifier.

### Parameters

- **copy** – A pointer to the structure describing the transfer. The same structure can be used with pi\_cl\_dma\_wait to wait for the completion of this transfer.

```
static inline void pi_cl_dma_wait(void *copy)
```

Simple DMA transfer completion wait.

This blocks the core until the specified transfer is finished. The transfer must be described through the identifier given to the copy function.

### Parameters

- **copy** – The copy structure (1d or 2d).

## CPI

### group CPI

This API provides support for capturing images from an image sensor through the Camera Parallel Interface (CPI) and processing them.

### Enums

```
enum pi_cpi_format_e
```

Image format identifier.

This can be used to describe the format of the image going through the interface.

*Values:*

```
enumerator PI_CPI_FORMAT_RGB565 = 0  
RGB565 format.
```

```
enumerator PI_CPI_FORMAT_RGB555 = 1  
RGB555 format.
```

```
enumerator PI_CPI_FORMAT_RGB444 = 2  
RGB444 format.
```

```
enumerator PI_CPI_FORMAT_RGB888 = 3  
YUV422 format.
```

```
enumerator PI_CPI_FORMAT_BYPASS_LITEND = 4  
Only least significant byte is kept.
```

```
enumerator PI_CPI_FORMAT_BYPASS_BIGEND = 5  
Only most significant byte is kept.
```

## Functions

**void pi\_cpi\_conf\_init(struct *pi\_cpi\_conf* \*conf)**  
Initialize a CPI configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened.

### Parameters

- **conf** – A pointer to the CPI configuration.

**int pi\_cpi\_open(struct pi\_device \*device)**  
Open a CPI device.

This function must be called before the CPI device can be used. It configures the specified device that can then be used to refer to the opened device when calling other functions.

### Parameters

- **device** – A pointer to the structure describing the device. This structure must be allocated by the caller and kept alive until the device is closed.

**Returns** 0 if it succeeded or -1 if it failed.

**void pi\_cpi\_close(struct pi\_device \*device)**  
Close an opened CPI device.

This function can be called to close an opened CPI device once it is not needed anymore in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – A pointer to the structure describing the device.

**void pi\_cpi\_capture(struct pi\_device \*device, void \*buffer, int32\_t bufferlen)**  
Capture a sequence of samples.

Queue a buffer that will receive samples from the CPI interface. This function is synchronous and will block the caller until the specified amount of bytes is received. The buffer will receive samples only if the interface is started. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – A pointer to the structure describing the device.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **bufferlen** – The size in bytes of the memory buffer.

**void pi\_cpi\_capture\_async(struct pi\_device \*device, void \*buffer, int32\_t bufferlen, pi\_evt\_t \*task)**  
Capture a sequence of samples asynchronously.

Queue a buffer that will receive samples from the CPI interface. This function is asynchronous and will not block the caller. It is possible to call it several times in order to queue several buffers. At a minimum 2 buffers should be queued to ensure that no data sampled is lost. This is also the most efficient way to retrieve data from the CPI device. You should always make sure that at least 2 buffers are always queued, by queuing a new one as soon as the current one is full. The buffer will receive samples only if the interface is started. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – A pointer to the structure describing the device.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **bufferlen** – The size in bytes of the memory buffer.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_task for more details.

```
static inline void pi_cpi_control_start(struct pi_device *device)
    Start capturing samples.
```

The samples arriving at the CPI interface are dropped until the interface is started by calling this function. Once started, the sampling will start at the next beginning of frame.

#### Parameters

- **device** – A pointer to the structure describing the device.

```
static inline void pi_cpi_control_stop(struct pi_device *device)
    Stop capturing samples.
```

The samples arriving at the CPI interface are dropped after this call.

#### Parameters

- **device** – A pointer to the structure describing the device.

```
static inline void pi_cpi_set_format(struct pi_device *device, pi_cpi_format_e format)
    Set frame format.
```

This can be used either when filtering is not active to specify which part of the samples to keep, or when filtering is active, to specify what is the input format, to be able to properly convert the image.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **format** – The format of the input frame.

```
static inline void pi_cpi_set_rowlen(struct pi_device *device, uint16_t rowlen)
    Set frame row length.
```

Set the row length of each frame, should be the same as the camera output. Will be set at the beginning of cpi open.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **rowlen** – The rowlen of each frame, should be divide by the datasize of CPI. For example, if the picture size is 320\*240, cpi transfer datasize is 16bits, the rowlen should be: 320/2 (bytes).

```
static inline void pi_cpi_set_frame_drop(struct pi_device *device, uint32_t nb_frame_dropped)
    Configure frame drop.
```

For each sampled frame, the specified number of frames will be dropped before the next frame is sampled.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **nb\_frame\_dropped** – Number of frames to drop. Can be set to 0 to deactivate this feature.

```
static inline void pi_cpi_set_filter(struct pi_device *device, uint32_t r_coeff, uint32_t g_coeff, uint32_t b_coeff, uint32_t shift)
```

Configure frame filtering. (ONLY available in GAP8)

Configure how to filter the input pixels to produce the 8bits output pixels. Each channel is multiplied by a coefficient. They are then summed together and shifted right to obtain an 8 bit pixel.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **r\_coeff** – Red channel is multiplied by this coefficient.
- **g\_coeff** – Green channel is multiplied by this coefficient.
- **b\_coeff** – Blue channel is multiplied by this coefficient.
- **shift** – The sum of all channels multiplied by their coefficient is shifted right by this value to fit an 8 bit pixel.

```
static inline void pi_cpi_set_rgb_sequence(struct pi_device *device, uint8_t rgb_seq)
```

Configure RGB Sequence.

Configure how the RGB be saved to L2 memory. (ONLY available in GAP9)

#### Parameters

- **device** – A pointer to the structure describing the device.
- **rgb\_seq** – Sequence of RGB: 3'h0 - RGB, 3'h1 - RBG, 3'h2 - GRB, 3'h3 - GBR, 3'h4 - BRG, 3'h5 - BGR,

```
static inline void pi_cpi_set_slice(struct pi_device *device, uint32_t x, uint32_t y, uint32_t w, uint32_t h)
```

Configure frame slicing.

Configure how to extract a window slice out of the input frame.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **x** – x position of the window to extract.
- **y** – y position of the window to extract.
- **w** – Width of the window to extract.
- **h** – Height of the window to extract.

```
static inline void pi_cpi_set_sync_polarity(struct pi_device *device, uint8_t vsync_pol_ena, uint8_t hsync_pol_ena)
```

Configure vsync/hsync polarity.

Configure to reverse the vsync and/or hsync polarity. (ONLY available in GAP9)

#### Parameters

- **device** – A pointer to the structure describing the device.
- **vsync\_pol\_ena** – Enable the vsync polarity reversal.
- **hsync\_pol\_ena** – Enable the hsync polarity reversal.

```
struct pi_cpi_conf
```

#include <cpi.h> CPI device configuration structure.

This structure is used to pass the desired CPI configuration to the runtime when opening the device.

## Public Members

`pi_device_e device`

Device type.

`uint8_t itf`

CPI interface ID where the device is connected.

`uint8_t datasize`

CPI transfer datasize

## CSI2

### group CSI2

This API provides support for capturing images from an image sensor through the Camera Parallel Interface (CSI2) and processing them.

### Enums

enum `pi_csi2_format_e`

Image format identifier.

This can be used to describe the format of the image going through the interface.

*Values:*

enumerator `PI_CSI2_FORMAT_RAW8` = 0

RAW8 format.

enumerator `PI_CSI2_FORMAT_RAW10` = 1

RAW10 format.

### Functions

void `pi_csi2_conf_init`(struct `pi_csi2_conf` \*conf)

Initialize a CSI2 configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened.

#### Parameters

- `conf` – A pointer to the CSI2 configuration.

`int32_t pi_csi2_open(struct pi_device *device)`

Open a CSI2 device.

This function must be called before the CSI2 device can be used. It configures the specified device that can then be used to refer to the opened device when calling other functions.

#### Parameters

- `device` – A pointer to the structure describing the device. This structure must be allocated by the caller and kept alive until the device is closed.

**Returns** 0 if it succeeded or -1 if it failed.

```
void pi_csi2_close(struct pi_device *device)
    Close an opened CSI2 device.
```

This function can be called to close an opened CSI2 device once it is not needed anymore in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

#### Parameters

- **device** – A pointer to the structure describing the device.

```
void pi_csi2_capture(struct pi_device *device, void *buffer, int32_t bufferlen)
    Capture a sequence of samples.
```

Queue a buffer that will receive samples from the CSI2 interface. This function is synchronous and will block the caller until the specified amount of bytes is received. The buffer will receive samples only if the interface is started. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **bufferlen** – The size in bytes of the memory buffer.

```
void pi_csi2_capture_async(struct pi_device *device, void *buffer, int32_t bufferlen, pi_evt_t *task)
    Capture a sequence of samples asynchronously.
```

Queue a buffer that will receive samples from the CSI2 interface. This function is asynchronous and will not block the caller. It is possible to call it several times in order to queue several buffers. At a minimum 2 buffers should be queued to ensure that no data sampled is lost. This is also the most efficient way to retrieve data from the CSI2 device. You should always make sure that at least 2 buffers are always queued, by queuing a new one as soon as the current one is full. The buffer will receive samples only if the interface is started. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **bufferlen** – The size in bytes of the memory buffer.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_task for more details.

```
static inline void pi_csi2_control_start(struct pi_device *device)
    Start capturing samples.
```

The samples arriving at the CSI2 interface are dropped until the interface is started by calling this function. Once started, the sampling will start at the next beginning of frame.

#### Parameters

- **device** – A pointer to the structure describing the device.

```
static inline void pi_csi2_control_stop(struct pi_device *device)
    Stop capturing samples.
```

The samples arriving at the CSI2 interface are dropped after this call.

#### Parameters

- **device** – A pointer to the structure describing the device.

```
static inline void pi_csi2_set_lane(struct pi_device *device, uint8_t num)
```

Set lane number format.

This can be used configure the MIPI CSI2 lane number.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **num** – Number of lanes, 0 - 1lane, 1 - 2lanes.

```
static inline void pi_csi2_set_pixel_clk_div(struct pi_device *device, uint8_t pixel_div)
```

Set pixel clock divider.

This can be used configure the CSI2's receiver's pixel clock divder to match &\* the frequency of MIPI CSI2 lane.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **pixel\_div** – Clock divider [0, 255].

```
static inline void pi_csi2_set_apb_clk_div(struct pi_device *device, uint8_t apb_div)
```

Set APB clock divider.

This can be used configure the CSI2's DPHY's APB clock divder to configure &\* the DPHY receiver.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **apb\_div** – Clock divider [0, 255].

```
static inline void pi_csi2_set_virtual_channel(struct pi_device *device, uint8_t vc)
```

Set Virtual channel ID.

This can be used configure the CSI2's DPHY's virtual channel number, only support VC0 and VC1.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **vc** – VC channel number, [0, 1]. If vc is 2, it means two outstanding streams

```
struct pi_csi2_conf
```

#include <csi2.h> CSI2 device configuration structure.

This structure is used to pass the desired CSI2 configuration to the runtime when opening the device.

## Public Members

**pi\_device\_e device**  
Device type.

**uint8\_t itf**  
CSI2 interface ID where the device is connected.

**uint8\_t datasize**  
CSI2 transfer datasize

## DMACPY

### group DMACPY

Memory copy using UDMA.

This API provides support for UDMA memory copy(DMACPY). The DMACPY allows memory copy between FC L1 memory and L2 memory.

### Enums

**enum pi\_dmacpy\_dir\_e**  
Memcpy direction.

*Values:*

enumerator **PI\_DMACPY\_FC\_L1\_L2** = 0  
Memcpy from FC\_L1 to L2.

enumerator **PI\_DMACPY\_L2\_FC\_L1** = 1  
Memcpy from L2 to FC\_L1.

enumerator **PI\_DMACPY\_L2\_L2** = 2  
Memcpy from L2 to L2.

### Functions

**void pi\_dmacpy\_conf\_init(struct *pi\_dmacpy\_conf* \*conf)**  
Initialize DMA memcpy config.

This function initializes DMA memcpy configuration struct with default values.

#### Parameters

- **conf** – Pointer to DMA Memcpy conf struct.

**int pi\_dmacpy\_open(struct pi\_device \*device)**  
Open a DMA Memcpy device.

This function opens a DMA Memcpy device. Once it is opened, user can do memory copy between FC memory and L2 memory.

---

**Note:** This function must be called before any use of the device.

---

### Parameters

- **device** – Pointer to device structure.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
void pi_dmacpy_close(struct pi_device *device)
```

Close an opened DMA Memcpy device.

This function closes a DMA Memcpy device.

### Parameters

- **device** – Pointer to device structure.

```
int pi_dmacpy_copy(struct pi_device *device, void *src, void *dst, uint32_t size, pi_dmacpy_dir_e dir)
```

Synchronous copy.

A memory copy is done from FC 11 memory or L2 memory and L2 memory, depending on memcpy direction.

---

**Note:** Both src and dst buffers must be aligned on 4 bytes.

---

---

**Note:** The size must be a multiple of 4 bytes.

---

### Parameters

- **device** – Pointer to device structure.
- **src** – Pointer to source buffer.
- **dst** – Pointer to dest buffer.
- **size** – Size of data to copy.
- **dir** – Direction of memcpy.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_dmacpy_copy_async(struct pi_device *device, void *src, void *dst, uint32_t size, pi_dmacpy_dir_e dir, struct pi_evt *task)
```

Asynchronous copy.

A memory copy is done from FC 11 memory or L2 memory and L2 memory, depending on memcpy direction.

---

**Note:** Both src and dst buffers must be aligned on 4 bytes.

---

---

**Note:** The size must be a multiple of 4 bytes.

---

### Parameters

- **device** – Pointer to device structure.
- **src** – Pointer to source buffer.
- **dst** – Pointer to dest buffer.
- **size** – Size of data to copy.
- **dir** – Direction of memcpy.
- **task** – Event task used to notify end of copy.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
struct pi_dmacpy_conf
    #include <dmacpy.h> DMA Memcpy configuration options.
```

### Public Members

```
uint8_t id
    DMA memcpy device ID.
```

## EFUSE

### group EFUSE

The EFUSE driver provides support for programming and reading the fuse.

### TypeDefs

```
typedef struct pi_fuser_reg pi_fuser_reg_t
```

### Enums

#### enum pi\_efuse\_ioctl\_e

Commands of efuse ioctl.

This is used to tell which command to execute through pi\_efuse\_ioctl

*Values:*

```
enumerator PI_EFUSE_IOCTL_PROGRAM_START = 1
```

Start the efuse programming.

enumerator **PI\_EFUSE\_IOCTL\_READ\_START** = 2  
Start the efuse reading.

enumerator **PI\_EFUSE\_IOCTL\_CLOSE** = 3  
Close and stop the efuse for programming/reading.

## Functions

`uint32_t pi_efuse_value_get(uint32_t id)`  
Get the value of an efuse register.

### Parameters

- **id** – The efuse register ID (Please check Datasheet for efuse registers)

**Returns** The efuse register value

`void pi_efuse_program(uint32_t id, uint32_t val)`  
Program the specific efuse register.

### Parameters

- **id** – The efuse register ID (Please check Datasheet for efuse registers)
- **val** – The register value (Please check Datasheet for efuse registers)

`int32_t pi_efuse_ioctl(uint32_t cmd, void *arg)`  
Dynamically control the efuse.

### Parameters

- **cmd** – The efuse ioctl command. The command must be one of those defined in `pi_efuse_ioctl_e`
- **\*arg** – Not used for now, reserved for the future.

`struct pi_fuser_reg`  
*#include <efuse.h>* efuse register structure.

This structure is used to program or read the value of efuse

## Public Members

`uint32_t id`  
Id of the efuse register

`uint32_t val`  
Val of the efuse register

## FFC

### group FFC

The FFC (Fixed point/Floating point Converter) driver provides an interface to use the FFC IP.

### Typedefs

typedef struct *pi\_ffc\_conf* **pi\_ffc\_conf\_t**  
FFC configuration structure.

This structure is used to pass the desired FFC configuration to the runtime when opening a device.

### Enums

enum **pi\_ffc\_float\_type\_e**

*Values:*

enumerator **PI\_FFC\_FLOAT\_FP16** = 0  
enumerator **PI\_FFC\_FLOAT\_BFP16** = 1  
enumerator **PI\_FFC\_FLOAT\_FP32** = 3

enum **pi\_ffc\_fixed\_type\_e**

*Values:*

enumerator **PI\_FFC\_FIXED\_8** = 0  
enumerator **PI\_FFC\_FIXED\_16** = 1  
enumerator **PI\_FFC\_FIXED\_24** = 2  
enumerator **PI\_FFC\_FIXED\_32** = 3

enum **pi\_ffc\_mode\_e**

*Values:*

enumerator **PI\_FFC\_FLOAT\_TO\_FIXED** = 0  
enumerator **PI\_FFC\_FIXED\_TO\_FLOAT** = 1

enum **pi\_ffc\_io\_mode\_e**

*Values:*

enumerator **PI\_FFC\_MEMORY\_IN\_MEMORY\_OUT** = 0  
enumerator **PI\_FFC\_STREAM\_IN\_MEMORY\_OUT** = 1  
enumerator **PI\_FFC\_MEMORY\_IN\_STREAM\_OUT** = 2  
enumerator **PI\_FFC\_STREAM\_IN\_STREAM\_OUT** = 3

enum **pi\_ffc\_ioctl\_e**

Commands for pi\_ffc\_ioctl.

*Values:*

**enumerator PI\_FFC\_IOCTL\_SET\_IO\_MODE**

Set the IO mode.

This command can be used to change the IO mode

**enumerator PI\_FFC\_IOCTL\_CONTINUOUS\_ENABLE**

Set the continuous mode.

This command can be used to change the continuous operating mode

- 1 enable continuous, ie ffc will continuously convert data
- 0 stops continuous mode. If a transfer is in progress it is stopped immediately.

**Warning:** if continuous mode is enabled with io mode PI\_FFC\_STREAM\_IN\_STREAM\_OUT there is no way to know when transfers are done.

## Functions

**void pi\_ffc\_conf\_init(pi\_ffc\_conf\_t \*conf)**

Initialize an FFC configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the FFC device is opened.

### Parameters

- **conf** – A pointer to the FFC configuration.

**int pi\_ffc\_open(pi\_device\_t \*device)**

Open an FFC device.

This function must be called before the FFC device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

**Warning:** The fifo needs at least one UDMA peripheral enabled to be enabled. Else configuration will not be set, and application will get stuck on push/pop commands.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

**void pi\_ffc\_close(pi\_device\_t \*device)**

Close an opened FFC device.

This function can be called to close an opened FFC device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – The device structure of the device to close.

**int32\_t pi\_ffc\_ioctl(pi\_device\_t \*device, uint32\_t cmd, void \*arg)**

FFC IOCTL function.

## Parameters

- **device** – A pi device structure pointing to FFC device
- **cmd** – ioctl number
- **arg** – argument to be passed to ioctl

**Returns** PI\_OK if operation successfull, else error code

```
void pi_ffc_convert(pi_device_t *device, void *src, void *dst, uint16_t size)
    TODO.
```

```
void pi_ffc_convert_async(pi_device_t *device, void *src, void *dst, uint16_t size, pi_evt_t *task)
    TODO.
```

```
struct pi_ffc_conf
    #include <ffc.h> FFC configuration structure.
```

This structure is used to pass the desired FFC configuration to the runtime when opening a device.

## Public Members

**int8\_t itf**  
FFC device number

**pi\_ffc\_fixed\_type\_e fixed\_type**  
FFC fixed point data type (8bits, 16bits, 24bits, 32bits)

**uint32\_t fixed\_scale**  
Scale number of Fixed point: Fixed Point = -1^Sign \* (Q(Size-Precision).Precision << Scale)

**uint32\_t fixed\_precision**  
Precision size fraction: Q(Size-precision).precision

**pi\_ffc\_float\_type\_e float\_type**  
FFC floating point data type (8bits, 16bits, 32bits)

**pi\_ffc\_mode\_e mode**  
FFC mode: float to fixed, or fixed to float

**pi\_ffc\_io\_mode\_e io\_mode**  
FFC IO mode: Mem In/out, stream In/Out

## GPIO

*group* **GPIO**  
GPIO (General Peripheral Input/Output)

The GPIO driver provides support for controlling GPIOs. The available GPIOs are listed in pmsis/chips folder.

**Typedefs**

```
typedef struct pi_gpio_callback_s pi_gpio_callback_t
```

**Enums****enum *pi\_gpio\_flags\_e***

GPIO configuration flags.

Flags to configure gpio : input/output mode, drive strength, pull activation.

*Values:*

```
enumerator PI_GPIO_PULL_DISABLE = (0 << PI_GPIO_PULL_OFFSET)  
Disable pull.
```

```
enumerator PI_GPIO_PULL_ENABLE = (1 << PI_GPIO_PULL_OFFSET)  
Enable pull.
```

```
enumerator PI_GPIO_DRIVE_STRENGTH_LOW = (0 << PI_GPIO_DRIVE_OFFSET)  
Low drive strength.
```

```
enumerator PI_GPIO_DRIVE_STRENGTH_HIGH = (1 << PI_GPIO_DRIVE_OFFSET)  
High drive strength.
```

```
enumerator PI_GPIO_INPUT = (0 << PI_GPIO_MODE_OFFSET)  
GPIO is an input.
```

```
enumerator PI_GPIO_OUTPUT = (1 << PI_GPIO_MODE_OFFSET)  
GPIO is an output.
```

**enum *pi\_gpio\_notif\_e***

Sensitivity of a GPIO for notifications.

This is used to tell which GPIO value modification will trigger a notification(IRQ).

*Values:*

```
enumerator PI_GPIO_NOTIF_FALL = (0 << PI_GPIO_IRQ_TYPE_OFFSET)  
IRQ are sent on a falling edge on the GPIO value.
```

```
enumerator PI_GPIO_NOTIF_RISE = (1 << PI_GPIO_IRQ_TYPE_OFFSET)  
IRQ are sent on a rising edge on the GPIO value.
```

```
enumerator PI_GPIO_NOTIF_EDGE = (2 << PI_GPIO_IRQ_TYPE_OFFSET)  
IRQ are sent on a rising or a falling edge on the GPIO value.
```

```
enumerator PI_GPIO_NOTIF_NONE = (3 << PI_GPIO_IRQ_TYPE_OFFSET)  
No IRQ.
```

## Functions

`void pi_gpio_conf_init(struct pi_gpio_conf *conf)`  
Initialize a GPIO configuration with default values.

The structure containing the configuration must be kept alive until the device is opened.

### Parameters

- **conf** – A pointer to the GPIO configuration.

`int pi_gpio_open(struct pi_device *device)`  
Open a GPIO device.

This function must be called before the GPIO device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. This operation must be done once for each GPIO port, i.e. for every group of 32 GPIOs. The opened device can then be used to drive the 32 GPIOs.

---

**Note:** The device structure is allocated by the caller and must be kept alive until the device is closed.

---

### Parameters

- **device** – A pointer to the device structure of the device to open.

**Returns 0** If the operation is successfull,

**Returns ERRNO** An error code otherwise.

`void pi_gpio_close(struct pi_device *device)`  
Close an opened GPIO device.

This function closes a GPIO device. If memory was allocated, it is freed.

---

**Note:** Closing a GPIO device will disable all GPIOs.

---

### Parameters

- **device** – A pointer to the device structure of the device to open.

`int pi_gpio_pin_configure(struct pi_device *device, pi_gpio_e gpio, pi_gpio_flags_e flags)`  
Configure a GPIO.

This function can be used to configure several aspects of a GPIO, like the direction.

### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **gpio** – GPIO enumerate type or GPIO number within the port (from 0 to 31).
- **flags** – A bitfield of flags specifying how to configure the GPIO.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

`int pi_gpio_pin_toggle(struct pi_device *device, uint32_t pin)`  
GPIO toggle.

This function can be used to toggle a single GPIO.

---

**Note:** The pin parameter can be a GPIO enumerate type `pi_gpio_e`.

---

---

**Note:** This function is to be used on GPIO pins configured as `PI_GPIO_OUTPUT`.

---

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_gpio_pin_write(struct pi_device *device, uint32_t pin, uint32_t value)
```

Set value of a single GPIO.

This function can be used to change the value of a single GPIO.

---

**Note:** The pin parameter can be a GPIO enumerate type `pi_gpio_e`.

---

---

**Note:** This function is to be used on GPIO pins configured as `PI_GPIO_OUTPUT`.

---

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).
- **value** – The value to be set. This can be either 0 or 1.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_gpio_pin_read(struct pi_device *device, uint32_t pin, uint32_t *value)
```

Get value of a single GPIO.

This function can be used to get the value of a single GPIO. It will store the current input value of a GPIO pin into a given buffer.

---

**Note:** The pin parameter can be a GPIO enumerate type `pi_gpio_e`.

---

---

**Note:** This function should be used on GPIO pins configured as `PI_GPIO_INPUT`. Although this function can be used to retrieve current value set on a GPIO pin configured as `PI_GPIO_OUTPUT`.

---

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.

- **pin** – The GPIO number within the port (from 0 to 31).
- **value** – A pointer to the variable where the GPIO value should be returned. The value will be either 0 or 1.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

---

**void pi\_gpio\_pin\_notif\_configure(struct pi\_device \*device, uint32\_t pin, *pi\_gpio\_notif\_e* flags)**  
Configure notifications for a GPIO.

This function can be used to configure how to be notified by GPIO value modifications. By default, notifications will be buffered and can be read and cleared.

---

**Note:** The pin parameter can be a GPIO enumerate type *pi\_gpio\_e*.

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).
- **flags** – The flags to configure how the notification should be triggered.

---

**void pi\_gpio\_pin\_notif\_clear(struct pi\_device \*device, uint32\_t pin)**  
Clear notification for a GPIO.

This function can be used to clear the notification of a GPIO. A GPIO notification is buffered and this function must be called to clear it so that a new one can be seen.

---

**Note:** The pin parameter can be a GPIO enumerate type *pi\_gpio\_e*.

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).

---

**int pi\_gpio\_pin\_notif\_get(struct pi\_device \*device, uint32\_t pin)**  
Get the value of a notification for a GPIO.

This function can be used to get the value of a notification of a GPIO. It returns 1 if at least one notification was received since the last time it was cleared. Reading the notification does not clear it.

---

**Note:** The pin parameter can be a GPIO enumerate type *pi\_gpio\_e*.

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).

**Returns 0** If a notification was received.

**Returns 1** Otherwise.

```
int pi_gpio_pin_task_add(struct pi_device *device, uint32_t pin, pi_evt_t *task, pi_gpio_notif_e flags)
```

Attach an event task callback to a GPIO pin.

This function is used to attach a callback that will be called when a GPIO triggers an IRQ.

This callback is executed by the event kernel.

---

**Note:** The pin parameter can be a GPIO enumerate type *pi\_gpio\_e*.

---

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).
- **task** – Event task executed when an IRQ is triggered.
- **flags** – The flags to configure how the notification should be triggered.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_gpio_pin_task_remove(struct pi_device *device, uint32_t pin)
```

Remove an event task callback attached to a GPIO pin.

This function is used to remove an attached callback to a GPIO pin. If a IRQ is triggered on the given pin, after removal of the event task, no handler will executed.

---

**Note:** The pin parameter can be a GPIO enumerate type *pi\_gpio\_e*.

---

#### Parameters

- **device** – A pointer to the device structure this GPIO belongs to.
- **pin** – The GPIO number within the port (from 0 to 31).

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

```
static inline void pi_gpio_callback_init(pi_gpio_callback_t *cb, uint32_t pin_mask, pi_callback_func_t  
handler, void *arg)
```

Init GPIO callback.

Initialize a GPIO callback with the handler to call and the user args.

#### Parameters

- **cb** – Pointer to callback to initialize.
- **pin\_mask** – Mask of GPIO pins.
- **handler** – Callback function.
- **arg** – Callback function arg.

```
int pi_gpio_callback_add(struct pi_device *device, pi_gpio_callback_t *cb)
```

Attach a callback.

Add an application callback to a GPIO device. Multiple callbacks can be attached to a device, and to a GPIO pin.

---

**Note:** The callback can initialized with `pi_gpio_callback_init()`.

---



---

**Note:** Using callbacks is incompatible with event handling model(using event tasks cf. `pi_gpio_pin_task_add()`).

---



---

**Note:** A callback is executed when a GPIO's corresponding bit is set in the callbk mask.

---

### Parameters

- **device** – Pointer to device struct.
- **cb** – Callback to add to the list.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

int **pi\_gpio\_callback\_remove**(struct pi\_device \*device, *pi\_gpio\_callback\_t* \*cb)

Remove a callback.

Remove an application callback from a GPIO device.

### Parameters

- **device** – Pointer to device struct.
- **cb** – Callback to remove from the list.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

struct **pi\_gpio\_conf**

#include <gpio.h> GPIO configuration structure.

This structure is used to pass the desired GPIO configuration to the runtime when opening the device.

### Public Members

pi\_device\_e **device**

Interface type.

int32\_t **port**

GPIO port. Each port can contain up to 32 GPIOs.

struct **pi\_gpio\_callback\_s**

#include <gpio.h> GPIO callback struct.

This structure is used by IRQ handler to replace regular GPIO IRQ handler.

## Public Members

`uint32_t pin_mask`

Mask of GPIO pins.

`pi_callback_func_t handler`

Callback handler.

`void *args`

Callback user args.

`struct pi_gpio_callback_s *next`

Next callback pointer.

`struct pi_gpio_callback_s *prev`

Previous callback pointer.

## Hyperbus

### group Hyperbus

The Hyperbus driver provides support for transferring data between an external Hyperbus chip and the processor running this driver.

This is a driver for the Hyperbus interface. Higher-level drivers can be built on top of this one to target specific devices such as Hyperflash or Hyperram. Please refer to the PMSIS BSP documentation for such drivers.

### Typedefs

`typedef struct pi_hyper_conf pi_hyper_conf_t`

`typedef struct pi_cl_hyper_req_s pi_cl_hyper_req_t`

Hyperbus cluster request structure.

This structure is used by the runtime to manage a cluster remote copy with the Hyperbus. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

### Enums

`enum pi_hyper_type_e`

Type of the device connected to the hyperbus interface.

This is used to know if the device is a flash or a RAM.

*Values:*

`enumerator PI_HYPER_TYPE_FLASH`

Device is an Hyperflash.

`enumerator PI_HYPER_TYPE_RAM`

Device is an Hyperram.

**enum pi\_hyper\_ioctl\_cmd**

IOCTL command

*Values:***enumerator PI\_HYPER\_IOCTL\_SET\_LATENCY**

Set the device latency.

This command can be used when the interface has been opened to configure the latency suitable for the device.

**Functions****void pi\_hyper\_conf\_init(struct *pi\_hyper\_conf* \*conf)**

Initialize an Hyperbus configuration with default values.

The structure containing the configuration must be kept alive until the device is opened.

**Parameters**

- **conf** – A pointer to the Hyperbus configuration.

**int32\_t pi\_hyper\_open(pi\_device\_t \*device)**

Open an Hyperbus device.

This function must be called before the Hyperbus device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

**Parameters**

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

**void pi\_hyper\_close(pi\_device\_t \*device)**

Close an opened Hyperbus device.

This function can be called to close an opened Hyperbus device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

**Parameters**

- **device** – The device structure of the device to close.

**int pi\_hyper\_ioctl(pi\_device\_t \*device, uint32\_t cmd, void \*arg)**

Dynamically change the device configuration.

This function can be called to change part of the device configuration after it has been opened or to control it.

**Parameters**

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in *pi\_hyper\_ioctl\_e*.
- **arg** – An additional value which is required for some parameters when they are set.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_read(pi_device_t *device, uint32_t hyper_addr, void *addr,  
                                  uint32_t size)
```

Enqueue a read copy to the Hyperbus (from Hyperbus to processor).

The copy will make a transfer between the Hyperbus and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_read_async(pi_device_t *device, uint32_t hyper_addr, void  
                                  *addr, uint32_t size, pi_evt_t *task)
```

Enqueue an asynchronous read copy to the Hyperbus (from Hyperbus to processor).

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_write(pi_device_t *device, uint32_t hyper_addr, void *addr,  
                                  uint32_t size)
```

Enqueue a write copy to the Hyperbus (from processor to Hyperbus).

The copy will make a transfer between the Hyperbus and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_write_async(pi_device_t *device, uint32_t hyper_addr, void  
                                  *addr, uint32_t size, pi_evt_t *task)
```

Enqueue an asynchronous write copy to the Hyperbus (from processor to Hyperbus).

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is

finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_read_2d(pi_device_t *device, uint32_t hyper_addr, void *addr,
                                         uint32_t size, uint32_t stride, uint32_t length)
```

Enqueue a 2D read copy (rectangle area) to the Hyperbus (from Hyperbus to processor).

The copy will make a transfer between the Hyperbus and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_read_2d_async(pi_device_t *device, uint32_t hyper_addr,
                                                 void *addr, uint32_t size, uint32_t stride,
                                                 uint32_t length, pi_evt_t *task)
```

Enqueue an asynchronous 2D read copy (rectangle area) to the Hyperbus (from Hyperbus to processor).

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.

- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_write_2d(pi_device_t *device, uint32_t hyper_addr, void  
*addr, uint32_t size, uint32_t stride, uint32_t  
length)
```

Enqueue a 2D write copy (rectangle area) to the Hyperbus (from processor to Hyperbus).

The copy will make a transfer between the Hyperbus and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_write_2d_async(pi_device_t *device, uint32_t hyper_addr,  
void *addr, uint32_t size, uint32_t stride,  
uint32_t length, pi_evt_t *task)
```

Enqueue an asynchronous 2D write copy (rectangle area) to the Hyperbus (from processor to Hyperbus).

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_copy_async(pi_device_t *device, uint32_t hyper_addr, void  
*addr, uint32_t size, int ext2loc, pi_evt_t *task)
```

Enqueue an asynchronous copy to the Hyperbus.

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. The transfer can be either a read or a write. Depending on the chip, there may be some restrictions on the

memory which can be used. Check the chip-specific documentation for more details. The hyper address specified with this function must be added the offset returned by pi\_hyper\_offset.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus, which must contain the hyper offset.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from HyperBus to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_HYPER_LVL_0 void pi_hyper_copy_2d_async(pi_device_t *device, uint32_t hyper_addr,
                                                void *addr, uint32_t size, uint32_t stride,
                                                uint32_t length, int ext2loc, pi_evt_t *task)
```

Enqueue an asynchronous 2D copy (rectangle area) to the Hyperbus.

The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. The transfer can be either a read or a write. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details. The hyper address specified with this function must be added the offset returned by pi\_hyper\_offset.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus, which must contain the hyper offset.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from HyperBus to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_cl_hyper_read(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size,
                                   pi_cl_hyper_req_t *req)
```

Enqueue a read copy to the Hyperbus from cluster side (from Hyperbus to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for an HyperBus read copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.

- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_hyper_read_2d(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size,
                                      uint32_t stride, uint32_t length, pi_cl_hyper_req_t *req)
```

Enqueue a 2D read copy (rectangle area) to the Hyperbus from cluster side (from Hyperbus to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an HyperBus read copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_hyper_read_wait(pi_cl_hyper_req_t *req)
```

Wait until the specified hyperbus request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_hyper_write(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size,
                                    pi_cl_hyper_req_t *req)
```

Enqueue a write copy to the Hyperbus from cluster side (from Hyperbus to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an HyperBus write copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_hyper_write_2d(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size, uint32_t stride, uint32_t length, pi_cl_hyper_req_t *req)
```

Enqueue a 2D write copy (rectangle area) to the Hyperbus from cluster side (from Hyperbus to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an HyperBus write copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_hyper_write_wait(pi_cl_hyper_req_t *req)
```

Wait until the specified hyperbus request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_hyper_copy(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size, int ext2loc, pi_cl_hyper_req_t *req)
```

Enqueue a copy with the Hyperbus from cluster side.

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an HyperBus copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from HyperBus to the chip or 0 for the contrary.
- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_hyper_copy_2d(pi_device_t *device, uint32_t hyper_addr, void *addr, uint32_t size,
                                      uint32_t stride, uint32_t length, int ext2loc, pi_cl_hyper_req_t
                                      *req)
```

Enqueue a 2D copy (rectangle area) with the Hyperbus from cluster side.

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an HyperBus copy. The copy will make an asynchronous transfer between the Hyperbus and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Hyperbus chip on which to do the copy.
- **hyper\_addr** – The address of the copy in the Hyperbus.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from HyperBus to the chip or 0 for the contrary.
- **req** – A pointer to the HyperBus request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
void pi_hyper_xip_lock(pi_device_t *device)
```

Forbid XIP refills.

This function can be called to prevent the hyperbus from triggering any XIP refill transfer. This can be used to do an operation in a device which would make an XIP refill fail, like an erase operation. Be careful that locking XIP refills can lead to a deadlock if XIP code is executed so only local code must be executed when the XIP refill is locked. This will only apply to the new transfer enqueued after calling this function, not to the pending transfers enqueued before.

#### Parameters

- **device** – The device structure of the device to close.

```
void pi_hyper_xip_unlock(pi_device_t *device)
```

Allow XIP refills.

This function can be called to allow again XIP refills after they have been forbidden.

#### Parameters

- **device** – The device structure of the device to close.

```
PI_INLINE_HYPER_LVL_0 uint32_t pi_hyper_offset(struct pi_device *device)
```

Return hyperbus address offset.

Some functions of the API require that an offset is added to the hyperbus address. This functions can be called to retrieve the offset to be applied. See the documentation of each function to see which ones need this offset.

#### Parameters

- **device** – The device structure of the device.

---

```
struct pi_hyper_conf
#include <hyperbus.h> Hyperbus configuration structure.
```

This structure is used to pass the desired Hyperbus configuration to the runtime when opening the device.

## Public Members

`pi_device_e device`  
Interface type.

`signed char id`  
Hyperbus interface where the device is connected.

`uint32_t cs`  
Chip select where the device is connected.

`pi_hyper_type_e type`  
Type of device connected on the hyperbus interface.

`uint32_t baudrate`  
Baudrate (in bytes/second).

## I2C

### group I2C

The I2C driver provides support for transferring data between an external I2C device and the chip running this driver.

## TypeDefs

`typedef struct pi_i2c_conf pi_i2c_conf_t`

## Enums

`enum pi_i2c_xfer_flags_e`  
Properties for I2C transfers.

This is used to specify additional behaviors when transferring data through I2C.

*Values:*

`enumerator PI_I2C_XFER_STOP = 0 << 0`  
Generate a STOP bit at the end of the transfer.

`enumerator PI_I2C_XFER_NO_STOP = 1 << 0`  
Don't generate a STOP bit at the end of the transfer.

enumerator **PI\_I2C\_XFER\_START** = 0 << 1  
Generate a START bit at the beginning of the transfer.

enumerator **PI\_I2C\_XFER\_NO\_START** = 1 << 1  
Don't generate a START bit at the beginning of the transfer.

enumerator **PI\_I2C\_XFER\_RESTART** = 1 << 2  
Generate a RESTART bit at the beginning of the transfer.

enumerator **PI\_I2C\_XFER\_NO\_RESTART** = 0 << 2  
Don't generate a RESTART bit at the beginning of the transfer.

enum **pi\_i2c\_ioctl\_e**  
Commands for pi\_i2c\_control.

This is used to tell which command to execute through pi\_i2c\_control.

*Values:*

enumerator **PI\_I2C\_CTRL\_SET\_MAX\_BAUDRATE** = 1 <<  
**\_PI\_I2C\_CTRL\_SET\_MAX\_BAUDRATE\_BIT**  
Change maximum baudrate.

**Parameters**

- **baudrate** – Max baudrate.

enumerator **PI\_I2C\_IOCTL\_ABORT\_RX** = 6  
Abort RX transfer.

Abort current RX transfert.

enumerator **PI\_I2C\_IOCTL\_ABORT\_TX** = 7  
Abort TX transfer.

Abort current TX transfert.

enumerator **PI\_I2C\_IOCTL\_ATTACH\_TIMEOUT\_RX** = 8  
Attach UDMA timer.

This command attaches a UDMA timer channel to UDMA reception channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_I2C\_IOCTL\_DETACH\_TIMEOUT\_RX** = 9  
Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA reception channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_I2C\_IOCTL\_ATTACH\_TIMEOUT\_TX** = 10  
Attach UDMA timer.

This command attaches a UDMA timer channel to UDMA transmission channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_I2C\_IOCTL\_DETACH\_TIMEOUT\_TX** = 11  
 Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA transmission channel.

#### Parameters

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_I2C\_IOCTL\_EN\_TIMESTAMP** = 12  
 Enable the timestamp.

Enable the timestamp feature for the i2c interface.

## Functions

**void pi\_i2c\_conf\_init(pi\_i2c\_conf\_t \*conf)**

Initialize an I2C configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the I2C device is opened.

#### Parameters

- **conf** – A pointer to the I2C configuration.

**void pi\_i2c\_conf\_set\_slave\_addr(struct pi\_i2c\_conf \*conf, uint16\_t slave\_addr, int8\_t is\_10\_bits)**  
 set slave\_addr in conf.

#### Parameters

- **conf** – A pointer to the I2C configuration.
- **slave\_addr** – Address of the slave device.
- **is\_10\_bits** – Indicate if slave address is 7 bits or 10 bits.

**void pi\_i2c\_conf\_set\_wait\_cycles(struct pi\_i2c\_conf \*conf, uint16\_t wait\_cycles)**  
 set wait\_cycles in conf.

#### Parameters

- **conf** – A pointer to the I2C configuration.
- **wait\_cycles** – Number of wait cycles applied at the end of transfer

**int32\_t pi\_i2c\_open(struct pi\_device \*device)**  
 Open an I2C device.

This function must be called before the Hyperbus device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

#### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

**void pi\_i2c\_close(struct pi\_device \*device)**  
 Close an opened I2C device.

This function can be called to close an opened I2C device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – The device structure of the device to close.

```
void pi_i2c_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
```

Dynamically change the device configuration.

This function can be called to change part of the device configuration after it has been opened.

### Parameters

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in `pi_i2c_ioctl_e`.
- **arg** – An additional value which is required for some parameters when they are set.

```
int32_t pi_i2c_read(struct pi_device *device, uint8_t *rx_buff, int32_t length, pi_i2c_xfer_flags_e flags)
```

Enqueue a burst read copy from the I2C (from I2C device to chip).

This function can be used to read at least 1 byte of data from the I2C device. The copy will make a synchronous transfer between the I2C and one of the chip memory. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – A pointer to the structure describing the device.
- **rx\_buff** – The address in the chip where the received data must be written.
- **length** – The size in bytes of the copy.
- **flags** – Specify additional transfer behaviors like start and stop bits management.

**Returns** PI\_OK if request is successful PI\_ERR\_I2C\_NACK if a NACK was received during the request

```
int32_t pi_i2c_write(struct pi_device *device, uint8_t *tx_data, int32_t length, pi_i2c_xfer_flags_e flags)
```

Enqueue a burst write copy to the I2C (from chip to I2C device).

This function can be used to write at least 1 byte of data to the I2C device. The copy will make a synchronous transfer between the I2C and one of the chip memory. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent is read.
- **length** – The size in bytes of the copy.
- **flags** – Specify additional transfer behaviors like start and stop bits management.

**Returns** PI\_OK if request is successful PI\_ERR\_I2C\_NACK if a NACK was received during the request

```
void pi_i2c_write_read(struct pi_device *device, void *tx_buffer, void *rx_buffer, uint32_t tx_size,
```

`uint32_t rx_size)`

```
void pi_i2c_write_dual(struct pi_device *device, void *tx_buffer0, void *tx_buffer1, uint32_t tx_size0,
                      uint32_t tx_size1)
```

```
void pi_i2c_read_async(struct pi_device *device, uint8_t *rx_buff, int length, pi_i2c_xfer_flags_e flags,
                       pi_evt_t *task)
```

Enqueue an asynchronous burst read copy from the I2C (from I2C device to chip).

This function can be used to read at least 1 byte of data from the I2C device. The copy will make an asynchronous transfer between the I2C and one of the chip memory. A task must be specified in order to specify how the caller should be notified when the transfer is finished. The task will contain the request status (success or error). It should be retrieved using the pi\_i2c\_get\_request\_status function. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **rx\_buff** – The address in the chip where the received data must be written.
- **length** – The size in bytes of the copy.
- **flags** – Specify additional transfer behaviors like start and stop bits management.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
void pi_i2c_write_async(struct pi_device *device, uint8_t *tx_data, int length, pi_i2c_xfer_flags_e flags,
                        pi_evt_t *task)
```

Enqueue a burst write copy to the I2C (from chip to I2C device).

This function can be used to write at least 1 byte of data to the I2C device. The copy will make an asynchronous transfer between the I2C and one of the chip memory. A task must be specified in order to specify how the caller should be notified when the transfer is finished. The task will contain the request status (success or error). It should be retrieved using the pi\_i2c\_get\_request\_status function. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent is read.
- **length** – The size in bytes of the copy
- **flags** – Specify additional transfer behaviors like start and stop bits management.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
void pi_i2c_write_read_async(struct pi_device *device, void *tx_buffer, void *rx_buffer, uint32_t
                             tx_size, uint32_t rx_size, pi_evt_t *callback)
```

```
void pi_i2c_write_dual_async(struct pi_device *device, void *tx_buffer0, void *tx_buffer1, uint32_t
                             tx_size0, uint32_t tx_size1, pi_evt_t *callback)
```

```
int32_t pi_i2c_read_timeout(struct pi_device *device, uint8_t *rx_buff, int32_t length,
                            pi_i2c_xfer_flags_e flags, uint32_t timeout_us)
```

Enqueue an asynchronous burst read copy from the I2C (from I2C device to chip), with timeout.

This function has the same behaviour as [pi\\_i2c\\_read\(\)](#), with timeout. This timeout value is used to abort/cancel a transfer when timeout is reached.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this functions :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_i2c\\_ioctl\(\) //PI\\_I2C\\_IOCTL\\_ATTACH\\_TIMEOUT\\_RX](#)
- 

**Note:** To use this feature asynchronously, proceed as follows :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_i2c\\_ioctl\(\) //PI\\_I2C\\_IOCTL\\_ATTACH\\_TIMEOUT\\_RX](#)
  - [pi\\_evt\\_timeout\\_set\(\)](#)
  - [pi\\_i2c\\_write\\_async\(\)](#)
- 

#### Parameters

- **device** – Pointer to the structure describing the device.
- **rx\_buff** – Address in the chip where the received data must be written.
- **length** – Size in bytes of the copy.
- **flags** – Specify additional transfer behaviors like start and stop bits management.
- **timeout\_us** – Timeout value in us.

```
int32_t pi_i2c_write_timeout(struct pi_device *device, uint8_t *tx_data, int32_t length,  
                           pi\_i2c\_xfer\_flags\_e flags, uint32_t timeout_us)
```

Enqueue a burst write copy to the I2C (from chip to I2C device), with timeout.

This function has the same behaviour as [pi\\_i2c\\_write\(\)](#), with timeout. This timeout value is used to abort/cancel a transfer when timeout is reached.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this functions :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_i2c\\_ioctl\(\) //PI\\_I2C\\_IOCTL\\_ATTACH\\_TIMEOUT\\_TX](#)
- 

**Note:** To use this feature asynchronously, proceed as follows :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_i2c\\_ioctl\(\) //PI\\_I2C\\_IOCTL\\_ATTACH\\_TIMEOUT\\_TX](#)
  - [pi\\_evt\\_timeout\\_set\(\)](#)
  - [pi\\_i2c\\_write\\_async\(\)](#)
- 

#### Parameters

- **device** – Pointer to the structure describing the device.

- **tx\_data** – Address in the chip where the data to be sent is read.
- **length** – Size in bytes of the copy
- **flags** – Specify additional transfer behaviors like start and stop bits management.
- **timeout\_us** – Timeout value in us.

```
int pi_i2c_get_request_status(pi_evt_t *task)
```

get the request status from a task

This function can be used to retrieve the request status from a task.

**Warning:** in order to be able to retrieve the request status you need to pass a pointer to the task as the first callback argument. You can use next arguments as you please.

#### Parameters

- **task** – the task to retrieve the status from

**Returns** PI\_OK if the request was successful PI\_ERR\_I2C\_NACK if a NACK was received during the request

```
int32_t pi_i2c_detect(struct pi_device *device, struct pi_i2c_conf *conf, uint8_t *rx_data)
```

Scan i2c bus to detect a dev.

This function can be used to detect if a device is connected to the i2c bus. The pi\_i2c\_conf\_t structure is used to pass the address of the device to look for, with different baudrate if needed.

#### Parameters

- **device** – Pointer to device structure.
- **conf** – Conf struct holding address and baudrate.
- **rx\_data** – Pointer to 1 Byte buffer to store data.

**Returns 0x00** If a device has been detected at given address.

**Returns 0xFF** Otherwise.

```
struct pi_i2c_conf
```

#include <i2c.h> I2C master configuration structure.

This structure is used to pass the desired I2C configuration to the runtime when opening a device.

#### Public Members

**uint8\_t itf**

Specifies on which I2C interface the device is connected.

**uint16\_t cs**

i2c slave address (7 or 10 bits), the runtime will take care of the LSB of read and write.

**uint32\_t max\_baudrate**

Maximum baudrate for the I2C bitstream which can be used with the opened device .

```
uint8_t ts_ch
    Enable the timestamp on TX (0) or RX (1)

uint8_t ts_evt_id
    UDMA Config Event ID for generating the timestamp
```

## I2C Slave

### group I2C\_SLAVE

The I2C driver provides support for transferring data between an external I2C device and the chip running this driver.

#### Typedefs

```
typedef struct pi_i2c_slave_args pi_i2c_slave_args_t
typedef void (*pi_i2c_callback_t)(pi_i2c_slave_args_t*)
typedef struct pi_i2c_slave_conf pi_i2c_slave_conf_t
```

I2C master configuration structure.

This structure is used to pass the desired I2C Slave configuration to the runtime when opening a device.

#### Enums

```
enum pi_i2c_slave_ioctl_e
    Values:
```

```
enumerator PI_I2C_SLAVE_CTRL_SET_MAX_BAUDRATE = 1 <<
    __PI_I2C_SLAVE_CTRL_SET_MAX_BAUDRATE_BIT
    Change maximum baudrate.
```

#### Functions

```
void pi_i2c_slave_conf_init(pi_i2c_slave_conf_t *conf)
    Initialize an I2C Slave configuration with default values.
```

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the I2C Slave device is opened.

##### Parameters

- **conf** – A pointer to the I2C Slave configuration.

```
void pi_i2c_slave_conf_set_addr0(struct pi_i2c_slave_conf *conf, uint16_t addr, uint8_t mask, uint8_t
    is_10_bit, uint8_t eof, uint8_t sof)
    set gap9 slave addr0 in conf.
```

##### Parameters

- **conf** – A pointer to the I2C slave configuration.
- **addr** – addr0 for gap slave mode

- **mask** – choose a match mask
- **is\_10\_bit** – if 1, addr is 10 bit

```
void pi_i2c_slave_conf_set_addr1(struct pi_i2c_slave_conf *conf, uint16_t addr, uint8_t mask, uint8_t_t
                                  is_10_bit, uint8_t eof, uint8_t sof)
set gap9 slave addr1 in conf.
```

#### Parameters

- **conf** – A pointer to the I2C slave configuration.
- **addr** – addr1 for gap slave mode
- **mask** – choose a match mask
- **is\_10\_bit** – if 1, addr is 10 bit

```
int32_t pi_i2c_slave_open(struct pi_device *device)
Open an I2C slave device.
```

This function must be called before the I2C slave device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

#### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
void pi_i2c_slave_close(struct pi_device *device)
Close an opened I2C slave device.
```

This function can be called to close an opened I2C slave device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

#### Parameters

- **device** – The device structure of the device to close.

```
void pi_i2c_slave_set_rx(void *handle, void *l2_addr, uint32_t size)
set rx channel registers for I2C slave
```

#### Parameters

- **handle** – A pointer to the low level handle passed by driver in slave irq context.
- **l2\_addr** – L2 address for the rx buffer
- **size** – size of the rx buffer

```
void pi_i2c_slave_set_tx(void *handle, void *l2_addr, uint32_t size)
set tx channel registers for I2C slave
```

#### Parameters

- **handle** – A pointer to the low level handle passed by driver in slave irq context.
- **l2\_addr** – L2 address for the tx buffer
- **size** – size of the tx buffer

```
void pi_i2c_slave_unlock(void *handle, int32_t is_rd)
Flush & unlock I2C slave bus (need to stop transfers before)
```

#### Parameters

- **handle** – A pointer to the low level handle passed by driver in slave irq context.
- **is\_rd** – if 1 flush for rx transfer, else for tx

```
void pi_i2c_slave_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
```

I2C slave IOCTL function.

This function can be called to change part of the device configuration after it has been opened.

#### Parameters

- **device** – A pi device structure pointing to I2C slave device
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in pi\_i2c\_ioctl\_e.
- **arg** – An additional value which is required for some parameters when they are set.

```
void pi_i2c_slave_stop_rx(void *handle)
```

Stop transfer on I2C slave rx channel.

#### Parameters

- **handle** – A pointer to the low level handle passed by driver in slave irq context.

```
void pi_i2c_slave_stop_tx(void *handle)
```

Stop transfer on I2C slave tx channel.

#### Parameters

- **handle** – A pointer to the low level handle passed by driver in slave irq context.

```
struct pi_i2c_slave_args  
struct pi_i2c_slave_conf  
#include <i2c_slave.h> I2C master configuration structure.
```

This structure is used to pass the desired I2C Slave configuration to the runtime when opening a device.

### Public Members

**uint8\_t itf**

Specifies on which I2C interface the device is connected.

**uint8\_t sof0**

Specifies whether addr0 triggers irq on START

**uint8\_t eof0**

Specifies whether addr1 triggers irq on STOP

**uint8\_t sof1**

Specifies whether addr1 triggers irq on START

**uint8\_t eof1**

Specifies whether addr1 triggers irq on STOP

**uint8\_t addr0\_10\_bit**

Specifies whether addr0 is 10 bits

**uint8\_t addr1\_10\_bit**

Specifies whether addr1 is 10 bits

**uint8\_t mask0**

Specifies addr matching mask for addr0

**uint8\_t mask1**

Specifies addr matching mask for addr1

**uint16\_t addr0**

Addr0 to which ift match

**uint16\_t addr1**

Addr1 to which ift match

**uint32\_t max\_baudrate**

Maximum baudrate for the I2C bitstream which can be used with the opened device .

***pi\_i2c\_callback\_t rx\_callback***

callback for rx transfers

***pi\_i2c\_callback\_t tx\_callback***

callback for tx transfers

**I2S****group I2S**

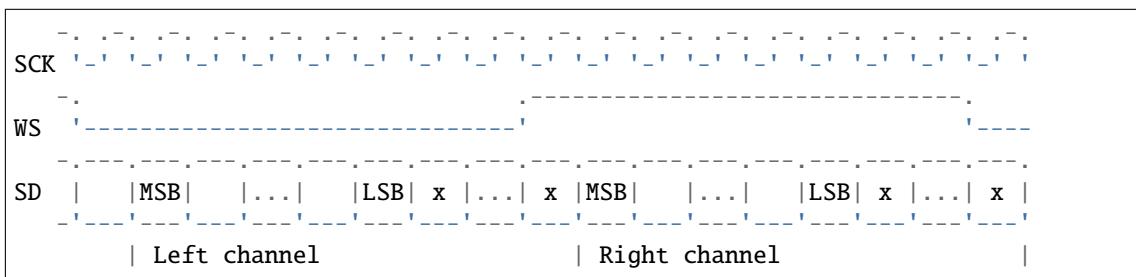
I2S (Inter-IC Sound) Interface.

The I2S API provides support for the I2S interface.

**Defines****PI\_I2S\_FMT\_DATA\_FORMAT\_I2S**

Standard I2S Data Format.

Serial data is transmitted in two's complement with the MSB first. Both Word Select (WS) and Serial Data (SD) signals are sampled on the rising edge of the clock signal (SCK). The MSB is always sent one clock period after the WS changes. Left channel data are sent first indicated by WS = 0, followed by right channel data indicated by WS = 1.



**PI\_I2S\_FMT\_DATA\_FORMAT\_PDM**

Pulse-Density Modulation Format.

Serial data is transmitted using the pulse-density modulation. Each sample is a one bit pulse, where the density of the pulses gives the amplitude of the signal. The driver will filter the input signals so that classic PCM samples are stored in the buffers. In single channel mode, the bits are transmitted on clock signal (CLK) rising edges. In dual channel mode, left channel is transmitted on SCK rising edges and right channel on SCK falling edges. Word Select (WS) is ignored.

**PI\_I2S\_CH\_FMT\_DATA\_ORDER\_MSB**

Data order MSB.

Data bits are transferred MSB first.

**PI\_I2S\_CH\_FMT\_DATA\_ORDER\_LSB**

Data order LSB.

Data bits are transferred LSB first.

**PI\_I2S\_CH\_FMT\_DATA\_ALIGN\_LEFT**

Data alignment left.

Left Justified.

**PI\_I2S\_CH\_FMT\_DATA\_ALIGN\_RIGHT**

Data alignment right.

Right Justified.

**PI\_I2S\_CH\_FMT\_DATA\_SIGN\_NO\_EXTEND**

Data sign extension disabled.

Data are not sign extended when I2S data size is inferior to memory data size.

**PI\_I2S\_CH\_FMT\_DATA\_SIGN\_EXTEND**

Data sign extension enabled.

Data are sign extended when I2S data size is inferior to memory data size. For example when I2S data are on 16 bits but are stored in a 32 bits word, MSB sign is extended.

**PI\_I2S\_OPT\_PINGPONG**

Ping pong mode.

In ping pong mode TX output or RX sampling will keep alternating between a ping buffer and a pong buffer. This is normally used in audio streams when one buffer is being populated while the other is being played (DMAed) and vice versa. So, in this mode, 2 sets of buffers fixed in size are used. These 2 buffers must be given in the configuration when the driver is opened and kept alive until the driver is closed.

**PI\_I2S\_OPT\_MEM\_SLAB**

Mem slab mode.

In mem slab mode TX output or RX sampling will keep alternating between a set of buffers given by the user. Memory slab pointed to by the mem\_slab field has to be defined and initialized by the user. For I2S driver to function correctly number of memory blocks in a slab has to be at least 2 per queue. Size of the memory block should be multiple of frame\_size where frame\_size = (channels \* word\_size\_bytes). As an example 16 bit word will occupy 2 bytes, 24 or 32 bit word will occupy 4 bytes.

**PI\_I2S\_OPT\_FULL\_DUPLEX**

Full duplex mode.

The normal and default mode is to use a single pin for both TX and RX. In full duplex mode, RX and TX will use 2 different pins(called sdi and sdo so that samples can be received and sent at the same time).

**PI\_I2S\_OPT\_DISABLED**

Disable the channel.

If set, this desactivates the channel being configured (either RX or TX). Once the sampling is running, this will make sure this channel is not receiving or sending samples.

**PI\_I2S\_OPT\_ENABLED**

Enable the channel.

If set, this activates the channel being configured (either RX or TX). Once the sampling is running, this will make sure this channel is receiving or sending samples.

**PI\_I2S\_OPT\_IS\_TX**

Configure TX channel.

If set, the configuration will apply to the TX channel and won't change anything for what concerns the RX channel. Note that this does not prevent from using the RX channel, they must just be configured separately.

Setting this configuration allows to send data from memory to speakers.

**PI\_I2S\_OPT\_IS\_RX**

Configure RX channel.

If set, the configuration will apply to the RX channel and won't change anything for what concerns the TX channel. Note that this does not prevent from using the TX channel, they must just be configured separately.

Setting this configuration allows to receive data from microphones to memory.

**PI\_I2S\_OPT\_LOOPBACK**

TX loopback.

If set, this activates an internal loopback between the RX pin and TX pin. All data received on the RX will be sent to the TX pin. Note that this does not prevent from receiving data. Data received on the RX can be sent both to a memory buffer (if RX channel is enabled) and to the TX (if the loopback is enabled). The loopback must be applied on the TX channel.

**PI\_I2S\_OPT\_TDM**

TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels where each channel can have a specific configuration. As for classic I2S mode, TX and RX channels are grouped together within a slot. Each slot can send and receive 1 data per frame. In this mode each slot must be configured separately using the I2S configuration. Each slot can have an RX and a TX channel.

**PI\_I2S\_OPT\_INT\_CLK**

Use internal clock.

Clock is generated internally from SOC's clock.

This is default clock source.

**PI\_I2S\_OPT\_EXT\_CLK**

Use external clock.

If this option is specified, no clock is generated and an external clock is used.

**PI\_I2S\_OPT\_INT\_WS**

Use internal word strobe.

WS signal is generated internally from SCK signal.

This is default WS source.

**PI\_I2S\_OPT\_EXT\_WS**

Use external word strobe.

If this option is specified, no word strobe is generated and an external one is used.

**PI\_I2S\_OPT\_EXT\_CLK\_INT\_ROUTED**

Use internally routed ext clk.

If this option is specified, the clk of SAI1 or SAI2 is routed internally from SAI0

**PI\_I2S\_OPT\_EXT\_WS\_INT\_ROUTED**

Use internally routed ext ws.

If this option is specified, the ws of SAI1 or SAI2 is routed internally from SAI0

**PI\_I2S\_OPT\_CLK\_POLARITY\_RISING\_EDGE**

Data sampling on rising edge of the clock.

If this option is specified, the data are sampled and generated on the rising edge of the clock

**PI\_I2S\_OPT\_CLK\_POLARITY\_FALLING\_EDGE**

Data sampling on falling edge of the clock.

If this option is specified, the data are sampled and generated on the falling edge of the clock

**PI\_I2S\_OPT\_WS\_POLARITY\_RISING\_EDGE**

WS sampling on rising edge of the clock.

If this option is specified, the WS is sampled and generated on the rising edge of the clock.

**PI\_I2S\_OPT\_WS\_POLARITY\_FALLING\_EDGE**

Data sampling on falling edge of the clock.

If this option is specified, the WS is sampled and generated on the falling edge of the clock

**PI\_I2S\_OPT\_REF\_CLK\_FAST**

Enable the ref clk fast.

If this option is specified, the clk will be based on the ref clk fast, but not the FLL.

## TypeDefs

```
typedef uint8_t pi_i2s_fmt_t
typedef uint16_t pi_i2s_opt_t
```

## Enums

enum **pi\_i2s\_ioctl\_cmd\_e**  
    IOCTL command.

*Values:*

enumerator **PI\_I2S\_IOCTL\_START**  
    Start the transmission / reception of data.

This command can be used when the interface has been opened or stopped to start sampling.

enumerator **PI\_I2S\_IOCTL\_STOP**  
    Stop the transmission / reception of data.

Stop the transmission / reception of data at the end of the current memory block. This command can be used when the interface is sampling and is stopping the interface. When the current TX / RX block is transmitted / received the interface is stopped. Subsequent START command will resume transmission / reception where it stopped.

enumerator **PI\_I2S\_IOCTL\_SYNC**  
    Synchronize transmission / reception of data between different interfaces.

This command can be used when the sampling over different I2S interfaces has to be synchronized. The argument is an integer which specifies the list of interfaces which must be synchronized (one bit per interface). Once called, all further PI\_I2S\_IOCTL\_START and PI\_I2S\_IOCTL\_STOP as well as calls to `pi_i2s_channel_enable` on other interfaces are put on hold until a PI\_I2S\_IOCTL\_START or PI\_I2S\_IOCTL\_STOP command is issued on the interface where PI\_I2S\_IOCTL\_SYNC was called, which commits the commands on all involved interfaces at the same time.

enumerator **PI\_I2S\_IOCTL\_CONF\_SET**  
    Configure a channel in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels, and each channel can have a specific configuration. This command can be used to give the configuration of one channel. The argument must be a pointer to a structure of type struct `pi_i2s_conf` containing the channel configuration.

enumerator **PI\_I2S\_IOCTL\_CONF\_GET**  
    Get the current configuration of a channel in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels, and each channel can have a specific configuration. This command can be used to get the current configuration of one channel. The argument must be a pointer to a structure of type struct `pi_i2s_conf` where the current channel configuration will be stored.

enumerator **PI\_I2S\_IOCTL\_CLOCK\_ENABLE**  
    Enable clock.

Enable clock for the i2s interface. This command does not sample data, and does not save data. It allows the interface to receive clock signals from SoC and propagate it to slave devices on end line.

enumerator **PI\_I2S\_IOCTL\_CLOCK\_DISABLE**

Disable clock.

Disable clock for the i2s interface.

enumerator **PI\_I2S\_IOCTL\_EN\_TIMESTAMP**

Enable the timestamp.

Enable the timestamp feature for the i2s interface.

## Functions

**void pi\_i2s\_setup(uint32\_t flags)**

Setup specific I2S aspects.

This function can be called to set specific I2S properties such as the number of clock generator. This is typically used by the BSP to give board specific information.

### Parameters

- **flags** – A bitfield of chip-dependant properties.

**void pi\_i2s\_conf\_init(struct *pi\_i2s\_conf* \*conf)**

Initialize an I2S configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the I2S device is opened.

### Parameters

- **conf** – A pointer to the I2S configuration.

**int pi\_i2s\_open(struct pi\_device \*device)**

Open an I2S device.

This function must be called before the I2S device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the caller and must be kept alive until the device is closed.

**Returns** 0 if the operation is successful, -1 if there was an error.

**void pi\_i2s\_close(struct pi\_device \*device)**

Close an opened I2S device.

This function can be called to close an opened I2S device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the structure describing the device.

---

```
int pi_i2s_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
```

Dynamically change the device configuration.

This function can be called to change part of the device configuration after it has been opened or to control it.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in pi\_spi\_ioctl\_e.
- **arg** – An additional value which is required for some parameters when they are set.

```
int pi_i2s_read(struct pi_device *dev, void **mem_block, size_t *size)
```

Read data from the RX queue.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

When using several channels, the organization of the samples for each channel in the buffer, is chip-dependent, check the chip-specific documentation to get more information.

If there is no data in the RX queue the function will block waiting for the next RX memory block to fill in.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **mem\_block** – Pointer to the variable storing the address of the RX memory block containing received data.
- **size** – Pointer to the variable storing the number of bytes read.

**Returns 0** If successful, -1 if not.

```
int pi_i2s_read_async(struct pi_device *dev, pi_evt_t *task)
```

Read data asynchronously from the RX queue.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

When using several channels, the organization of the samples for each channel in the buffer, is chip-dependent, check the chip-specific documentation to get more information.

The specified task will be pushed as soon as data is ready in the RX queue, and the information about the memory block and the size will be available in the task.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful, -1 if not.

```
int pi_i2s_write(struct pi_device *dev, void *mem_block, size_t size)
    Write data to the TX queue of a channel.
```

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

This fonction will block until the specified buffer has been transfered.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.

**Returns 0** If successful.

**Returns -1** An error occured.

```
int pi_i2s_write_async(struct pi_device *dev, void *mem_block, size_t size, pi_evt_t *task)
    Write data asynchronously to the TX queue of a channel.
```

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

The specified task will be pushed as soon as data is has been transfered.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful.

**Returns -1** An error occured.

```
void pi_i2s_channel_conf_init(struct pi_i2s_channel_conf *conf)
    Initialize an I2S channel configuration with default values.
```

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the I2S channel is configured.

#### Parameters

- **conf** – A pointer to the I2S channel configuration.

---

```
PI_INLINE_I2S_LVL_0 int pi_i2s_channel_conf_set(struct pi_device *dev, int channel, struct
                                               pi_i2s_channel_conf *conf)
```

Configure a channel in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels, and each channel can have a specific configuration. This function can be used to give the configuration of one channel.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **conf** – A pointer to the I2S channel configuration.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
PI_INLINE_I2S_LVL_0 int pi_i2s_frame_channel_conf_set(struct pi_device *dev, uint32_t frame, int
                                                       channel, struct pi_i2s_channel_conf
                                                       *conf)
```

Configure a channel of a frame in TDM mode.

A frame is a set of channels gathered together so that they can be controlled all together in terms of data transfer. This function can be used to configure a channel which is part of a frame. In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels, and each channel can have a specific configuration. This function can be used to give the configuration of one channel.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **frame** – A bitfield containing the channels of the part (one bit per channel).
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **conf** – A pointer to the I2S channel configuration.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
int pi_i2s_channel_conf_get(struct pi_device *dev, int channel, struct pi_i2s_channel_conf *conf)
```

Get the current configuration of a channel in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple channels, and each channel can have a specific configuration. This function can be used to get the current configuration of one channel

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **conf** – A pointer to the I2S channel configuration.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
int pi_i2s_channel_timestamp_set(struct pi_device *dev, int channel, struct pi_i2s_channel_conf *conf)
```

Enable the timestamp.

Enable the timestamp feature for the i2s interface.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the channel, from 0 to the number of channels minus 1.
- **conf** – A pointer to the I2S channel configuration.

int **pi\_i2s\_slots\_enable**(struct pi\_device \*dev, uint32\_t slots, int enabled)

Enable or disable slots in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple slots of time. Each slot can send and receive data at the same time in full-duplex mode, and is thus made of one 1 RX channels and 1 TX channel. This function can be called to enable or disable the 2 channels of a slot, for example for muting the channels to reconfigure them. The slot is by default enabled when the channel configuration is initialized. Enabling slots will allow them to received or send data, while disabling them will allow it. These operations are not immediate and can take up to 2 frame periods to be effective. When disabling a slot, the buffers are kept, and data continues from the same buffer position after the slot is reenabled.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **slots** – Bitfield of the slots to enable or disable, with one bit per slot. Each slot whose bit is set to 1 will be configured.
- **enabled** – 1 if the specified slots should be enabled, 0 if they should be disabled.

**Returns 0** If successful.

**Returns -1** An error occurred.

int **pi\_i2s\_slots\_stop\_async**(struct pi\_device \*dev, uint32\_t slots, pi\_evt\_t \*task)

Stop slots in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple slots of time. Each slot can send and receive data at the same time in full-duplex mode, and is thus made of one 1 RX channels and 1 TX channel. This function can be called to enable or disable the 2 channels of a slot, for example for muting the channels to reconfigure them. Stopping a slot will first disable it, and will also cancel all pending buffers. If some samples were already present in the current buffer, the buffer is considered finished and its associated task is triggered. The size of the data already sent or received is notified in the buffer task. Since stopping requires waiting for 2 frame periods, this operation can take some time and is asynchronous.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **slots** – Bitfield of the slots to stop, with one bit per slot. Each slot whose bit is set to 1 will be stopped.

**Returns 0** If successful.

**Returns -1** An error occurred.

int **pi\_i2s\_slots\_stop**(struct pi\_device \*dev, uint32\_t slots)

Stop slots in TDM mode.

In TDM mode, the same interface is time-multiplexed to transmit data for multiple slots of time. Each slot can send and receive data at the same time in full-duplex mode, and is thus made of one 1 RX channels and 1 TX channel. This function can be called to enable or disable the 2 channels of a slot, for example for muting the channels to reconfigure them. Stopping a slot will first disable it, and will also cancel all pending buffers. If some samples were already present in the current buffer, the buffer is considered finished and its associated task is triggered. The size of the data already sent or received is notified in the task. Since stopping requires waiting for 2 frame periods, this operation can take some time and is blocking.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **slots** – Bitfield of the slots to stop, with one bit per slot. Each slot whose bit is set to 1 will be stopped.

**Returns 0** If successful.

**Returns -1** An error occurred.

int **pi\_i2s\_channel\_read**(struct pi\_device \*dev, int channel, void \*\*mem\_block, size\_t \*size)

Read data from the RX queue of a channel in TDM mode.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

This will return data for the specified channel and must only be used in TDM mode.

If there is no data in the RX queue the function will block waiting for the next RX memory block to fill in.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the variable storing the address of the RX memory block containing received data.
- **size** – Pointer to the variable storing the number of bytes read.

**Returns 0** If successful, -1 if not.

int **pi\_i2s\_channel\_read\_async**(struct pi\_device \*dev, int channel, pi\_evt\_t \*task)

Read data asynchronously from the RX queue of a channel in TDM mode.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

This will return data for the specified channel and must only be used in TDM mode.

The specified task will be pushed as soon as data is ready in the RX queue, and the information about the memory block and the size will be available in the task.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful, -1 if not.

PI\_INLINE\_I2S\_LVL\_0 int **pi\_i2s\_frame\_read**(struct pi\_device \*dev, uint32\_t frame, void \*\*mem\_block, size\_t \*size)

Read data from the RX queue of a frame channel in TDM mode.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

This will return data for the specified channel and must only be used in TDM mode.

If there is no data in the RX queue the function will block waiting for the next RX memory block to fill in.

This function is reading for the whole frame. The returned buffer contains the samples for the whole frame, but the size is the size of one channel.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **frame** – A bitfield containing the channels of the part (one bit per channel).
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the variable storing the address of the RX memory block containing received data.
- **size** – Pointer to the variable storing the number of bytes read.

**Returns 0** If successful, -1 if not.

```
PI_INLINE_I2S_LVL_0 int pi_i2s_frame_read_async(struct pi_device *dev, uint32_t frame, pi_evt_t *task)
```

Read data asynchronously from the RX queue of a frame channel in TDM mode.

Data received by the I2S interface is stored in the RX queue consisting of two memory blocks preallocated by the user and given to the driver in the configuration. Calling this function will return the next available buffer to the caller, which has to use it before the sampling for this buffer starts again.

The data is read in chunks equal to the size of the memory block.

This will return data for the specified channel and must only be used in TDM mode.

The specified task will be pushed as soon as data is ready in the RX queue, and the information about the memory block and the size will be available in the task.

This function is reading for the whole frame. The returned buffer contains the samples for the whole frame, but the size is the size of one channel.

Due to hardware constraints, the address of the buffer must be aligned on 4 bytes.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **frame** – A bitfield containing the channels of the part (one bit per channel).
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful, -1 if not.

```
int pi_i2s_read_status(pi_evt_t *task, void **mem_block, size_t *size)
```

Read the status of an asynchronous read.

After pi\_i2s\_read\_async or pi\_i2s\_channel\_read\_async is called to be notified when a read buffer is available, and the notification is received, the output information can be retrieved by calling this function.

### Parameters

- **task** – The task used for notification.
- **mem\_block** – Pointer to the variable storing the address of the RX memory block containing received data.
- **size** – Pointer to the variable storing the number of bytes read.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
int pi_i2s_channel_write(struct pi_device *dev, int channel, void *mem_block, size_t size)
```

Write data to the TX queue of a channel in TDM mode.

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

This fonction will block until the specified buffer has been transfered.

This will sent data to the specified channel and must only be used in TDM mode.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
int pi_i2s_channel_write_async(struct pi_device *dev, int channel, void *mem_block, size_t size,
```

 $\quad \quad \quad$ 

```
pi_evt_t *task)
```

Write data asynchronously to the TX queue of a channel in TDM mode.

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

This will sent data to the specified channel and must only be used in TDM mode.

The specified task will be pushed as soon as data is has been transfered.

### Parameters

- **dev** – Pointer to the device structure for the driver instance.

- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
PI_INLINE_I2S_LVL_0 int pi_i2s_frame_write(struct pi_device *dev, uint32_t frame, void *mem_block,  
                                         size_t size)
```

Write data to the TX queue of a frame channel in TDM mode.

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

This fonction will block until the specified buffer has been transferred.

This function is writing for the whole frame. The buffer must contain the samples for the whole frame, but the size is the size of one channel.

This will sent data to the specified channel and must only be used in TDM mode.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **frame** – A bitfield containing the channels of the part (one bit per channel).
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.

**Returns 0** If successful.

**Returns -1** An error occurred.

```
int pi_i2s_frame_write_async(struct pi_device *dev, uint32_t frame, void *mem_block, size_t size,  
                           pi_evt_t *task)
```

Write data asynchronously to the TX queue of a frame channel in TDM mode.

Data to be sent by the I2S interface is stored first in the TX queue consisting of memory blocks preallocated by the user with either pingpong buffers or a memory slab allocator.

In pingpong mode, the driver will automatically alternate between 2 buffers and the user code is supposed to call this function to notify the driver that the specified buffer is ready to be sent. This is used by the driver to report when an underrun or an overrun occurs.

In memory slab allocator mode, the user has to allocate buffers from the memory slab allocator and pass them to the driver by calling this function when they are ready to be sent.

This will sent data to the specified channel and must only be used in TDM mode.

This function is writing for the whole frame. The buffer must contain the samples for the whole frame, but the size is the size of one channel.

The specified task will be pushed as soon as data is has been transferred.

#### Parameters

- **dev** – Pointer to the device structure for the driver instance.
- **frame** – A bitfield containing the channels of the part (one bit per channel).
- **channel** – ID of the slot, from 0 to the number of channels minus 1.
- **mem\_block** – Pointer to the TX memory block containing data to be sent.
- **size** – Number of bytes to write. This value has to be equal or smaller than the size of the memory block.
- **task** – The task used to notify the end of transfer.

**Returns 0** If successful.

**Returns -1** An error occured.

int **pi\_i2s\_write\_status**(pi\_evt\_t \*task)

Read the status of an asynchronous write.

After pi\_i2s\_write\_async or pi\_i2s\_channel\_write\_async and the notification for the end of transfer is received, return value can be retrieved by calling this function.

#### Parameters

- **task** – The task used for notification.

**Returns 0** If successful.

**Returns -1** An error occured.

```
struct pi_i2s_conf
#include <i2s.h> Interface configuration options.
```

#### Public Members

uint32\_t **frame\_clk\_freq**

Frame clock (WS) frequency, this is sampling rate.

size\_t **block\_size**

Size of one RX/TX memory block(buffer) in bytes. On some chips, this size may have to be set under a maximum size, check the chip-specific section.

pi\_mem\_slab\_t \***mem\_slab**

memory slab to store RX/TX data.

void \***pingpong\_buffers**[2]

Pair of buffers used in double-bufferin mode to capture the incoming samples.

*pi\_i2s\_fmt\_t* **format**

Data stream format as defined by PI\_I2S\_FMT\_\* constants.

**uint8\_t word\_size**

Number of bits representing one data word.

**int8\_t mem\_word\_size**

Number of bits representing one data word in memory. If it is -1, this is equal to word\_size.

**uint8\_t ws\_delay**

Sets the distance (in bits) in i2s cycles from the WS rising edge to the first bit of the frame

**uint8\_t channels**

Number of words per frame.

**uint8\_t itf**

I2S device ID.

***pi\_i2s\_opt\_t options***

Configuration options as defined by PI\_I2S\_OPT\_\* constants.

**uint8\_t pdm\_polarity**

Only 2b' LSB are used, for choosing the mode of the pin SDI and the pin SDO:

- SDI - b0: 0-TX, 1-RX
- SDO - b1: 0-TX, 1-RX In GAP9, if both are configured as 0, the ouput data will be on the SDO

**uint8\_t pdm\_diff**

In PDM Output mode only: set differential mode on pairs,

- bit0: (SDI,WS)
- bit1: (SDO,SCK)

**uint32\_t ref\_clk\_freq**

Configure the ref clk fast value.

**uint8\_t ts\_evt\_id**

UDMA Config Event ID for generating the timestamp

**uint8\_t ws\_type**

Specifies the form of the WS: 0=pulse, 1=i2s stereo mode

**struct pi\_i2s\_channel\_conf**

#include <i2s.h> Interface channel configuration options. This configuration has to be used when configuring a channel in TDM mode. This can also be used to configure channels when they are part of a frame. Be careful in that case that some fields are not specific restrictions in this case.

## Public Members

### **size\_t block\_size**

Size of one RX/TX memory block(buffer) in bytes. On some chips, this size may have to be set under a maximum size, check the chip-specific section. In frame-based mode, this field should be the same for all channels.

### **pi\_mem\_slab\_t \*mem\_slab**

memory slab to store RX/TX data. In frame-based mode, this field should be the same for all channels.

### **void \*pingpong\_buffers[2]**

Pair of buffers used in double-bufferin mode to capture the incoming samples. In frame-based mode, this field should be the same for all channels.

### **pi\_i2s\_fmt\_t format**

Data stream format as defined by PI\_I2S\_FMT\_\* constants.

### **uint8\_t word\_size**

Number of bits representing one data word.

### **int8\_t mem\_word\_size**

Number of bits representing one data word in memory. If it is -1, this is equal to word\_size. In frame-based mode, this field should be the same for all channels.

### **pi\_i2s\_opt\_t options**

Configuration options as defined by PI\_I2S\_OPT\_\* constants.

### **int8\_t stream\_id**

If different from -1, this redirect the specified stream(can be input or output) to/from the SFU/FFC/AES block with the channel specified here. In frame-based mode, this field should be -1. In stream-based mode, this ID should be between 0 and 21 according to the block:

- 0-15: SFU
- 16-17: AES
- 18-21: FFC

### **uint8\_t ts\_evt\_id**

UDMA Config Event ID for generating the timestamp. In frame-based mode, this field should be the same for all channels.

### **uint8\_t slot\_enable**

Specifies if the corresponding slot must be enabled or not. It is by default set to 1.

## I3C

### group I3C

The I3C driver provides support for transferring data between an external I3C device and the chip running this driver.

#### Typedefs

```
typedef struct pi_i3c_conf pi_i3c_conf_t
```

#### Enums

##### enum **pi\_i3c\_ioctl\_e**

Commands for pi\_i3c\_control.

This is used to tell which command to execute through pi\_i3c\_control.

*Values:*

```
enumerator PI_I3C_CTRL_SET_MAX_BAUDRATE = 1 <<  
__PI_I3C_CTRL_SET_MAX_BAUDRATE_BIT  
Change maximum baudrate.
```

##### Parameters

- **baudrate** – Max baudrate.

#### Functions

```
void pi_i3c_conf_init(pi_i3c_conf_t *conf)
```

Initialize an I3C configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the I3C device is opened.

##### Parameters

- **conf** – A pointer to the I3C configuration.

```
void pi_i3c_conf_set_slave_addr(struct pi_i3c_conf *conf, uint16_t slave_addr)
```

Set slave address in conf.

##### Parameters

- **conf** – A pointer to the I3C configuration.
- **slave\_addr** – Address of the slave device.

```
int32_t pi_i3c_open(struct pi_device *device)
```

Open an I3C device.

This function must be called before the Hyperbus device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

##### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns 0** if the operation is successfull.

**Returns -1** if there was an error.

**Returns** Operation return code.

void **pi\_i3c\_close**(struct pi\_device \*device)

Close an opened I3C device.

This function can be called to close an opened I3C device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

#### Parameters

- **device** – The device structure of the device to close.

void **pi\_i3c\_ioctl**(struct pi\_device \*device, uint32\_t cmd, void \*arg)

Dynamically change the device configuration.

This function can be called to change part of the device configuration after it has been opened.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in `pi_i3c_ioctl_e`.
- **arg** – An additional value which is required for some parameters when they are set.

int32\_t **pi\_i3c\_read**(struct pi\_device \*device, uint8\_t \*rx\_buff, int32\_t length)

Enqueue a burst read copy from the I3C (from I3C device to chip).

This function can be used to read at least 1 byte of data from the I3C device. The copy will make a synchronous transfer between the I3C and one of the chip memory. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **rx\_buff** – The address in the chip where the received data must be written.
- **length** – The size in bytes of the buffer to read.

**Returns** PI\_OK if request is successful PI\_ERR\_I3C\_NACK if a NACK was received during the request

int32\_t **pi\_i3c\_write**(struct pi\_device \*device, uint8\_t \*tx\_data, int32\_t length)

Enqueue a burst write copy to the I3C (from chip to I3C device).

This function can be used to write at least 1 byte of data to the I3C device. The copy will make a synchronous transfer between the I3C and one of the chip memory. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent is read.
- **length** – The size in bytes of the buffer to write.

**Returns** PI\_OK if request is successful PI\_ERR\_I3C\_NACK if a NACK was received during the request

```
void pi_i3c_read_async(struct pi_device *device, uint8_t *rx_buff, int32_t length, pi_evt_t *event)  
Enqueue an asynchronous burst read copy from the I3C (from I3C device to chip).
```

This function can be used to read at least 1 byte of data from the I3C device. The copy will make an asynchronous transfer between the I3C and one of the chip memory. An event must be specified in order to specify how the caller should be notified when the transfer is finished. The event will contain the request status (success or error). It should be retrieved using the pi\_i3c\_get\_request\_status function. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **rx\_buff** – The address in the chip where the received data must be written.
- **length** – The size in bytes of the copy.
- **event** – The event used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
void pi_i3c_write_async(struct pi_device *device, uint8_t *tx_data, int32_t length, pi_evt_t *event)  
Enqueue a burst write copy to the I3C (from chip to I3C device).
```

This function can be used to write at least 1 byte of data to the I3C device. The copy will make an asynchronous transfer between the I3C and one of the chip memory. An event must be specified in order to specify how the caller should be notified when the transfer is finished. The event will contain the request status (success or error). It should be retrieved using the pi\_i3c\_get\_request\_status function. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent is read.
- **length** – The size in bytes of the copy
- **event** – The event used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
int32_t pi_i3c_get_request_status(pi_evt_t *event)  
Give information on the transfer status of the given event.
```

**Returns** Code for success if transfer went well, or error if an error occurred.

```
struct pi_i3c_conf  
#include <i3c.h> I3C master configuration structure.
```

This structure is used to pass the desired I3C configuration to the runtime when opening a device.

## Public Members

### `uint8_t ift`

Specifies on which I3C interface the device is connected. For now, we have only one interface (0).

### `uint16_t cs`

i3c slave address (7 bits), the runtime will take care of the LSB of read and write.

### `uint32_t max_baudrate`

Maximum baudrate for the I3C bitstream which can be used with the opened device.

## OctoSPI

### `group Octospi`

The Octospi driver provides support for transferring data between an external Octospi chip and the processor running this driver.

This is a driver for the Octospi interface. Higher-level drivers can be built on top of this one to target specific devices such as Octospiflash or Octospiram. Please refer to the PMSIS BSP documentation for such drivers.

## TypeDefs

`typedef struct pi_octospi_op_conf pi_octospi_op_conf_t`

`typedef struct pi_octospi_conf pi_octospi_conf_t`

`typedef struct pi_cl_octospi_req_s pi_cl_octospi_req_t`

Octospi cluster request structure.

This structure is used by the runtime to manage a cluster remote copy with the Octospi. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

## Enums

### `enum pi_octospi_type_e`

Type of the device connected to the octospi interface.

This is used to know if the device is a flash or a RAM.

*Values:*

enumerator `PI_OCTOSPI_TYPE_FLASH`

Device is an Octospiflash.

enumerator `PI_OCTOSPI_TYPE_RAM`

Device is an Octospiram.

### `enum pi_octospi_flags_e`

SPI operation flags.

This is used to describe some SPI operation flags like command size, address size and so on.

*Values:*

enumerator **PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_0** = (0 << PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_OFFSET)  
No command.

enumerator **PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_1** = (1 << PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_OFFSET)  
Command is 1 byte.

enumerator **PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_2** = (2 << PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_OFFSET)  
Command is 2 bytes.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_0** = (0 << PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_OFFSET)  
No address.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_1** = (1 << PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_OFFSET)  
Address is 1 byte.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_2** = (2 << PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_OFFSET)  
Address is 2 bytes.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_3** = (3 << PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_OFFSET)  
Address is 3 bytes.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_4** = (4 << PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_OFFSET)  
Address is 4 bytes.

enumerator **PI\_OCTOSPI\_FLAG\_LINE\_SINGLE** = (2 << PI\_OCTOSPI\_FLAG\_LINE\_OFFSET)  
Use 1 SPI line.

enumerator **PI\_OCTOSPI\_FLAG\_LINE\_QUAD** = (1 << PI\_OCTOSPI\_FLAG\_LINE\_OFFSET)  
Use 4 SPI lines.

enumerator **PI\_OCTOSPI\_FLAG\_LINE\_OCTO** = (0 << PI\_OCTOSPI\_FLAG\_LINE\_OFFSET)  
Use 8 SPI lines.

enumerator **PI\_OCTOSPI\_FLAG\_CMD\_DTR** = (0 << PI\_OCTOSPI\_FLAG\_CMD\_RATE\_OFFSET)  
Use DTR mode for command.

enumerator **PI\_OCTOSPI\_FLAG\_CMD\_STR** = (1 << PI\_OCTOSPI\_FLAG\_CMD\_RATE\_OFFSET)  
Use STR mode for command.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_DTR** = (0 << PI\_OCTOSPI\_FLAG\_ADDR\_RATE\_OFFSET)  
Use DTR mode for address.

enumerator **PI\_OCTOSPI\_FLAG\_ADDR\_STR** = (1 << PI\_OCTOSPI\_FLAG\_ADDR\_RATE\_OFFSET)  
Use STR mode for address.

enumerator **PI\_OCTOSPI\_FLAG\_DATA\_DTR** = (0 << PI\_OCTOSPI\_FLAG\_DATA\_RATE\_OFFSET)  
Use DTR mode for data.

---

enumerator **PI\_OCTOSPI\_FLAG\_DATA\_STR** = (1 << PI\_OCTOSPI\_FLAG\_DATA\_RATE\_OFFSET)  
Use STR mode for data.

enumerator **PI\_OCTOSPI\_FLAG\_DATA\_DTR\_LSB** = (0 <<  
PI\_OCTOSPI\_FLAG\_DATA\_RATE\_MSB\_OFFSET)  
Use LSB in DTR mode for data.

enumerator **PI\_OCTOSPI\_FLAG\_DATA\_DTR\_MSB** = (1 <<  
PI\_OCTOSPI\_FLAG\_DATA\_RATE\_MSB\_OFFSET)  
Use MSB in DTR mode for data.

enum **pi\_octospi\_cmd\_e**  
SPI operation command flags.

This is used to describe some SPI operation flags for the command.

*Values:*

enumerator **PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_EN** = (1 <<  
PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_EN\_OFFSET)  
Automatically generate additional R/W bit at MSB on top of the specified SPI command. When using automatic R/W bit generation, this will generate 1 if the operation is a read.

enumerator **PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_SET\_READ** = (1 <<  
PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_READ\_OFFSET)  
Always send even addresses to the device.

enumerator **PI\_OCTOSPI\_CMD\_ADDR EVEN** = (1 << PI\_OCTOSPI\_CMD\_ADDR\_OFFSET)

enum **pi\_octospi\_ioctl\_cmd**  
IOCTL command

*Values:*

enumerator **PI\_OCTOSPI\_IOCTL\_SET\_OP**  
Configure the SPI operation.

This command can be used when the interface has been opened to configure the SPI operation used for the next transfer (latency, SPI command, etc). The argument must be a pointer to a variable of type `pi_octospi_op_conf_t`.

enumerator **PI\_OCTOSPI\_IOCTL\_SET\_XIP\_OP**  
Configure the SPI operation for XIP.

This command can be used when the interface has been opened to configure the SPI operation used for all XIP transfers (latency, SPI command, etc). The argument must be a pointer to a variable of type `pi_octospi_op_conf_t`. NOTE: For flash, only set read command, as XIP flash is RO

## Functions

void **pi\_octospi\_conf\_init**(struct *pi\_octospi\_conf* \*conf)

Initialize an Octospi configuration with default values.

The structure containing the configuration must be kept alive until the device is opened.

### Parameters

- **conf** – A pointer to the Octospi configuration.

int32\_t **pi\_octospi\_open**(struct *pi\_device* \*device)

Open an Octospi device.

This function must be called before the Octospi device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

void **pi\_octospi\_close**(struct *pi\_device* \*device)

Close an opened Octospi device.

This function can be called to close an opened Octospi device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – The device structure of the device to close.

int **pi\_octospi\_ioctl**(struct *pi\_device* \*device, uint32\_t cmd, void \*arg)

Dynamically change the device configuration.

This function can be called to change part of the device configuration after it has been opened or to control it.

### Parameters

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in *pi\_spi\_ioctl\_e*.
- **arg** – An additional value which is required for some parameters when they are set.

PI\_INLINE\_OCTOSPI\_LVL\_0 void **pi\_octospi\_read**(struct *pi\_device* \*device, uint32\_t octospi\_addr, void \*addr, uint32\_t size, *pi\_octospi\_op\_conf\_t* \*op)

Enqueue a read copy to the Octospi (from Octospi to processor).

The copy will make a transfer between the Octospi and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.

- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_read_async(struct pi_device *device, uint32_t
                                                octospi_addr, void *addr, uint32_t size,
                                                pi_octospi_op_conf_t *op, pi_evt_t *evt)
```

Enqueue an asynchronous read copy to the Octospi (from Octospi to processor).

The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. An event must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **evt** – The event used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_write(struct pi_device *device, uint32_t octospi_addr,
                                              void *addr, uint32_t size, pi_octospi_op_conf_t
                                              *op)
```

Enqueue a write copy to the Octospi (from processor to Octospi).

The copy will make a transfer between the Octospi and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_write_async(struct pi_device *device, uint32_t
                                                 octospi_addr, void *addr, uint32_t size,
                                                 pi_octospi_op_conf_t *op, pi_evt_t *evt)
```

Enqueue an asynchronous write copy to the Octospi (from processor to Octospi).

The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. An event must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.

- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **evt** – The event used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_read_2d(struct pi_device *device, uint32_t octospi_addr,  
                                              void *addr, uint32_t size, uint32_t stride,  
                                              uint32_t length, pi_octospi_op_conf_t *op)
```

Enqueue a 2D read copy (rectangle area) to the Octospi (from Octospi to processor).

The copy will make a transfer between the Octospi and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_read_2d_async(struct pi_device *device, uint32_t  
                                                 octospi_addr, void *addr, uint32_t size,  
                                                 uint32_t stride, uint32_t length,  
                                                 pi_octospi_op_conf_t *op, pi_evt_t  
                                                 *evt)
```

Enqueue an asynchronous 2D read copy (rectangle area) to the Octospi (from Octospi to processor).

The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. An event must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.

- **evt** – The event used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_write_2d(struct pi_device *device, uint32_t
                                                octospi_addr, void *addr, uint32_t size,
                                                uint32_t stride, uint32_t length,
                                                pi_octospi_op_conf_t *op)
```

Enqueue a 2D write copy (rectangle area) to the Octospi (from processor to Octospi).

The copy will make a transfer between the Octospi and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.

```
PI_INLINE_OCTOSPI_LVL_0 void pi_octospi_write_2d_async(struct pi_device *device, uint32_t
                                                       octospi_addr, void *addr, uint32_t size,
                                                       uint32_t stride, uint32_t length,
                                                       pi_octospi_op_conf_t *op, pi_evt_t
                                                       *evt)
```

Enqueue an asynchronous 2D write copy (rectangle area) to the Octospi (from processor to Octospi).

The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. An event must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **evt** – The event used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_cl_octospi_read(struct pi_device *device, uint32_t octospi_addr, void *addr, uint32_t size, pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a read copy to the Octospi from cluster side (from Octospi to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for an Octospi read copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_octospi_read_2d(struct pi_device *device, uint32_t octospi_addr, void *addr, uint32_t size, uint32_t stride, uint32_t length, pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a 2D read copy (rectangle area) to the Octospi from cluster side (from Octospi to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an Octospi read copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_octospi_read_wait(pi_cl_octospi_req_t *req)
```

Wait until the specified octospi request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_octospi_write(struct pi_device *device, uint32_t octospi_addr, void *addr,
                                      uint32_t size, pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a write copy to the Octospi from cluster side (from Octospi to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an Octospi write copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_octospi_write_2d(struct pi_device *device, uint32_t octospi_addr, void *addr,
                                         uint32_t size, uint32_t stride, uint32_t length,
                                         pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a 2D write copy (rectangle area) to the Octospi from cluster side (from Octospi to processor).

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an Octospi write copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_octospi_write_wait(pi_cl_octospi_req_t *req)
```

Wait until the specified octospi request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_octospi_copy(struct pi_device *device, uint32_t octospi_addr, void *addr, uint32_t size, int ext2loc, pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a copy with the Octospi from cluster side.

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an Octospi copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from Octospi to the chip or 0 for the contrary.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_octospi_copy_2d(struct pi_device *device, uint32_t octospi_addr, void *addr, uint32_t size, uint32_t stride, uint32_t length, int ext2loc, pi_octospi_op_conf_t *op, pi_cl_octospi_req_t *req)
```

Enqueue a 2D copy (rectangle area) with the Octospi from cluster side.

This function is a remote call that the cluster can issue to the fabric-controller in order to ask for an Octospi copy. The copy will make an asynchronous transfer between the Octospi and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the Octospi chip on which to do the copy.
- **octospi\_addr** – The address of the copy in the Octospi.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from Octospi to the chip or 0 for the contrary.
- **op** – The SPI operation configuration. Can be NULL to keep the latest one which was set.
- **req** – A pointer to the Octospi request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
void pi_octospi_xip_lock(struct pi_device *device)
```

Forbid XIP refills.

This function can be called to prevent the octospi from triggering any XIP refill transfer. This can be used to do an operation in a device which would make an XIP refill fail, like an erase operation. Be careful that locking XIP refills can lead to a deadlock if XIP code is executed so only local code must be execyted when the XIP refill is locked. This will only apply to the new transfer enqueued after calling this function, not to the pending transfers enqueued before.

#### Parameters

- **device** – The device structure of the device to close.

```
void pi_octospi_xip_unlock(struct pi_device *device)
    Allow XIP refills.
```

This function can be called to allow again XIP refills after they have been forbidden.

#### Parameters

- **device** – The device structure of the device to close.

```
struct pi_octospi_op_conf
    #include <octospi.h> SPI operation configuration structure.
```

This structure is used to specify the desired SPI operation configuration. It can be passed either with a transfer or through IOCTL to set it permanently.

### Public Members

```
uint32_t cmd
    SPI command.
```

```
uint32_t latency
    SPI dummy cycles.
```

```
uint32_t flags
    SPI flags (command size, etc).
```

```
struct pi_octospi_conf
    #include <octospi.h> Octospi configuration structure.
```

This structure is used to pass the desired Octospi configuration to the runtime when opening the device.

### Public Members

```
pi_device_e device
    Interface type.
```

```
signed char id
    Octospi interface where the device is connected.
```

```
uint8_t xip_en
    Specify whether xip is on
```

```
uint32_t cs
    Chip select where the device is connected.
```

***pi\_octospi\_type\_e*** **type**

Type of device connected on the octospi interface.

**uint32\_t** **baudrate**

Baudrate (in bytes/second).

**uint32\_t** **mba**

Mapping base address for a given CS. Allows IP to choose CS based on input address.

**Padframe****group Padframe**

Padframe.

The padframe driver provides support for controlling PADs.

**Enums****enum pi\_pad\_func\_e**

Pad functions.

This is used to identify the function for each pad.

*Values:*

**enumerator PI\_PAD\_FUNC0 = 0**

Alternate func0, default func.

**enumerator PI\_PAD\_FUNC1 = 1**

Alternate func1. pads as GPIOs.

**enumerator PI\_PAD\_FUNC2 = 2**

Alternate func2,

**enumerator PI\_PAD\_FUNC3 = 3**

Alternate func3.

**enum pi\_pad\_flags\_e**

Pad configuration flags.

Flags are used to configure pad : drive strength, pull activation.

*Values:*

**enumerator PI\_PAD\_PULL\_DISABLE = (0 << PI\_PAD\_PULL\_OFFSET)**

Disable pull.

**enumerator PI\_PAD\_PULL\_ENABLE = (1 << PI\_PAD\_PULL\_OFFSET)**

Enable pull.

enumerator **PI\_PAD\_DS\_LOW** = (0 << PI\_PAD\_DRIVE\_OFFSET)  
 Low drive strength.

enumerator **PI\_PAD\_DS\_HIGH** = (1 << PI\_PAD\_DRIVE\_OFFSET)  
 High drive strength.

enum **pi\_pad\_sleepcfg\_flags\_e**  
 Pad sleep configuration flags.

Flags are used to configure pad : direction, active state.

*Values:*

enumerator **PI\_PAD\_SLEEPCFG\_INPUT** = (0 << PI\_PAD\_SLEEPCFG\_DIR\_OFFSET)  
 Pad is an input.

enumerator **PI\_PAD\_SLEEPCFG\_OUTPUT** = (1 << PI\_PAD\_SLEEPCFG\_DIR\_OFFSET)  
 Pad is an output.

enumerator **PI\_PAD\_SLEEPCFG\_ACTIVE\_LOW** = (0 << PI\_PAD\_SLEEPCFG\_STATE\_OFFSET)  
 Pad is active low.

enumerator **PI\_PAD\_SLEEPCFG\_ACTIVE\_HIGH** = (1 << PI\_PAD\_SLEEPCFG\_STATE\_OFFSET)  
 Pad is active high.

## Functions

void **pi\_pad\_set\_function**(pi\_pad\_e pad, *pi\_pad\_func\_e* function)  
 Set the function of one pad.

This function can be used to configure the function of the specified pad in the case that it supports several functions.

### Parameters

- **pad** – Pad number. See the chip specific configuration for more details.
- **function** – Pad function. See the chip specific configuration for more details.

*pi\_pad\_func\_e* **pi\_pad\_get\_function**(pi\_pad\_e pad)  
 Get the function of a pad.

### Parameters

- **pad** – Pad number. See the chip specific configuration for more details.

**Returns** *pi\_pad\_func\_e* Pad function. See the chip specific configuration for more details.

void **pi\_pad\_init**(uint32\_t pad\_values[])  
 Set the function of all pads.

This function can be used to configure the function of all the pads in the case that they support several functions.

### Parameters

- **pad\_values** – Pad values. This is an array of 32 bit values, with one bit per pad and one 32 bit value per group of 32 pads.

```
void pi_pad_set_configuration(pi_pad_e pad, pi_pad_flags_e cfg)
```

Set the configuration for a pin.

This function configures the pull activation and drive strength of a given pin.

#### Parameters

- **pad** – Pad to configure.
- **cfg** – Pad configuration, set of flags.

```
void pi_pad_sleepcfg_set(uint32_t sleepcfg[], uint8_t sleep_ena)
```

Set the sleep configuration for a set of pins.

This function configures pads behaviour when microcontroller goes into sleep or deep sleep mode. The array sent to this function should be filled for available pads(cf pi\_pad\_e), with 2 bits per pad(active state and direction).

Ex : Pad 10 - PI\_PAD\_18\_A43\_CAM\_PCLK

```
uint32_t array[3] = {0};  
array[PI_PAD_18_A43_CAM_PCLK / 16] |= ((PI_PAD_SLEEPCFG_INPUT |  
PI_PAD_SLEEPCFG_ACTIVE_HIGH) << PI_PAD_18_A43_CAM_PCLK);
```

#### Parameters

- **sleepcfg** – Pads sleep configuration.
- **sleep\_ena** – Enable/disable pad sleep mode.

## Performance counters

### group Perf

This API gives access to the core performance counters. Each core has a few performance counters which can be configured to count one event out of several available. An event is a cycle, an instruction, a cache miss and so on. The number of counters limits the number of events which can be monitored at the same time and depends on the platform. Real chips have only 1 counter while other platforms have one per event.

In addition, this API uses a few other HW mechanisms useful for monitoring performance such as timers.

## Functions

```
static inline void pi_perf_conf(uint32_t events)
```

Configure performance counters.

The set of events which can be activated at the same time depends on the platform. On real chips (rather than with the simulator), there is always only one counter. It is advisable to always use only one to be compatible with simulator and chip. At least PI\_PERF\_CYCLES and another event can be monitored at the same time as the first one is using the timer. This API can be called both from fabric-controller or cluster side.

#### Parameters

- **events** – A mask containing the events to activate. Bit positions in the mas are defined by pi\_perf\_event\_e. For example, to activate “cycles” event counter, add (1 << PI\_PERF\_CYCLES) to the mask.

---

```
static inline void pi_perf_reset()
    Reset all hardware performance counters.

    All hardware performance counters are set to 0, except the time, as it is a shared resource within the cluster.
    This API can be called both from fabric-controller or cluster side.

static inline void pi_perf_start()
    Start monitoring configured events.

    This function is useful for finely controlling the point where performance events start being monitored. The
    counter retains its value between stop and start so it is possible to easily sum events for several portions of
    code. This API can be called both from fabric-controller or cluster side.

static inline void pi_perf_stop()
    Stop monitoring configured events.

    This API can be called both from fabric-controller or cluster side.

static inline uint32_t pi_perf_read(pi_perf_event_e id)
    Read a performance counter.

    This does a direct read of the specified performance counter. Calling this function is useful for getting
    the performance counter with very low overhead (just few instructions). This API can be called both from
    fabric-controller or cluster side.
```

#### Parameters

- **id** – The performance event identifier to read (see `pi_perf_event_e`).

**Returns** The performance counter value.

## PMU

### GAP9

group **GAP9\_PMU**

#### Typedefs

```
typedef enum _pi_pmu_domain_state pi_pmu_domain_state_e
typedef enum pi_pmu_domain_state_flags pi_pmu_domain_state_flags_e
```

#### Enums

```
enum pi_pmu_domain_e
    PMU domains.
```

This is used to identify power domains on GAP9.

*Values:*

```
enumerator PI_PMU_DOMAIN_CL = 3
    Cluster power domain.
```

enumerator **PI\_PMU\_DOMAIN\_CSI2** = 2  
CSI power domain.

enumerator **PI\_PMU\_DOMAIN\_MRAM** = 1  
MRAM power domain.

enumerator **PI\_PMU\_DOMAIN\_SFU** = 0  
SFU power domain.

enumerator **PI\_PMU\_DOMAIN\_CHIP** = 4  
Chip power domain.

enum **\_pi\_pmu\_domain\_state**  
*Values:*

enumerator **PI\_PMU\_DOMAIN\_STATE\_OFF** = 0  
Domain is off.

enumerator **PI\_PMU\_DOMAIN\_STATE\_ON** = 1  
Domain is on.

enumerator **PI\_PMU\_DOMAIN\_STATE\_DEEP\_SLEEP\_RETENTIVE** = 2  
Domain is in sleep mode with retention.

enumerator **PI\_PMU\_DOMAIN\_STATE\_DEEP\_SLEEP** = 3  
Domain is in sleep mode without retention.

enum **pi\_pmu\_domain\_state\_flags**  
*Values:*

enumerator **PI\_PMU\_DOMAIN\_STATE\_NO\_FLAGS**  
No additional flag.

enum **pi\_pmu\_wakeup\_e**  
Wakeup sources.

This is used to specify which the possible sources of wakeup.

*Values:*

enumerator **PI\_PMU\_WAKEUP\_RTC** = (1 << 0)  
RTC wakeup

enumerator **PI\_PMU\_WAKEUP\_GPIO** = (1 << 1)  
GPIO wakeup

enumerator **PI\_PMU\_WAKEUP\_SPISLAVE** = (1 << 2)  
SPI slave wakeup

enum **pi\_pmu\_wakeup\_sequence\_e**  
Wakeup sequences.

This is used to specify in which state the chip is put after wakeup.

*Values:*

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_ON** = ((0 << 0) | (0 << 1) | (0 << 2))  
SOC is ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_MRAM\_ON** = ((1 << 0) | (0 << 1) | (0 << 2))  
SOC and MRAM are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_CL\_ON** = ((0 << 0) | (1 << 1) | (0 << 2))  
SOC and cluster are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_CL\_MRAM\_ON** = ((1 << 0) | (1 << 1) | (0 << 2))  
SOC, cluster and MRAM are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_SFU\_ON** = ((0 << 0) | (0 << 1) | (1 << 2))  
SOC and SFU are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_SFU\_MRAM\_ON** = ((1 << 0) | (0 << 1) | (1 << 2))  
SOC, SFU and MRAM are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_SFU\_CL\_ON** = ((0 << 0) | (1 << 1) | (1 << 2))  
SOC, SFU and cluster are ON.

enumerator **PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_SFU\_CL\_MRAM\_ON** = ((1 << 0) | (1 << 1) | (1 << 2))  
SOC, SFU, cluster and MRAM are ON.

enum **pi\_pmu\_voltage\_domain\_e**  
Voltage domains.

This is used to specify voltage domains.

*Values:*

enumerator **PI\_PMU\_VOLTAGE\_DOMAIN\_CHIP** = 0  
Chip voltage domain.

## Functions

static inline *pi\_pmu\_domain\_state\_e* **pi\_pmu\_boot\_state\_get()**  
Get the state before booting.

This function is used to get the state of the chip before it booted.

**Returns** *pi\_pmu\_domain\_state\_e* Boot state.

int **pi\_pmu\_wakeup\_control**(*pi\_pmu\_wakeup\_e* wakeup, uint32\_t gpio\_mask)  
Wakeup mode.

This function is used to specify how the chip should wakeup from deep sleep.

### Parameters

- **wakeup** – Possible wakeup causes which can be ORed together.
- **gpio\_mask** – In case GPIO wakeup is enabled, specifies the sets of gpios which can wakeup the chip.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
static inline pi_pmu_wakeup_e pi_pmu_wakeup_reason()
```

Get reason that woke up the chip.

This function is used to know which sources woke up the chip from deep sleep.

**Returns** Reasons that woke up the chip(*pi\_pmu\_wakeup\_mode\_e*).

```
static inline uint32_t pi_pmu_gpio_wakeup_pins()
```

Get GPIO wakeup pins.

This function returns the GPIO pin numbers that woke up the chip, in case chip has been woken up by a GPIO.

**Returns** *GPIO\_ID* GPIO pin number.

```
int pi_pmu_wakeup_sequence(pi_pmu_wakeup_sequence_e sequence)
```

Set wakeup sequence.

This function can be used to specify the state in which the chip should be put after wakeup from deep sleep.

#### Parameters

- **sequence** – The state after wakeup.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
int pi_pmu_voltage_set(pi_pmu_voltage_domain_e domain, uint32_t voltage)
```

Set voltage.

This function allows to set voltage of a given power domain.

---

**Note:** When changing frequency, voltage may need to be changed first.

---

#### Parameters

- **domain** – Voltage domain to configure.
- **voltage** – Voltage to set, in mV.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
uint32_t pi_pmu_voltage_get(pi_pmu_voltage_domain_e domain)
```

```
int pi_pmu_wakeup_fll_settings(int settings)
```

Configure FLL settings after wakeup.

This function tells the ROM how to configure the FLL after wakeup from non-retentive deep sleep. The ROM can take the fll settings from 2 different sets of efuse. This function will tell if they are taken from the first or the second set. This can be used to configure the fl at a lower frequency in case the voltage is lower, until the runtime can set the appropriate ABB settings.

#### Parameters

- **settings** – 0 if the ROM should use the first set of efuses, or 1 if it should use the second.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
void pi_pmu_domain_state_change(pi_pmu_domain_e domain, pi_pmu_domain_state_e state,
                                pi_pmu_domain_state_flags_e flags)
```

Change power domain state.

This function is used to change the state of a power domain(cf *pi\_pmu\_domain\_e*).

#### Parameters

- **pd** – Power domain.
- **state** – State.
- **flags** – Additional flags.

```
void pi_pmu_domain_state_change_async(pi_pmu_domain_e domain, pi_pmu_domain_state_e state,
                                       pi_pmu_domain_state_flags_e flags, pi_evt_t *task)
```

Change power domain state.

This function is used to change the state of a power domain(cf *pi\_pmu\_domain\_e*). Same as *pi\_pmu\_power\_domain\_change()*, with this function, user has the possibilty to do other things while waiting for the end of power domain change.

#### Parameters

- **pd** – Power domain.
- **state** – State.
- **flags** – Additional flags.
- **task** – Event task used to notify when action is completed.

## RTC

group RTC

### Enums

```
enum pi_rtc_mode_e
    RTC working mode.
```

*Values:*

```
enumerator PI_RTC_MODE_CALENDAR = 0x1
    RTC calendar mode.
```

```
enumerator PI_RTC_MODE_ALARM = 0x2
    RTC alarm mode.
```

```
enumerator PI_RTC_MODE_TIMER = 0x4
    RTC countdown mode.
```

```
enumerator PI_RTC_MODE_CALIBRATION = 0x8
    RTC calibration mode.
```

**enum pi\_rtc\_alarm\_repeat\_e**

RTC alarm repeat mode.

*Values:*

enumerator **PI\_RTC\_ALARM\_RPT\_NONE** = 0x0

Alarm not repeated.

enumerator **PI\_RTC\_ALARM\_RPT\_SEC** = 0x3

Alarm repeated every seconds.

enumerator **PI\_RTC\_ALARM\_RPT\_MIN** = 0x4

Alarm repeated every minutes.

enumerator **PI\_RTC\_ALARM\_RPT\_HOUR** = 0x5

Alarm repeated every hours.

enumerator **PI\_RTC\_ALARM\_RPT\_DAY** = 0x6

Alarm repeated every days.

enumerator **PI\_RTC\_ALARM\_RPT\_MON** = 0x7

Alarm repeated every months.

enumerator **PI\_RTC\_ALARM\_RPT\_YEAR** = 0x8

Alarm repeated every years.

**enum pi\_rtc\_ioctl\_cmd\_e**

RTC ioctl commands.

*Values:*

enumerator **PI\_RTC\_CALENDAR\_START** = 0

Start RTC device.

enumerator **PI\_RTC\_CALENDAR\_STOP** = 1

Stop RTC device.

enumerator **PI\_RTC\_ALARM\_START** = 2

Start alarm function.

enumerator **PI\_RTC\_ALARM\_STOP** = 3

Stop alarm function.

enumerator **PI\_RTC\_TIMER\_START** = 4

Start countdown.

enumerator **PI\_RTC\_TIMER\_STOP** = 5

Stop countdown.

enumerator **PI\_RTC\_ALARM\_ATTACH\_TASK** = 6

Attach task to be enqueued when alarm is reached.

---

enumerator **PI\_RTC\_TIMER\_ATTACH\_TASK** = 7  
Attach task to be enqueued when timer is reached.

## Functions

**void pi\_rtc\_conf\_init(struct *pi\_rtc\_conf* \*conf)**  
Initialize a RTC configuration structure.

This function initializes a RTC configuration structure with default values.

### Parameters

- **conf** – RTC configuration structure.

**int pi\_rtc\_open(struct pi\_device \*device)**  
Open a RTC device.

This function opens a RTC device.

### Parameters

- **device** – Device structure.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

**void pi\_rtc\_close(struct pi\_device \*device)**  
Close a RTC device.

This function closes a RTC device.

### Parameters

- **device** – Device structure.

**int pi\_rtc\_datetime\_set(struct pi\_device \*device, struct tm \*time)**  
Set date and time.

This function is used to set current date and current time.

---

### Note:

Standard is not used to set the struct tm here.

Ex: 20/12/2019 15:20:45

```
time->tm_year = 2019;
time->tm_mon = 12;
time->tm_date = 20;
time->tm_hour = 15;
time->tm_min = 20;
time->tm_sec = 45;
```

If this struct needs to be used with standarg functions, copy this struct into a new one, adjust members according to standards.

---

### Parameters

- **device** – Device structure.
- **time** – Current date and time to set.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
int pi_rtc_datetime_get(struct pi_device *device, struct tm *time)
    Get current date ans time.
```

This function retrieves current date and time.

#### Parameters

- **device** – Device structure.
- **time** – Current date and time to store.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
int pi_rtc_alarm_set(struct pi_device *device, struct tm *alarm)
    Set alaram.
```

This function is used to set alarm. The alarm can be one shot or rearmed. An IRQ is triggered when the alarm date and time are reached.

#### Parameters

- **device** – Device structure.
- **alarm** – Date and time to set alarm.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
int pi_rtc_alarm_get(struct pi_device *device, struct tm *alarm)
    Get current alaram.
```

This function retrieves alarm date and time, if set.

#### Parameters

- **device** – Device structure.
- **alarm** – Curremt alarm date and time to store.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

```
int pi_rtc_timer_set(struct pi_device *device, uint32_t countdown)
    Set counter.
```

This function is used to set counter. The timer counts down from the given till 0, and triggers an IRQ.

#### Parameters

- **device** – Device structure.
- **countdown** – Countdown value.

**Returns 0** If operation is successful.

**Returns ERRNO** An error code otherwise.

---

```
uint32_t pi_rtc_timer_get(struct pi_device *device)
    Get current counter value.
```

This function returns current timer value.

#### Parameters

- **device** – Device structure.

**Returns Value** Current timer value..

```
int pi_rtc_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
    Iioctl command.
```

This function allows to send different commands to RTC device. The commands are listed above in pi\_rtc\_ioctl\_cmd\_e.

#### Parameters

- **device** – Device structure.
- **cmd** – Iioctl command.
- **arg** – Iioctl command args.

**Returns -1** If wrong ioctl command.

**Returns Value** Otherwise return value depending on ioctl command.

```
struct pi_rtc_conf
    #include <rtc.h> RTC configuration structure.
```

#### Public Members

```
uint8_t rtc_id
    RTC device ID.
```

```
pi_rtc_mode_e mode
    RTC mode.
```

```
struct tm time
    Current time.
```

```
struct tm alarm
    Alarm to set.
```

```
uint32_t counter
    Counter initial value.
```

```
int clk_div
    Set the clock divider.
```

## SPI

### group SPI

The SPIM driver provides support for transferring data between an external SPIM device and the chip running this driver.

#### Defines

```
PI_SPI_DUMMY_CLK_CYCLE_MODE_DEFAULT  
PI_SPI_DUMMY_CLK_CYCLE_DEFAULT  
PI_SPI_DUMMY_CLK_CYCLE_MIN  
PI_SPI_DUMMY_CLK_CYCLE_MAX  
PI_SPI_IS_VALID_DUMMY_CLK_CYCLE(cycle)  
  
PI_SPI_IS_VALID_DUMMY_CLK_CYCLE_MODE(mode)
```

#### Enums

##### enum **pi\_spi\_wordsize\_e**

Wordsize of the SPI bitstream elements.

This is used to know how the endianness must be applied. Not all sizes are supported on all chips, check the chip-specific section to get more information.

*Values:*

###### enumerator **PI\_SPI\_WORDSIZE\_8** = 0

Each element is 8 bits. Thus the endianness has no effect.

###### enumerator **PI\_SPI\_WORDSIZE\_16** = 1

Each element is 16 bits. The way each element is stored in memory can then be specified with the endianness.

###### enumerator **PI\_SPI\_WORDSIZE\_32** = 2

Each element is 32 bits. The way each element is stored in memory can then be specified with the endianness.

##### enum **pi\_spi\_polarity\_e**

Clock polarity.

*Values:*

###### enumerator **PI\_SPI\_POLARITY\_0** = 0

Leading edge is rising edge, trailing edge is falling edge.

###### enumerator **PI\_SPI\_POLARITY\_1** = 1

Leading edge is falling edge, trailing edge is rising edge.

---

**enum pi\_spi\_dummy\_clk\_cycle\_mode\_e**

*Values:*

enumerator **PI\_SPI\_DUMMY\_CLK\_CYCLE\_BEFORE\_CS** = 0

Dummy clock cycles are sent before toggling Chip select

enumerator **PI\_SPI\_DUMMY\_CLK\_CYCLE\_AFTER\_CS** = 1

Dummy clock cycles are sent after toggling Chip select

**enum pi\_spi\_phase\_e**

Clock phase.

*Values:*

enumerator **PI\_SPI\_PHASE\_0** = 0

Data shifted out on trailing edge of preceding clock cycle. Data sampled on leading edge of clock cycle.

enumerator **PI\_SPI\_PHASE\_1** = 1

Data shifted out on leading edge of current clock cycle. Data sampled on trailing edge of clock cycle.

**enum pi\_spi\_ioctl\_e**

Possible parameters which can be set through the pi\_spi\_control API function.

This is used to reconfigure dynamically some of the parameters of an opened device.

*Values:*

enumerator **PI\_SPI\_CTRL\_CPOL0** = 1 << **\_PI\_SPI\_CTRL\_CPOL\_BIT**

Set the clock polarity to 0.

enumerator **PI\_SPI\_CTRL\_CPOL1** = 2 << **\_PI\_SPI\_CTRL\_CPOL\_BIT**

Set the clock polarity to 1.

enumerator **PI\_SPI\_CTRL\_CPHA0** = 1 << **\_PI\_SPI\_CTRL\_CPHA\_BIT**

Set the clock phase to 0.

enumerator **PI\_SPI\_CTRL\_CPHA1** = 2 << **\_PI\_SPI\_CTRL\_CPHA\_BIT**

Set the clock phase to 1.

enumerator **PI\_SPI\_CTRL\_WORDSIZE\_8** = 1 << **\_PI\_SPI\_CTRL\_WORDSIZE\_BIT**

Set the wordsize to 8 bits.

enumerator **PI\_SPI\_CTRL\_WORDSIZE\_32** = 2 << **\_PI\_SPI\_CTRL\_WORDSIZE\_BIT**

Set the wordsize to 32 bits.

enumerator **PI\_SPI\_CTRL\_BIG\_ENDIAN** = 1 << **\_PI\_SPI\_CTRL\_ENDIANNESS\_BIT**

Handle the elements in memory in a big-endian way.

enumerator **PI\_SPI\_CTRL\_LITTLE\_ENDIAN** = 2 << **\_PI\_SPI\_CTRL\_ENDIANNESS\_BIT**

Handle the elements in memory in a little-endian way.

```
enumerator PI_SPI_CTRL_SET_MAX_BAUDRATE = 1 <<
    __PI_SPI_CTRL_SET_MAX_BAUDRATE_BIT
    Change maximum baudrate.

enumerator PI_SPI_CTRL_SET_TIMESTAMP = 1 << __PI_SPI_CTRL_SET_TIMESTAMP
    Enable the timestamp for SPI.
```

enum **pi\_spi\_flags\_e**  
Specifies additional behaviors for transfers.  
This flags can be given when transferring data.  
*Values:*

```
enumerator PI_SPI_CS_AUTO = 0 << 0
    Handles the chip select automatically. It is set low just before the transfer is started and set back high when the transfer is finished.

enumerator PI_SPI_CS_KEEP = 1 << 0
    Handle the chip select manually. It is set low just before the transfer is started and is kept low until the next transfer.

enumerator PI_SPI_CS_NONE = 2 << 0
    Don't do anything with the chip select.

enumerator PI_SPI_LINES_SINGLE = 0 << 2
    Use a single MISO line.

enumerator PI_SPI_LINES_QUAD = 1 << 2
    Use quad MISO lines.

enumerator PI_SPI_LINES_OCTAL = 2 << 2
    Use octal MISO lines.

enumerator PI_SPI_COPY_EXT2LOC = 1 << 4
    Do a copy from external memory to local chip memory.

enumerator PI_SPI_COPY_LOC2EXT = 0 << 4
    Do a copy from local chip memory to external memory.
```

## Functions

```
void pi_spi_conf_init(struct pi_spi_conf *conf)
    Initialize an SPI master configuration with default values.
```

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the SPI device is opened.

### Parameters

- **conf** – A pointer to the SPI master configuration.

```
int pi_spi_open(struct pi_device *device)
    Open an SPI device.
```

This function must be called before the SPI device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the caller and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
void pi_spi_close(struct pi_device *device)
    Close an opened SPI device.
```

This function can be called to close an opened SPI device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used. The caller is blocked until the operation is finished.

### Parameters

- **device** – A pointer to the structure describing the device.

```
void pi_spi_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
    Dynamically change the device configuration.
```

This function can be called to change part of the device configuration after it has been opened.

### Parameters

- **device** – A pointer to the structure describing the device.
- **cmd** – The command which specifies which parameters of the driver to modify and for some of them also their values. The command must be one of those defined in `pi_spi_ioctl_e`.
- **arg** – An additional value which is required for some parameters when they are set.

```
void pi_spi_send(struct pi_device *device, void *data, size_t len, pi_spi_flags_e flag)
    Enqueue a write copy to the SPI (from Chip to SPI device).
```

This function can be used to send data to the SPI device. The copy will make a synchronous transfer between the SPI and one of the chip memory. This is by default using classic SPI transfer with MOSI and MISO lines, but other kind of transfers like quad SPI can be used by specifying additional flags. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of 4. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – A pointer to the structure describing the device.
- **data** – The address in the chip where the data to be sent must be read.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using `PI_SPI_CS_AUTO` and `PI_SPI_LINES_SINGLE`.

```
void pi_spi_receive(struct pi_device *device, void *data, size_t len, pi_spi_flags_e flag)
    Enqueue a read copy to the SPI (from Chip to SPI device).
```

This function can be used to receive data from the SPI device. The copy will make a synchronous transfer between the SPI and one of the chip memory. This is by default using classic SPI transfer with MOSI and MISO lines, but other kind of transfers like quad SPI can be used by specifying additional flags. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of

4. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **data** – The address in the chip where the received data must be written.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using PI\_SPI\_CS\_AUTO and PI\_SPI\_LINES\_SINGLE.

```
void pi_spi_transfer(struct pi_device *device, void *tx_data, void *rx_data, size_t len, pi_spi_flags_e flag)
```

Enqueue a read and write copy to the SPI (using full duplex mode).

This function can be used to send and receive data with the SPI device using full duplex mode. The copy will make a synchronous transfer between the SPI and one of the chip memory. This is using classic SPI transfer with MOSI and MISO lines. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of 4. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent must be read.
- **rx\_data** – The address in the chip where the received data must be written.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using PI\_SPI\_CS\_AUTO and PI\_SPI\_LINES\_SINGLE.

```
void pi_spi_send_async(struct pi_device *device, void *data, size_t len, pi_spi_flags_e flag, pi_evt_t *task)
```

Enqueue an asynchronous write copy to the SPI (from Chip to SPI device).

This function can be used to send data to the SPI device. The copy will make an asynchronous transfer between the SPI and one of the chip memory. This is by default using classic SPI transfer with MOSI and MISO lines, but other kind of transfers like quad SPI can be used by specifying additional flags. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of 4. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **data** – The address in the chip where the data to be sent must be read.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using PI\_SPI\_CS\_AUTO and PI\_SPI\_LINES\_SINGLE.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

---

```
void pi_spi_receive_async(struct pi_device *device, void *data, size_t len, pi_spi_flags_e flag, pi_evt_t
                         *task)
```

Enqueue an asynchronous read copy to the SPI (from Chip to SPI device).

This function can be used to receive data from the SPI device. The copy will make an asynchronous transfer between the SPI and one of the chip memory. This is by default using classic SPI transfer with MOSI and MISO lines, but other kind of transfers like quad SPI can be used by specifying additional flags. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of 4. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **data** – The address in the chip where the received data must be written.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using PI\_SPI\_CS\_AUTO and PI\_SPI\_LINES\_SINGLE.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.details.details.

```
void pi_spi_transfer_async(struct pi_device *device, void *tx_data, void *rx_data, size_t len,
                           pi_spi_flags_e flag, pi_evt_t *task)
```

Enqueue an asynchronous read and write copy to the SPI (using full duplex mode).

This function can be used to send and receive data with the SPI device using full duplex flag. The copy will make an asynchronous transfer between the SPI and one of the chip memory. This is using classic SPI transfer with MOSI and MISO lines. Due to hardware constraints, the address of the buffer must be aligned on 4 bytes and the size must be a multiple of 4. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – A pointer to the structure describing the device.
- **tx\_data** – The address in the chip where the data to be sent must be read.
- **rx\_data** – The address in the chip where the received data must be written.
- **len** – The size in bits of the copy.
- **flag** – Additional behaviors for the transfer can be specified using this flag. Can be 0 to use the default flag, which is using PI\_SPI\_CS\_AUTO and PI\_SPI\_LINES\_SINGLE.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.details.details.

```
struct pi_spi_conf
```

## Public Members

**int max\_baudrate**

Maximum baudrate for the SPI bitstream which can be used with the opened device .

**char wordsize**

Wordsize of the elements in the bitstream. Can be PI\_SPI\_WORDSIZE\_8 for 8 bits data or PI\_SPI\_WORDSIZE\_32 for 32 bits data. This is used to interpret the endianness.

**char big\_endian**

If 1, the elements are stored in memory in a big-endian way, i.e. the most significant byte is stored at the lowest address. This is taken into account only if the wordsize is 32 bits.

*pi\_spi\_polarity\_e* **polarity**

Polarity of the clock.

*pi\_spi\_phase\_e* **phase**

Phase of the clock.

**signed char cs**

Specifies which SPI chip select is used for the device.

**signed char itf**

Specifies on which SPI interface the device is connected.

**int max\_rcv\_chunk\_size**

Specifies maximum chunk size for reception when using copies.

**int max\_snd\_chunk\_size**

Specifies maximum chunk size for sending when using copies.

**int is\_slave**

If 1, the SPI interface is configured as a slave.

**uint8\_t ts\_ch**

Enable the timestamp on TX (0) or RX (1)

**uint8\_t ts\_evt\_id**

UDMA Config Event ID for generating the timestamp

**uint8\_t dummy\_clk\_cycle**

Number of clock cycles to send before sending the data.

*pi\_spi\_dummy\_clk\_cycle\_mode\_e* **dummy\_clk\_cycle\_mode**

Select if dummy clock cycles are sent before or after toggling the chip select.

**struct pi\_spi\_conf\_t**

#include <spi.h> SPI master configuration structure.

This structure is used to pass the desired SPI master configuration to the runtime when opening a device.

## UART

### group UART

UART Universal Asynchronous Receiver Transmitter.

This API provides support for transferring data between an external UART device and the processor running this driver.

### Defines

```
pi_uart_timeout_get_bytes_left
pi_uart_timeout_get_curr_addr
```

### Typedefs

```
typedef struct pi_cl_uart_req_s pi_cl_uart_req_t
UART cluster request structure.
```

This structure is used by the runtime to manage a cluster remote copy with the UART. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

### Enums

```
enum pi_uart_stop_bits
Stop bits enum.
```

*Values:*

```
enumerator PI_UART_STOP_BITS_ONE = 0
One stop bit.
```

```
enumerator PI_UART_STOP_BITS_TWO = 1
Two stop bits.
```

```
enum pi_uart_parity_mode
Parity mode enum.
```

*Values:*

```
enumerator PI_UART_PARITY_DISABLE = 0
Disable parity mode.
```

```
enumerator PI_UART_PARITY_ENABLE = 1
Enable parity mode.
```

```
enum pi_uart_word_size
Bit length of each word.
```

*Values:*

enumerator **PI\_UART\_WORD\_SIZE\_5\_BITS** = 0  
5 bits length.

enumerator **PI\_UART\_WORD\_SIZE\_6\_BITS** = 1  
6 bits length.

enumerator **PI\_UART\_WORD\_SIZE\_7\_BITS** = 2  
7 bits length.

enumerator **PI\_UART\_WORD\_SIZE\_8\_BITS** = 3  
8 bits length.

enum **pi\_uart\_ioctl\_cmd**  
UART ioctl commands.

UART ioctl commands to configure, enable device.

*Values:*

enumerator **PI\_UART\_IOCTL\_CONF\_SETUP** = 0  
Setup UART device.

Setup UART with given conf. The parameter for this command is a struct *pi\_uart\_conf*.

**Parameters**

- **conf** – Pointer to struct *pi\_uart\_conf*.

enumerator **PI\_UART\_IOCTL\_ABORT\_RX** = 1  
Abort RX transfers.

Disable RX channel, abort current RX transfert, and flush all pending transferts.

---

**Note:** This function disables reception channel after clearing UDMA channels. In order to send again data, the reception channel must re-enabled.

---

enumerator **PI\_UART\_IOCTL\_ABORT\_TX** = 2  
Abort TX transfers.

Disable TX channel, abort current TX transfert, and flush all pending transferts.

---

**Note:** This function disables transmission channel after clearing UDMA channels. In order to send again data, the transmission channel must re-enabled.

---

enumerator **PI\_UART\_IOCTL\_ENABLE\_RX** = 3  
Enable reception.

This command enables reception on UART device.

enumerator **PI\_UART\_IOCTL\_ENABLE\_TX** = 4  
Enable transmission.

This command enables transmission on UART device.

---

enumerator **PI\_UART\_IOCTL\_ENABLE\_FLOW\_CONTROL** = 5  
Enable flow control.

This command enables flow control on UART device.

---

**Note:** On GAP8, flow control is emulated using a PWM device(Timer0, channel2) and two GPIOs(PI\_GPIO\_A1\_PAD\_9\_B3, PI\_GPIO\_A0\_PAD\_8\_A4).

---



---

**Note:** On GAP9 this will disable and re-enable TX and RX.

---

enumerator **PI\_UART\_IOCTL\_DISABLE\_FLOW\_CONTROL** = 6  
Disable flow control.

This command disables flow control on UART device.

enumerator **PI\_UART\_IOCTL\_FLUSH** = 7  
Flush UART TX.

This command will wait until all pending buffers are flushed outside

enumerator **PI\_UART\_IOCTL\_ATTACH\_TIMEOUT\_RX** = 8  
Attach UDMA timer.

This command attaches a UDMA timer channel to UDMA reception channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_UART\_IOCTL\_DETACH\_TIMEOUT\_RX** = 9  
Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA reception channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_UART\_IOCTL\_ATTACH\_TIMEOUT\_TX** = 10  
Attach UDMA timer.

This command attaches a UDMA timer channel to UDMA transmission channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_UART\_IOCTL\_DETACH\_TIMEOUT\_TX** = 11  
Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA transmission channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_UART\_IOCTL\_RESUME\_TX** = 12  
Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA transmission channel.

**Parameters**

- **timeout\_id** – UDMA timeout channel ID.

enumerator **PI\_UART\_IOCTL\_RESUME\_RX** = 13  
Detach UDMA timer.

This command removes a UDMA timer channel attached to UDMA transmission channel.

#### Parameters

- **timeout\_id** – UDMA timeout channel ID.

enum **pi\_uart\_timeout\_flags**

*Values:*

enumerator **PI\_UART\_TIMEOUT\_FLAG\_HW** = 1  
Use hw based timeout.

enumerator **PI\_UART\_TIMEOUT\_FLAG\_SW** = 2  
Use SW based timeout.

enumerator **PI\_UART\_TIMEOUT\_FLAG\_SW\_HW** = 3  
Use SW based timeout + HW Use the same timeout\_us.

## Functions

void **pi\_uart\_conf\_init**(struct *pi\_uart\_conf* \*conf)

Initialize a UART configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the uart device is opened.

#### Parameters

- **conf** – Pointer to the UART configuration.

int **pi\_uart\_open**(struct *pi\_device* \*device)

Open a UART device.

This function must be called before the UART device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

---

**Note:** This structure is allocated by the called and must be kept alive until the device is closed.

---

#### Parameters

- **device** – Pointer to device structure of the device to open.

**Returns 0** If the operation is successfull.

**Returns ERRNO** An error code otherwise.

void **pi\_uart\_close**(struct *pi\_device* \*device)

Close an opened UART device.

This function can be called to close an opened UART device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – Pointer to device structure of the device to close.

**int pi\_uart\_ioctl(struct pi\_device \*device, uint32\_t cmd, void \*arg)**

Dynamically change device configuration.

This function allows to send different commands to UART device. The commands are listed above, cf. enum pi\_uart\_ioctl\_cmd.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **cmd** – Ioctl command.
- **arg** – Ioctl command args.

**Returns -1** If wrong ioctl command.

**Returns Value** Otherwise return value depending on ioctl command.

**int pi\_uart\_write\_async(struct pi\_device \*device, void \*buffer, uint32\_t size, pi\_evt\_t \*callback)**

Write data to an UART asynchronously.

## WRITE FUNCTIONS

This writes data to the specified UART asynchronously. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.
- **callback** – Event task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

**int pi\_uart\_write(struct pi\_device \*device, void \*buffer, uint32\_t size)**

Write data to an UART.

This writes data to the specified UART. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_write_byte_async(struct pi_device *device, uint8_t *byte, pi_evt_t *callback)
```

Write a byte to an UART asynchronously.

This writes a byte to the specified UART asynchronously. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **byte** – Pointer to data buffer.
- **callback** – Event task used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_write_byte(struct pi_device *device, uint8_t *byte)
```

Write a byte to an UART.

This writes a byte to the specified UART. The caller is blocked until the transfer is finished.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **byte** – Pointer to data buffer.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_write_timeout_async(struct pi_device *device, void *buffer, uint32_t size, uint32_t
                               timeout_us, pi_evt_t *timeout_task, pi_uart_timeout_flags flags, void
                               *args)
```

Write data from an UART asynchronously, with timeout.

This function is the same as `pi_uart_write_async()`, with timeout feat enabled. This timeout value is used to abort/cancel a transfer when timeout is reached.

The timeout can be used to advertise the user that no byte has been written, however it does NOT guaranty the data incoming once timeout has occurred. The user should abort and restart, or deals with it.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this function :

- `pi_udma_timeout_alloc()`
  - `pi_uart_ioctl() //PI_UART_IOCTL_ATTACH_TIMEOUT_RX`
  - `pi_uart_write_timeout()`
- 

**Note:** Once `timeout_task` is pushed, use `pi_evt_status_get()` to know if the timeout has been reached (`status == PI_ERR_TIMEOUT`).

---

**Warning:** \* `timeout_task` MUST be initialized! Holds the timeout status and works as the callback of the function

---

- Once timeout\_task is pushed, if you use the software timeout, please CALL [pi\\_evt\\_cancel\\_delayed\\_us\(\)](#) with the sw\_timeout\_task that you have created!

### Parameters

- device** – Pointer to device descriptor of the UART device.
- buffer** – Pointer to data buffer.
- size** – Size of data to copy in bytes.
- timeout\_us** – Timeout value in us.
- timeout\_task** – Task which will be used to hold the timeout status, and for callback.
- flags** – Flags to configure actual timeout.
- args** – When using software timeout, arguments for it. See `uart_sw_timeout_arg_t` structure.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

**Returns** Opeartion status.

```
int pi_uart_write_timeout(struct pi_device *device, void *buffer, uint32_t size, uint32_t timeout_us,
                         pi_evt_t *timeout_task, pi\_uart\_timeout\_flags flags)
```

Write data to an UART synchronously, with timeout.

This function is the same as [pi\\_uart\\_write\(\)](#), with timeout feat enabled. This timeout value is used to abort/cancel a transfer when timeout is reached.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this functions :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_uart\\_ioctl\(\) //PI\\_UART\\_IOCTL\\_ATTACH\\_TIMEOUT\\_TX](#)
  - [pi\\_uart\\_write\\_timeout\(\)](#)
- 

**Warning:** timeout\_task MUST NOT be initialized

### Parameters

- device** – Pointer to device descriptor of the UART device.
- buffer** – Pointer to data buffer.
- size** – Size of data to copy in bytes.
- timeout\_us** – Timeout value in us.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_read_async(struct pi_device *device, void *buffer, uint32_t size, pi_evt_t *callback)
```

Read data from an UART asynchronously.

## READ FUNCTIONS

This reads data from the specified UART asynchronously. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.
- **callback** – Event task used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_read(struct pi_device *device, void *buffer, uint32_t size)
```

Read data from an UART.

This reads data from the specified UART. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_read_byte(struct pi_device *device, uint8_t *byte)
```

Read a byte from an UART.

This reads a byte from the specified UART. The caller is blocked until the transfer is finished.

### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **byte** – Pointer to data buffer.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_uart_read_timeout_async(struct pi_device *device, void *buffer, uint32_t size, uint32_t timeout_us, pi_evt_t *timeout_task, pi_uart_timeout_flags flags, void *args)
```

Read data from an UART asynchronously, with timeout.

This function is the same as [pi\\_uart\\_read\\_async\(\)](#), with timeout feat enabled. This timeout value is used to abort/cancel a transfer when timeout is reached.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this function :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
  - [pi\\_uart\\_ioctl\(\) //PI\\_UART\\_IOCTL\\_ATTACH\\_TIMEOUT\\_RX](#)
  - [pi\\_uart\\_read\\_timeout\(\)](#)
- 

---

**Note:** Once timeout\_task is pushed, use [pi\\_evt\\_status\\_get\(\)](#) to know if the timeout has been reached (status == PI\_ERR\_TIMEOUT).

---

**Warning:** \* timeout\_task MUST be initialized! Holds the timeout status and works as the callback of the function

- Once timeout\_task is pushed, if you use the software timeout, please CALL [pi\\_evt\\_cancel\\_delayed\\_us\(\)](#) with the sw\_timeout\_task that you have created!

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to read in bytes.
- **timeout\_us** – Timeout value in us.
- **timeout\_task** – Task which will be used to hold the timeout status, and for callback.
- **flags** – Flags to configure actual timeout.
- **args** – When using software timeout, arguments for it. See [uart\\_sw\\_timeout\\_arg\\_t](#) structure.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

**Returns** Opeartion status.

```
int pi_uart_read_timeout(struct pi_device *device, void *buffer, uint32_t size, uint32_t timeout_us,
                        pi_evt_t *timeout_task, pi\_uart\_timeout\_flags flags)
```

Read data from an UART synchronously, with timeout.

This function is the same as [pi\\_uart\\_read\(\)](#), with timeout feat enabled. This timeout value is used to abort/cancel a transfer when timeout is reached.

---

**Note:** To use this feature, a UDMA timeout channel must be allocated before a call to this function :

- [pi\\_udma\\_timeout\\_alloc\(\)](#)
- [pi\\_uart\\_ioctl\(\) //PI\\_UART\\_IOCTL\\_ATTACH\\_TIMEOUT\\_RX](#)
- [pi\\_uart\\_read\\_timeout\(\)](#)

**Warning:** timeout\_task MUST NOT be initialized

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to read in bytes.
- **timeout\_us** – Timeout value in us.
- **timeout\_task** – Task which will be used to hold the timeout status.
- **flags** – Flags to configure actual timeout.

**Returns 0** If operation is successfull.

**Returns PI\_ERR\_TIMEOUT** Error code if timeout has been reached.

**Returns ERRNO** An error code otherwise.

**Returns** Operation status.

```
int pi_cl_uart_write(pi_device_t *device, void *buffer, uint32_t size, pi_cl_uart_req_t *req)  
Write data to an UART from cluster side.
```

### CLUSTER SIDE FUNCTIONS

This function implements the same feature as pi\_uart\_write but can be called from cluster side in order to expose the feature on the cluster. A pointer to a request structure must be provided so that the runtime can properly do the remote call.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.
- **req** – Request structure used for termination.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_cl_uart_write_byte(pi_device_t *device, uint8_t *byte, pi_cl_uart_req_t *req)  
Write a byte to an UART from cluster side.
```

This function implements the same feature as pi\_uart\_write\_byte but can be called from cluster side in order to expose the feature on the cluster. A pointer to a request structure must be provided so that the runtime can properly do the remote call.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **byte** – Pointer to data buffer.

- **req** – Request structure used for termination.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
static inline void pi_cl_uart_write_wait(pi_cl_uart_req_t *req)
```

Wait until the specified UART cluster write request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – Request structure used for termination.

```
int pi_cl_uart_read(pi_device_t *device, void *buffer, uint32_t size, pi_cl_uart_req_t *req)
```

Read a byte from an UART from cluster side.

This function implements the same feature as pi\_uart\_read\_byte but can be called from cluster side in order to expose the feature on the cluster. A pointer to a request structure must be provided so that the runtime can properly do the remote call.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **buffer** – Pointer to data buffer.
- **size** – Size of data to copy in bytes.
- **req** – Request structure used for termination.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
int pi_cl_uart_read_byte(pi_device_t *device, uint8_t *byte, pi_cl_uart_req_t *req)
```

Read a byte from an UART.

This reads a byte from the specified UART. The caller is blocked until the transfer is finished.

#### Parameters

- **device** – Pointer to device descriptor of the UART device.
- **byte** – Pointer to data buffer.
- **req** – Request structure used for termination.

**Returns 0** If operation is successfull.

**Returns ERRNO** An error code otherwise.

```
static inline void pi_cl_uart_read_wait(pi_cl_uart_req_t *req)
```

Wait until the specified UART cluster read request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

#### Parameters

- **req** – Request structure used for termination.

```
void pi_uart_pause(struct pi_device *device, int channel, pi_evt_t *task)
```

Pause the current uart transfer.

## PAUSE FUNCTIONS

This blocks the driver from executing any transfer until a resume happens.

**Warning:** To use this function, uart control flow MUST be enabled !

#### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.

```
void pi_uart_resume_async(struct pi_device *device, int channel, pi_evt_t *task)
```

Resume the current uart transfer, without any timeout.

### RESUME FUNCTIONS

---

**Note:** Asynchronous. Use ‘task’ as the callback: need to call pi\_evt\_sig\_init or similar before.

---

---

**Note:** Once task is pushed, use pi\_evt\_status\_get() to know if the timeout has been reached (status == PI\_ERR\_TIMEOUT).

---

#### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.

```
int pi_uart_resume(struct pi_device *device, int channel, pi_evt_t *task)
```

Resume the current uart transfer, without any timeout.

#### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.

**Returns 0** If operation is successfull.

**Returns PI\_ERR\_TIMEOUT** Error code if timeout has been reached.

**Returns ERRNO** An error code otherwise.

**Returns** Operation status.

```
void pi_uart_resume_timeout_async(struct pi_device *device, int channel, pi_evt_t *task, uint32_t
                                 timeout_us, pi_uart_timeout_flags flags)
```

Resume the current uart transfer and add a timeout again.

---

**Note:** Asynchronous. Use ‘task’ as the callback: need to call pi\_evt\_sig\_init or similar before.

---



---

**Note:** Once task is pushed, use pi\_evt\_status\_get() to know if the timeout has been reached (status == PI\_ERR\_TIMEOUT).

---

### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.
- **timeout\_us** – Number of micro seconds before next timeout.
- **flags** – Flags for the timeout.

```
int pi_uart_resume_timeout(struct pi_device *device, int channel, pi_evt_t *task, uint32_t timeout_us,
                           pi_uart_timeout_flags flags)
```

Resume the current uart transfer and add a timeout again.

### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.
- **timeout\_us** – Number of micro seconds before next timeout.
- **flags** – Flags for the timeout.

**Returns 0** If operation is successfull.

**Returns PI\_ERR\_TIMEOUT** Error code if timeout has been reached.

**Returns ERRNO** An error code otherwise.

**Returns** Operation status.

```
void pi_uart_abort(struct pi_device *device, int channel, pi_evt_t *task)
```

Abort the current uart transfer.

## ABORT FUNCTIONS

Aborts the current transfer, executes the next one if it exists.

### Parameters

- **device** – Uart device structure.
- **channel** – Channel (RX\_CHANNEL or TX\_CHANNEL).
- **task** – Task that will be filled with necessary info to resume the transfer.

```
uint32_t pi_uart_get_bytes_left(pi_evt_t *task_timeout)
```

Returns number of bytes left, at the moment of timeout irq.

## OTHER FUNCTIONS

### Parameters

- **timeout\_task** – pi\_task used for timeout access

**Returns** `uint32_t` number of bytes left in transfer

```
uint32_t pi_uart_get_curr_addr(pi_evt_t *task_timeout)
```

Returns current pointer in l2 buffer, at the moment of timeout irq.

### Parameters

- **timeout\_task** – pi\_task used for timeout access

**Returns** `uint32_t` Pointer to current l2 address in the buffer

```
struct pi_uart_conf
```

`#include <uart.h>` UART device configuration structure.

This structure is used to pass the desired UART configuration to the runtime when opening the device.

### Public Members

`uint32_t baudrate_bps`

Required baudrate, in baud per second.

`uint8_t stop_bit_count`

Number of stop bits, 1 stop bit (default) or 2 stop bits

`uint8_t parity_mode`

1 to activate it, 0 to deactivate it. Even.

`uint8_t word_size`

Word size, in bits.

`uint8_t enable_rx`

1 to activate reception, 0 to deactivate it.

`uint8_t enable_tx`

1 to activate transmission, 0 to deactivate it.

`uint8_t uart_id`

Uart interface ID.

`uint8_t use_ctrl_flow`

1 to activate control flow.

`uint8_t is_usart`

1 to activate usart

`uint8_t usart_polarity`  
If 1, the clock polarity is reversed.

`uint8_t usart_phase`  
If 0, the data are sampled on the first clock edge, otherwise on the second clock edge.

`uint8_t use_fast_clk`  
If 0, use fll periph as source otherwise use external fast clock.

## UDMA Datamove

### group UDMA\_DATAMOVE

The UDMA DATAMOVE driver provides an interface to use UDMA DATAMOVE channel.

#### TypeDefs

```
typedef struct pi_udma_datamove_transf_cfg_s pi_udma_datamove_transf_cfg_t
typedef struct pi_udma_datamove_conf_s pi_udma_datamove_conf_t
```

UDMA DATAMOVE configuration structure.

This structure is used to pass the desired UDMA DATAMOVE configuration to the runtime when opening a device.

#### Enums

```
enum pi_udma_datamove_transf_type_e
    type of transfer
```

*Values:*

```
enumerator PI_UDMA_DATAMOVE_TRF_LINEAR
enumerator PI_UDMA_DATAMOVE_TRF_2D
```

#### Functions

```
void pi_udma_datamove_conf_init(pi_udma_datamove_conf_t *conf)
```

Initialize an UDMA DATAMOVE configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the UDMA DATAMOVE device is opened.

#### Parameters

- **conf** – A pointer to the UDMA DATAMOVE configuration.

```
int pi_udma_datamove_open(pi_device_t *device)
```

Open an UDMA DATAMOVE device.

This function must be called before the UDMA DATAMOVE device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

**Warning:** The datamove needs at least one UDMA peripheral enabled to be enabled. Else configuration will not be set, and application will get stuck on push/pop commands.

#### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
void pi_udma_datamove_close(pi_device_t *device)
```

Close an opened UDMA DATAMOVE device.

This function can be called to close an opened UDMA DATAMOVE device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

#### Parameters

- **device** – The device structure of the device to close.

```
int32_t pi_udma_datamove_copy(pi_device_t *device, void *src, void *dst, uint32_t len)
```

copy data from L2 to L2 (blocking)

#### Parameters

- **device** – udma\_datamove device pointer
- **src** – source address
- **dst** – destination address
- **len** – length of the transfer in bytes

**Returns** PI\_OK if operation succeeded, else error code

```
int32_t pi_udma_datamove_copy_async(pi_device_t *device, void *src, void *dst, uint32_t len, pi_evt_t *task)
```

copy data from L2 to L2 (asynchronous)

#### Parameters

- **device** – udma\_datamove device pointer
- **src** – source address
- **dst** – destination address
- **len** – length of the transfer in bytes
- **task** – task executed at the end of transfer

**Returns** PI\_OK if operation succeeded, else error code

```
struct pi_udma_datamove_transf_cfg_s
```

---

```
struct pi_udma_datamove_conf_s
#include <udma_datamove.h> UDMA DATAMOVE configuration structure.
```

This structure is used to pass the desired UDMA DATAMOVE configuration to the runtime when opening a device.

### Public Members

```
uint8_t device_id
device id
```

```
pi_udma_datamove_transf_cfg_t src_trf_cfg
Source data transfer configuration
```

```
pi_udma_datamove_transf_cfg_t dst_trf_cfg
Destination data transfer configuration
```

## UDMA Fifo

### group **UDMA\_FIFO**

The UDMA FIFO driver provides an interface to use UDMA FIFO channel.

### TypeDefs

```
typedef struct pi_udma_fifo_conf pi_udma_fifo_conf_t
```

This structure is used to pass the desired UDMA FIFO configuration to the runtime when opening a device.

### Enums

#### enum **pi\_udma\_fifo\_ioctl\_e**

Commands for pi\_udma\_fifo\_ioctl.

This is used to tell which command to execute through pi\_udma\_fifo\_ioctl. Parameters are passed as pointers.

*Values:*

##### enumerator **PI\_UDMA\_FIFO\_GET\_ID**

type uint32\_t: return the fifo id

##### enumerator **PI\_UDMA\_FIFO\_SET\_SIZE**

set the FIFO size (in bytes, minimum 16)

##### enumerator **PI\_UDMA\_FIFO\_GET\_LEVEL**

get current number of bytes in FIFO

**enumerator PI\_UDMA\_FIFO\_SET\_EVENT\_THRESHOLD**

type uint32\_t: set the number of bytes after which the event will be triggered. Setting a value greater than 0 enable the event, setting 0 disables it

**enumerator PI\_UDMA\_FIFO\_SET\_EVENT\_CB**

type pi\_evt\_t: set the event callback, use NULL for no callback

**enumerator PI\_UDMA\_FIFO\_PUSH8**

type uint8\_t: push 8 bits in the FIFO

**enumerator PI\_UDMA\_FIFO\_PUSH16**

type uint16\_t: push 16 bits in the FIFO

**enumerator PI\_UDMA\_FIFO\_PUSH24**

type uint32\_t: push 24 bits in the FIFO

**enumerator PI\_UDMA\_FIFO\_PUSH32**

type uint32\_t: push 32 bits in the FIFO

**enumerator PI\_UDMA\_FIFO\_POP8**

type uint8\_t: pop 8 bits from the FIFO

**enumerator PI\_UDMA\_FIFO\_POP16**

type uint16\_t: pop 16 bits from the FIFO

**enumerator PI\_UDMA\_FIFO\_POP24**

type uint32\_t: pop 24 bits from the FIFO

**enumerator PI\_UDMA\_FIFO\_POP32**

type uint32\_t: pop 32 bits from the FIFO

## Functions

**void pi\_udma\_fifo\_conf\_init(pi\_udma\_fifo\_conf\_t \*conf)**

Initialize an UDMA FIFO configuration with default values.

This function can be called to get default values for all parameters before setting some of them. The structure containing the configuration must be kept alive until the UDMA FIFO device is opened.

### Parameters

- **conf** – A pointer to the UDMA FIFO configuration.

**int pi\_udma\_fifo\_open(pi\_device\_t \*device)**

Open an UDMA FIFO device.

This function must be called before the UDMA FIFO device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

**Warning:** The fifo needs at least one UDMA peripheral enabled to be enabled. Else configuration will not be set, and application will get stuck on push/pop commands.

**Parameters**

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
void pi_udma_fifo_close(pi_device_t *device)
```

Close an opened UDMA FIFO device.

This function can be called to close an opened UDMA FIFO device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

**Parameters**

- **device** – The device structure of the device to close.

```
void pi_udma_fifo_ioctl(pi_device_t *device, uint32_t cmd, void *arg)
```

UDMA FIFO IOCTL function.

**Parameters**

- **device** – A pi device structure pointing to UDMA FIFO device
- **cmd** – ioctl number
- **arg** – argument to be passed to ioctl

```
struct pi_udma_fifo_conf
```

#include <udma\_fifo.h> This structure is used to pass the desired UDMA FIFO configuration to the runtime when opening a device.

**Public Members**

pi\_device\_e **device**

Device type.

uint32\_t **size**

size of the FIFO in bytes, minimum value of 16

**UDMA Timeout**

group **UDMA\_TIMEOUT**

Timeout.

This API provides support to handle timeout feature on UDMA channels.

## Enums

### enum `pi_udma_timeout_mode_e`

UDMA timeout mode enum.

*Values:*

enumerator `PI_UDMA_TIMEOUT_MODE_SW_TRIGGER` = 0

Timeout triggered by SW.

enumerator `PI_UDMA_TIMEOUT_MODE_TRANSFER` = 1

Timeout triggered at the beginning of transfer.

enumerator `PI_UDMA_TIMEOUT_MODE_RXTX` = 2

Timeout triggered at the beginning of transfer and cleared at each data received.

### enum `pi_udma_timeout_ioctl_cmd`

UDMA timeout ioctl commands.

*Values:*

enumerator `PI_UDMA_TIMEOUT_IOCTL_START` = 0

Start timeout counter.

Start UDMA timeout counter.

---

**Note:** This is to be used with `PI_UDMA_TIMEOUT_MODE_SW_TRIGGER`.

---

enumerator `PI_UDMA_TIMEOUT_IOCTL_STOP` = 1

Stop timeout counter.

Stop UDMA timeout counter and reset it.

## Functions

### `int32_t pi_udma_timeout_alloc(pi_udma_timeout_mode_e mode)`

Allocate a UDMA timeout channel.

This function is called to allocate a UDMA timeout channel before using it. A UDMA timeout channel must be allocated before calling timeout functions.

#### Parameters

- `mode` – UDMA timeout channel mode.

**Returns -1** If no UDMA timeout channel available.

**Returns UTID** UDMA timeout channel ID.

### `void pi_udma_timeout_free(int32_t timeout_id)`

Free a UDMA timeout channel.

This function frees an allocated a UDMA timeout channel.

#### Parameters

- `timeout_id` – UDMA timeout channel to free.

---

```
int32_t pi_udma_timeout_ioctl(int32_t timeout_id, uint32_t cmd, void *arg)
```

Ioctl commands.

This function is used to configure UDMA timeout channels.

**Returns -1** If wrong ioctl command.

**Returns Value** Otherwise return value depending on ioctl command.

## UDMA Timestamp

*group* **UDMA\_TIMESTAMP**

Timestamp.

This API provides support to handle timestamp feature on UDMA channels.

### Enums

enum [**anonymous**]

*Values:*

```
enumerator PI_UDMA_TIMESTAMP_IOCTL_CLR = 1
enumerator PI_UDMA_TIMESTAMP_IOCTL_STOP
enumerator PI_UDMA_TIMESTAMP_IOCTL_EVT_ALLOC
enumerator PI_UDMA_TIMESTAMP_IOCTL_SET_EVT
enumerator PI_UDMA_TIMESTAMP_IOCTL_SET_DEST
enumerator PI_UDMA_TIMESTAMP_IOCTL_SET_INPUT
enumerator PI_UDMA_TIMESTAMP_IOCTL_FREE_EVT
enumerator PI_UDMA_TIMESTAMP_IOCTL_FREE_INPUT
enumerator PI_UDMA_TIMESTAMP_IOCTL_GET_LIN_ID
enumerator PI_UDMA_TIMESTAMP_IOCTL_GET_FIFO_ID
```

enum **pi\_timestamp\_cnt\_src\_e**

*Values:*

```
enumerator PI_TIMESTAMP_CNT_PWM = 0
enumerator PI_TIMESTAMP_CNT_GPIO
enumerator PI_TIMESTAMP_CNT_REF_CLK_QUICK
enumerator PI_TIMESTAMP_CNT_SOC_CLK
```

enum **pi\_timestamp\_cnt\_gpio\_trig\_type\_e**

*Values:*

```
enumerator PI_TIMESTAMP_GPIO_RISE_EDGE = 0
enumerator PI_TIMESTAMP_GPIO_FALL_EDGE
enumerator PI_TIMESTAMP_GPIO_BOTH_EDGE
```

enumerator **PI\_TIMESTAMP\_AUX\_INPUT**

## Functions

void **pi\_timestamp\_conf\_init**(struct *pi\_timestamp\_conf* \*conf)

void **pi\_udma\_timestamp\_open**(struct pi\_device \*timestamp)

void **pi\_udma\_timestamp\_close**(struct pi\_device \*timestamp)

int32\_t **pi\_udma\_timestamp\_ioctl**(struct pi\_device \*timestamp, uint32\_t cmd, void \*arg)

struct **pi\_timestamp\_conf**

### Public Members

uint8\_t **itf**

device ID for timestamp

uint8\_t **cnt\_trig\_gpio**

gpio number for trigger the timestamp counter

*pi\_timestamp\_cnt\_gpio\_trig\_type\_e* **cnt\_trig\_type**

how the gpio trigger the timestamp counter

*pi\_timestamp\_cnt\_src\_e* **cnt\_src**

timestamp counter source

uint8\_t **cnt\_src\_id**

GPIO/PWM ID depends on the counter source

uint8\_t **prescaler**

Prescaler for timestamp counter

struct **pi\_timestamp\_input\_t**

### Public Members

uint8\_t **ts\_input\_id**

Timestamp input ID, max 8 input. Reg0-7

uint8\_t **input\_sel**

Timestamp input selection: if input\_type=3, then 0-7 are SFU, 8-10 are SAI. Else input sel are GPIO 0-63

```

uint8_t input_type
    Timestamp input GPIO trigger or input from AUX

struct pi_timestamp_event_t

```

## WATCHDOG

*group* **WATCHDOG**

The watchdog driver allows setting a watchdog timer to reset the chip in case it gets stuck.

### Functions

**PI\_WATCHDOG\_INLINE0 uint8\_t pi\_watchdog\_timer\_set(uint32\_t us)**

Set watchdog timer.

This function sets the watchdog timer (in micro seconds). Clock is the refclk.

---

**Note:** The limit of the timer depends on the ref clock value.

---

### Parameters

- **us** – Timer in micro seconds

**Returns 0** if timer has been set

**Returns 1** if timer value is bigger than the limit

**PI\_WATCHDOG\_INLINE0 void pi\_watchdog\_start(void)**

Start watchdog timer.

This function start the countdown of the timer

**PI\_WATCHDOG\_INLINE0 void pi\_watchdog\_timer\_rearm(void)**

Restart watchdog timer.

This function rearm (at the set us value) the countdown of the timer. You have to start it again to launch the countdown.

**PI\_WATCHDOG\_INLINE0 void pi\_watchdog\_stop(void)**

Stop watchdog timer.

This function stops the countdown of the timer.

## XIP

*group* **XIP**

## Defines

**PI\_INLINE\_XIP0**  
PI\_INLINE\_XIP0.

## TypeDefs

typedef struct *pi\_xip\_conf\_s* **pi\_xip\_conf\_t**  
typedef struct *pi\_xip\_data\_err\_s* **pi\_xip\_data\_err\_t**

## Functions

PI\_INLINE\_XIP0 int32\_t **pi\_xip\_mount**(pi\_device\_t \*mount, void \*virt\_addr, uint32\_t ext\_addr, uint32\_t mount\_size, uint8\_t cacheable)

Mount an area in XIP virtual memory.

Allows to mount an area from external memory, starting at external address **ext\_addr**, of size **mount\_size** \* **page\_size** into the XIP virtual memory starting at **virt\_addr**.

### Parameters

- **mount** – A pi device containing the mount configuration.
- **virt\_addr** – An address contained in XIP area where mount will happen.
- **ext\_addr** – An external address from which the mount will happen.
- **mount\_size** – A mount size, expressed in number of pages.
- **cacheable** – Whether I-Cache should be aware of the pages used by this mount.

**Returns 0** Success

**Returns <errno>** Error code

PI\_INLINE\_XIP0 void **pi\_xip\_unmount**(pi\_device\_t \*mount)

Unmount previously mounted XIP area.

**Warning:** Should only be used from L2 code.

### Parameters

- **mount** – A pi device containing the mount configuration.

PI\_INLINE\_XIP0 int32\_t **pi\_xip\_free\_page\_mask\_get**(uint32\_t \*page\_mask, uint8\_t nb\_pages)

Acquire previously allocated XIP cache pages.

### Parameters

- **page\_mask** – A pointer to a page mask to be filled.
- **nb\_pages** – Desired number of pages.

**Returns 0** Success

**Returns > 0** Number of free pages left (if not enough are available)

---

PI\_INLINE\_XIP0 int32\_t **pi\_xip\_dcache\_page\_alloc**(uint32\_t page\_id, xip\_page\_size\_e page\_size)  
Allocate chunks of L2 memory for XIP pages.

**Parameters**

- **page\_id** – Id of the hardware page to be allocated.
- **page\_size** – Size of the page to be allocated.

**Returns 0** Success**Returns <errno>** Error code

PI\_INLINE\_XIP0 int32\_t **pi\_xip\_dcache\_page\_free**(uint32\_t page\_id, xip\_page\_size\_e page\_size)  
Release L2 page used by XIP.

**Parameters**

- **page\_id** – Id of the hardware page to be freed.
- **page\_size** – Size of the page to be freed.

**Returns 0** Success**Returns <errno>** Error code

```
struct pi_xip_conf_s
```

#include <xip.h> XIP configuration structure.

This structure is used to pass the desired XIP configuration to the runtime when mounting a memory area.

**Public Members**

uint8\_t **ro**

whether this mount is read only. Write accesses will trigger an exception.

uint8\_t **per\_id**

Peripheral / mem controller on which to mount.

uint8\_t **tlb\_en**

Whether TLB is enabled.

uint16\_t **page\_mask**

XIP cache pages to be used for this mount.

xip\_page\_size\_e **page\_size**

Desired page size, starts at 512, computes as  $512 \ll \text{page\_size}$ . Maximum is 64KB

*func\_t* **data\_exception\_handler**

Exception handler for fc data accesses.

*func\_t* **instr\_exception\_handler**

Exception handler for fc insn accesses.

*func\_t* **cl\_irq\_handler**

Exception handler for cluster accesses.

```
struct pi_xip_data_err_s
```

#include <xip.h> Structure which will be used as argument for user handler. It contains exception info.

This structure can be used by user exception handler to handle faults

### Public Members

```
uint32_t error_addr
```

Address at which the error happened.

```
uint32_t violation
```

Type of error.

### Chips

TODO

### Platforms

#### GVSOC

The GVSOC API provides some features specific to GVSOC through semi-hosting to ease the debug and profiling of the simulated software.

##### Engine control

This provides a set of functions for controlling GVSOC execution.

*group* **ENGINE**

##### Functions

```
static inline void gv_stop()
```

Stop execution.

This function can be called to stop the GVSOC engine. In case a proxy is connected, it will be notified that the engine has been stopped.

##### Trace control

This provides a set of functions for controlling system traces from the simulated software.

*group* **TRACES**

## Functions

static inline void **gv\_trace\_enable**(char \*path)  
Enable GVSOC traces.

This function can be called to dynamically enable GVSOC traces from the simulated SW. A regular expression can be passed to specify what should be enabled, as with option trace on the command-line.

### Parameters

- **path** – A regular expression specifying the path of the traces to enable.

static inline void **gv\_trace\_disable**(char \*path)  
Disable GVSOC traces.

This function can be called to dynamically disable GVSOC traces from the simulated SW. A regular expression can be passed to specify what should be disabled, as with option trace on the command-line.

### Parameters

- **path** – A regular expression specifying the path of the traces to disable.

## PCER control

This provides a set of functions for reading and writing core performance counters. These counters are usually accessed with CSR instructions but this API can also be used to get more convenient ways of accessing them, like for example dumping them all to a file.

*group* **PERF**

## Functions

static inline void **gv\_pcer\_conf**(unsigned int events)  
Configure performance counters.

This function can be called to dynamically enable or disable performance counters. Note that this is done through semi-hosting, and can also be done through a CSR instruction.

### Parameters

- **events** – A bitfields specifying which counter should be enabled. There is one bit per counter, bit 0 is for counter 0.

static inline void **gv\_pcer\_reset()**  
Reset all performance counters.

This function can be called to dynamically set all performance counters to 0. Note that this is done through semi-hosting, and can also be done through a CSR instruction.

static inline void **gv\_pcer\_start()**  
Enable performance counting.

This function can be called to dynamically enable performance counting, which means all enabled performance counters, will register the events they are monitoring. Note that this is done through semi-hosting, and can also be done through a CSR instruction.

static inline void **gv\_pcer\_stop()**  
Disable performance counting.

This function can be called to dynamically enable performance counting, which means all enabled performance counters, will register the events they are monitoring. Note that this is done through semi-hosting, and can also be done through a CSR instruction.

```
static inline unsigned int gv_pcer_read(int pcer)  
    Read a performance counter.
```

This function can be called to get the current value of a performance counter. Note that this is done through semi-hosting, and can also be done through a CSR instruction.

#### Parameters

- **pcer** – The performance counter index to be read.

**Returns** The performance counter value.

```
static inline int gv_pcer_dump(const char *path, const char *mode)  
    Dump performance counters to a file.
```

This function can be called to get the current value of all performance counters to a file.

#### Parameters

- **path** – The path of the file where the performance counters should be dumped.
- **mode** – The string specifying the mode used to open the file (directly passed to fopen)

**Returns** 0 if the operation was successful, otherwise the error code.

```
static inline void gv_pcer_dump_start()  
    Start performance counting .
```

This function can be called to do all the required steps to start the performance counters (configure all events, reset and start).

```
static inline void gv_pcer_dump_end()  
    Stop performance counting and dump to file.
```

This function can be called to do all the required steps to stop the performance counters and dump their values to a file (pcer.log).

## VCD control

This provides a set of functions for controlling VCD traces from the simulated software, in particular to modify VCD traces value in order to give some context to the VCD view.

*group* **VCD**

### Functions

```
static inline void gv_vcd_configure(int active, gv_vcd_conf_t *conf)  
    Configure VCD traces.
```

This function can be called to modify global VCD configuration.

#### Parameters

- **pcer** – 1 if VCD tracing must be enabled, 0 if it must be disabled.
- **conf** – VCD configuration, should be NULL for now.

---

```
static inline int gv_vcd_open_trace(char *path)
    Open a VCD trace.
```

This function can be called to open a VCD trace from its full path and get a descriptor which can be used for other API functions. The trace must have been created either by a model or by user traces.

#### Parameters

- **path** – The full path of the trace.

**Returns** The trace descriptor which can be used to change the value of the trace.

```
static inline void gv_vcd_dump_trace(int trace, unsigned int value)
    Change VCD trace value.
```

This function can be called to give the new value of the specified trace.

#### Parameters

- **trace** – The trace descriptor returned when the trace was opened.
- **value** – The new trace value.

```
static inline void gv_vcd_release_trace(int trace)
    Release a VCD trace.
```

This function can be called to give the special z value (high impedance).

#### Parameters

- **trace** – The trace descriptor returned when the trace was opened.

```
static inline void gv_vcd_dump_trace_string(int trace, char *str)
    Change VCD trace value for a string trace.
```

This function can be called to give the new string value of the specified trace.

#### Parameters

- **trace** – The trace descriptor returned when the trace was opened.
- **str** – The new trace string.

```
struct gv_vcd_conf_t
```

## Remote control

```
class gvsoc_control.Proxy(host: str = 'localhost', port: int = 42951)
    A class used to control GVSOC through the socket proxy
```

#### Parameters

- **host** – str, a string giving the hostname where the proxy is running
- **port** – int, the port where to connect

```
close()
```

Close the proxy.

This will free resources and close threads so that simulation can properly exit.

```
event_add(event: str)
    Enable an event.
```

**Parameters** **event** – A regular expression used to enable events

**event\_remove**(*event: str*)  
Disable a trace.

**Parameters** **event** – A regular expression used to enable events

**quit**(*status: int = 0*)  
Exit simulation.

**Parameters** **status** – Specify the status value.

**register\_exit\_callback**(*callback, \*kargs, \*\*kwargs*)  
Register exit callback

The callback is called when GVSOC exits. If no callback is registered, os.\_exit is called when GVSOC exits.

**Parameters**

- **callback** – The function to be called when GVSOC exits
- **kargs** – Arguments propagated to the callback
- **kwargs** – Arguments propagated to the callback

**run**(*duration: Optional[int] = None*)  
Starts execution.

**Parameters** **duration** – Specify the duration of the execution in picoseconds (will execute forever by default)

**stop()**  
Stop execution.

**trace\_add**(*trace: str*)  
Enable a trace.

**Parameters** **trace** – A regular expression used to enable traces

**trace\_level**(*level: str*)  
Changes the trace level.

**Parameters** **level** – The trace level, can be “error”, “warning”, “info”, “debug” or “trace”

**trace\_remove**(*trace: str*)  
Disable a trace.

**Parameters** **trace** – A regular expression used to disable traces

**wait\_running()**  
Wait until GVSOC is running.

This will block the caller until gvsoc starts execution.

**wait\_stop()**  
Wait until execution stops.

This will block the caller until gvsoc stops execution.

**class gvsoc\_control.Router**(*proxy: gvsoc\_control.Proxy, path: str = '\*\*/chip/soc/axi\_ico'*)  
A class used to inject memory accesses into a router

**Parameters**

- **proxy** – The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **path** – The path to the router in the architecture.

**mem\_read(addr: int, size: int) → bytes**

Inject a memory read.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

#### Parameters

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.

**Returns** bytes, The sequence of bytes read, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_read\_int(addr: int, size: int) → int**

Read an integer.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

#### Parameters

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.

**Returns** int, The integer read.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_write(addr: int, size: int, values: bytes)**

Inject a memory write.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

#### Parameters

- **addr** – The address of the access.
- **size** – The size of the access in bytes.
- **values** – The sequence of bytes to be written, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**mem\_write\_int(addr: int, size: int, value: int)**

Write an integer.

The access is generated by the router where this class is connected and is injected as a debug request to not disturb the timing.

#### Parameters

- **addr** – int, The address of the access.
- **size** – int, The size of the access in bytes.
- **value** – int, The integer to be written.

**Raises** RuntimeError, if the access generates an error in the architecture.

**class gvsoc\_control.Testbench(proxy: gvsoc\_control.Proxy, path: str = '\*\*/testbench/testbench')**

Testbench class.

This class can be instantiated to get access to the testbench.

**Parameters**

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **path** – string, optional, The path to the testbench in the architecture.

**i2s\_get**(*id*: int = 0)

Open an SAI.

Open an SAI and return an object which can be used to interact with it.

**Parameters** **id** – int, optional, The SAI identifier.**Returns** Testbench\_i2s, An object which can be used to access the specified SAI.**uart\_get**(*id*: int = 0)

Open a uart interface.

Open a uart interface and return an object which can be used to interact with it.

**Parameters** **id** – int, optional, The uart interface identifier.**Returns** Testbench\_uart, An object which can be used to access the specified uart interface.**class gvsoc\_control.Testbench\_i2s(proxy: gvsoc\_control.Proxy, testbench: gvsoc\_control.Testbench, id=0)**  
Class instantiated for each manipulated SAI.

It can be used to interact with the SAI, like injecting streams.

**Parameters**

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **testbench** – int, The testbench object.
- **id** – int, optional, The identifier of the SAI interface.

**clk\_start()**

Start clock.

This can be used when the clock is generated by the testbench to start the generation.

**Raises** RuntimeError, if there is any error while starting the clock.**clk\_stop()**

Stop clock.

This can be used when the clock is generated by the testbench to stop the generation.

**Raises** RuntimeError, if there is any error while stopping the clock.**close()**

Close SAI.

**Raises** RuntimeError, if there is any error while closing.**open(word\_size: int = 16, sampling\_freq: int = -1, nb\_slots: int = 1, is\_pdm: bool = False, is\_full\_duplex: bool = False, is\_ext\_clk: bool = False, is\_ext\_ws: bool = False, is\_sai0\_clk: bool = False, is\_sai0\_ws: bool = False, clk\_polarity: int = 0, ws\_polarity: int = 0, ws\_delay: int = 1)**  
Open and configure SAI.**Parameters**

- **word\_size** – int, optional, Specify the frame word size in bits.

- **sampling\_freq** – int, optional, Specify the sampling frequency. This is used either to generate the clock when it is external or to check that internally generated one is correct.
- **nb\_slots** – int, optional, Number of slots in the frame.
- **is\_pdm** – bool, optional, True if the stream is a PDM stream.
- **is\_full\_duplex** – bool, optional, True if the SAI is used in full duplex mode.
- **is\_ext\_clk** – bool, optional, True is the clock is generated by the testbench.
- **is\_ext\_ws** – bool, optional, True is the word strobe is generated by the testbench.
- **is\_sai0\_clk** – bool, optional, True is the the clock should be taken from SAI0.
- **is\_sai0\_ws** – bool, optional, True is the word strobe should be taken from SAI0.
- **clk\_polarity** – int, optional, Clock polarity, definition is the same as SAI0 specifications.
- **ws\_polarity** – int, optional, Word strobe polarity, definition is the same as SAI0 specifications.
- **ws\_delay** – int, optional, Word strobe delay, definition is the same as SAI0 specifications.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_close(slot: int = 0)**

Close a slot.

**Parameters** **slot** – int, optional, Slot identifier

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_open(slot: int = 0, is\_rx: bool = True, word\_size: int = 16, is\_msb: bool = True, sign\_extend: bool = False, left\_align: bool = False)**

Open and configure a slot.

**Parameters**

- **slot** – int, optional, Slot identifier
- **is\_rx** – bool, optional, True if gap receives the samples.
- **word\_size** – int, optional, Slot width in number of bits.
- **is\_msb** – bool, optional, True if the samples are received or sent with MSB first.
- **sign\_extend** – bool, optional, True if the samples are sign-extended.
- **left\_align** – bool, optional, True if the samples are left aligned.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_rx\_file\_reader(slot: Optional[int] = None, slots: list = [], filetype: str = 'wav', filepath: Optional[str] = None, encoding: str = 'asis', channel: int = 0, width: int = 0)**

Read a stream of samples from a file.

This will open a file and stream it to the SAI so that gap receives the samples. It can be used either in mono-channel mode with the slot parameter or multi-channel mode with the slots parameter. In multi-channel mode, the slots parameters give the list of slots associated to each channel. To allow empty channels, a slot of -1 can be given.

**Parameters**

- **slot** – int, optional, Slot identifier
- **slots** – list, optional, List of slots when using multi-channel mode. slot must be None if this one is not empty.

- **filetype** – string, optional, Describes the type of the file, can be “wav”, “raw”, “bin” or “au”.
- **width** – int, optional, width of the samples, in case the file is in binary format
- **filepath** – string, optional, Path to the file.
- **encoding** – string, optional, Encoding type for binary files, can be: “asis”, “plusminus”
- **channel** – int, optional, If the format supports it, this will get the samples from the specified channel in the input file.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_stop**(*slot: int = 0, stop\_rx: bool = True, stop\_tx: bool = True*)

Stop a slot.

This will stop the streamings (file reader or dumper) configured on the specified slot.

#### Parameters

- **slot** – int, optional, Slot identifier
- **stop\_rx** – bool, optional, Stop the stream sent to gap.
- **stop\_tx** – bool, optional, Stop the stream received from gap.

**Raises** RuntimeError, if there is any invalid parameter.

**slot\_tx\_file\_dumper**(*slot: Optional[int] = None, slots: list = [], filetype: str = 'wav', filepath: Optional[str] = None, encoding: str = 'asis', channel: int = 0, width: int = 0*)

Write a stream of samples to a file.

This will open a file and write to it all the samples received from gap. It can be used either in mono-channel mode with the slot parameter or multi-channel mode with the slots parameter. In multi-channel mode, the slots parameters give the list of slots associated to each channel. To allow empty channels, a slot of -1 can be given. A slot can be given several times in order to push the samples to several channels.

#### Parameters

- **slot** – int, optional, Slot identifier
- **slots** – list, optional, List of slots when using multi-channel mode. slot must be None if this one is not empty.
- **filetype** – string, optional, Describes the type of the file, can be “wav”, “raw”, “bin” or “au”.
- **encoding** – string, optional, Encoding type for binary files, can be: “asis”, “plusminus”
- **width** – int, optional, width of the samples, in case the file is in binary format
- **filepath** – string, optional, Path to the file.
- **channel** – int, optional, If the format supports it, this will dump the samples to the specified channel in the output file.

**Raises** RuntimeError, if there is any invalid parameter.

**class gvsoc\_control.Testbench\_uart**(*proxy: gvsoc\_control.Proxy, testbench: gvsoc\_control.Testbench, id=0*)

Class instantiated for each manipulated uart interface.

It can be used to interact with the uart interface, like injecting streams.

#### Parameters

- **proxy** – Proxy, The proxy object. This class will use it to send command to GVSOC through the proxy connection.
- **testbench** – int, The testbench object.
- **id** – int, optional, The identifier of the uart interface.

**close()**

Close the uart interface.

**Raises** RuntimeError, if there is any error while closing.

**open(baudrate: int, word\_size: int = 8, stop\_bits: int = 1, parity\_mode: bool = False, ctrl\_flow: bool = True, is\_usart: bool = False, usart\_polarity: int = 0, usart\_phase: int = 0)**

Open and configure a uart interface.

**Parameters**

- **baudrate** – int, Specify the uart baudrate in bps
- **word\_size** – int, optional, Specify the size in bits of the uart bytes.
- **stop\_bits** – int, optional, Specify the number of stop bits.
- **parity\_mode** – bool, optional, True if parity is enabled.
- **ctrl\_flow** – bool, optional, True if control flow is enabled.
- **is\_usart** – bool, optional, True if uart is in usart mode.
- **usart\_polarity** – int, optional, Usart polarity.
- **usart\_phase** – int, optional, Usart phase.

**Raises** RuntimeError, if there is any invalid parameter.

**rx(size=None)**

Read data from the uart.

Once reception on the uart is enabled, the received bytes are pushed to a fifo. This method can be called to pop received bytes from the FIFO.

**Parameters** **size** – int, The number of bytes to be read. If it is None, it returns the bytes which has already been received.

**Returns** bytes, The sequence of bytes received, in little endian byte ordering.

**Raises** RuntimeError, If the access generates an error in the architecture.

**rx\_attach\_callback(callback, \*kargs, \*\*kwargs)**

Attach callback for receiving bytes from the uart.

All bytes received from the uart now triggers the execution of the specified callback. This must be called only when uart reception is disabled. The callback will be called asynchronously by a different thread and so special care must be taken to access shared variables using locks. Also the proxy can not be used from the callback.

**Parameters**

- **callback** – The function to be called when bytes are received from the uart. First parameters will contain number of bytes and received, and second one will be the bytes received.
- **kargs** – Arguments propagated to the callback
- **kwargs** – Arguments propagated to the callback

**Returns** bytes, The sequence of bytes received, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_detach\_callback()**

Detach a callback.

The callback previously attached won't be called anymore. This must be called only when uart reception is disabled.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_disable()**

Disable receiving bytes from the uart.

**Raises** RuntimeError, if the access generates an error in the architecture.

**rx\_enable()**

Enable receiving bytes from the uart.

Any byte received from the uart either triggers the callback execution if it has been registered, or is pushed to a FIFO which can read.

**Raises** RuntimeError, if the access generates an error in the architecture.

**tx(*values: bytes*)**

Send data to the uart.

This enqueues an array of bytes to be transmitted. If previous transfers are not finished, these bytes will be transferred after.

**Parameters** **values** – bytes, The sequence of bytes to be sent, in little endian byte ordering.

**Raises** RuntimeError, if the access generates an error in the architecture.

## 7.2 PMSIS BSP

### 7.2.1 Introduction

The PMSIS BSP is a set of high level-drivers written on top of the PMSIS API, which makes them available on any operating system which implements the PMSIS API.

### 7.2.2 Conventions

All functions prefixed by `pi_` can only be called from fabric-controller side while the ones prefixed by `pi_cl_` can only be called from cluster side. Any exception to these rules is documented where it applies.

All functions on fabric-controller side are by default synchronous and are blocking the caller until the operation is done. All the functions suffixed by `_async` are asynchronous and are not blocking the caller. The termination of such operations is managed with a `pi_evt_t` object, see PMSIS API documentation for more information.

Functions on cluster-side are by default synchronous but can also be asynchronous if the documentation of the function mentions it.

### 7.2.3 Chip-specific information

#### GAP8

All functions transferring data between an external device and a chip memory must use the L2 memory for the chip memory.

### 7.2.4 Drivers

#### ADC

##### ADS1014

*group* **ADS1014**

TI ADS1014 Analog-To-Digital-Converter.

**Warning:** Support for comparator IRQ handling is not implemented.

#### Enums

enum **ads1014\_pga**

ADS1014 PGA (Programmable Gain Amplifier) values

It is the range of the measured value. The absolute value of the measured voltage will never go above the power supply voltage value.

*Values:*

```
enumerator ADS1014_PGA_FSR_6V144 = 0x0
enumerator ADS1014_PGA_FSR_4V096 = 0x1
enumerator ADS1014_PGA_FSR_2V048 = 0x2
enumerator ADS1014_PGA_FSR_1V024 = 0x3
enumerator ADS1014_PGA_FSR_0V512 = 0x4
enumerator ADS1014_PGA_FSR_0V256 = 0x5
```

enum **ads1014\_operating\_mode**

ADS1014 operating mode values

*Values:*

```
enumerator ADS1014_OPERATING_MODE_CONTINUOUS = 0x0
The ADC measures continuously.
```

```
enumerator ADS1014_OPERATING_MODE_SINGLE_SHOT = 0x1
The ADC only measures once, and goes back to power-saving mode
```

**enum ads1014\_data\_rate**

ADS1014 Sampling rate (samples per second)

*Values:*

enumerator **ADS1014\_DATA\_RATE\_SPS\_128** = 0x0  
enumerator **ADS1014\_DATA\_RATE\_SPS\_250** = 0x1  
enumerator **ADS1014\_DATA\_RATE\_SPS\_490** = 0x2  
enumerator **ADS1014\_DATA\_RATE\_SPS\_920** = 0x3  
enumerator **ADS1014\_DATA\_RATE\_SPS\_1600** = 0x4  
enumerator **ADS1014\_DATA\_RATE\_SPS\_2400** = 0x5  
enumerator **ADS1014\_DATA\_RATE\_SPS\_3300** = 0x6

**enum ads1014\_comparator\_mode**

ADS1014 Comparator mode

*Values:*

enumerator **ADS1014\_COMPARATOR\_MODE\_TRADITIONAL** = 0x0

The comparator triggers when the measured value goes above the high threshold, and resets when the value goes below the low threshold.

enumerator **ADS1014\_COMPARATOR\_MODE\_WINDOW** = 0x1

The comparator triggers if the measured value goes outside the window, i.e. above the high threshold or below the low threshold.

**enum ads1014\_comparator\_polarity**

ADS1014 Alert/Ready comparator pin polarity

Controls the ADC Alert/Ready pin active polarity

*Values:*

enumerator **ADS1014\_COMPARATOR\_POLARITY\_ACTIVE\_LOW** = 0x0

enumerator **ADS1014\_COMPARATOR\_POLARITY\_ACTIVE\_HIGH** = 0x1

**enum ads1014\_comparator\_latch**

ADS1014 comparator latching mode

Determines whether the comparator latches after triggering. When the comparator is set to latch, it will only be cleared by reading the ADC measured value.

*Values:*

enumerator **ADS1014\_COMPARATOR\_LATCH\_DISABLED** = 0x0

enumerator **ADS1014\_COMPARATOR\_LATCH\_ENABLED** = 0x1

**enum ads1014\_comparator\_status**

ADS1014 comparator status

Determines after how many out-of-bounds conversions the comparator will trigger.

*Values:*

enumerator **ADS1014\_COMPARATOR\_STATUS\_ASSERT\_ONE** = 0x0  
triggers after 1 conversion

enumerator **ADS1014\_COMPARATOR\_STATUS\_ASSERT\_TWO** = 0x1  
triggers after 2 out-of-bounds conversions

enumerator **ADS1014\_COMPARATOR\_STATUS\_ASSERT\_THREE** = 0x2  
triggers after 3 out-of-bounds conversions

enumerator **ADS1014\_COMPARATOR\_STATUS\_DISABLED** = 0x3  
the comparator is disabled

## Functions

**void pi\_ads1014\_conf\_init(struct *pi\_ads1014\_conf* \*conf)**  
Initialize an ADS1014 configuration with default values.

The structure containing the configuration must be kept alive until the device is opened. It can only be called from fabric-controller side.

### Parameters

- **conf** – [inout] Pointer to the device configuration.

**int pi\_ads1014\_open(pi\_device\_t \*device)**  
Open a ADS1014 device

### Parameters

- **device** – [inout] pointer to the ADS1014 device

**Returns** PI\_OK if operation was sucessful, an error code otherwise

**void pi\_ads1014\_close(pi\_device\_t \*device)**  
Close a ADS1014 device

### Parameters

- **device** – [inout] pointer to the ADS1014 device

**int pi\_ads1014\_read(pi\_device\_t \*device, float \*value)**  
Read the value measured by the ADC.

### Parameters

- **device** – [in] pointer to the ads1014 device
- **value** – [out] value in mV returned by the ADC

**Returns** PI\_OK if operation was successful, an error code otherwise

**int pi\_ads1014\_set\_comparator\_thresholds(pi\_device\_t \*device, float threshold\_low, float threshold\_high)**  
Set the comparator thresholds (low and high)

### Parameters

- **device** – [in] pointer to the ads1014 device
- **threshold\_low** – [in] new value for comparator low threshold (in mV)
- **threshold\_high** – [in] new value for comparator high threshold (in mV)

**Returns** PI\_OK if operation was successful, an error code otherwise

```
struct pi_ads1014_conf
```

#### Public Members

`uint8_t i2c_if`

I2C interface which is connected to the ADC

`uint8_t i2c_addr`

Address of the ADC

`enum ads1014_operating_mode operating_mode`

ADC operating mode (single or continous)

`enum ads1014_pga pga`

range of the measured value

`enum ads1014_data_rate data_rate`

sampling rate

`enum ads1014_comparator_status comparator_status`

ADC comparator status (enabled&trigger conditions, or disabled)

`enum ads1014_comparator_mode comparator_mode`

ADC comparator mode (traditional or window)

`enum ads1014_comparator_latch comparator_latch`

ADC comparator latch setting

`enum ads1014_comparator_polarity comparator_polarity`

ADC comparator triggered polarity

## BLE

### Common API

*group* **BLE**

The BLE driver provides support for wireless data transfer between the host and an external peripheral such as a smartphone, using Bluetooth connection.

## Typedefs

```
typedef struct pi_ble_api_s pi_ble_api_t
BLE specific API.

Structure holding BLE specific API.
```

## Functions

`int32_t pi_ble_open(struct pi_device *device)`  
Open a BLE device.

This function opens and initializes a BLE device. This function must be called before using device.

### Parameters

- **device** – Pointer to the BLE device structure.

**Returns 0** If the operation is successful.

**Returns ERRNO** Error code otherwise.

`void pi_ble_close(struct pi_device *device)`  
Close a BLE device.

This function closes an opened BLE device. It frees all allocated objects and interfaces used.

### Parameters

- **device** – Pointer to the BLE device structure.

`int32_t pi_ble_ioctl(struct pi_device *device, uint32_t cmd, void *arg)`  
IOctl commands.

This function is used to send special command to BLE device.

### Parameters

- **device** – Pointer to the BLE device structure.
- **cmd** – Ioctl command.
- **arg** – Ioctl command arg.

**Returns** Value Value depends on ioctl command.

`int32_t pi_ble_at_cmd(struct pi_device *device, const char *cmd, char *resp, uint32_t size)`  
Send AT command to BLE device.

This function is used to send AT commands to BLE device.

---

**Note:** The command string should be plain command, i.e. without the “AT” part.

---



---

**Note:** The command may or may not return a response.

---

### Parameters

- **device** – Pointer to the BLE device structure.
- **cmd** – Command string to send.

- **resp** – Buffer to store response.
- **size** – Size of the response to store, in Bytes.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

int32\_t **pi\_ble\_peer\_connect**(struct pi\_device \*device, const char \*addr)

Connect to a peer.

This function should be used to connect to a remote peer.

#### Parameters

- **device** – Pointer to the BLE device structure.
- **addr** – Address in string format.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

int32\_t **pi\_ble\_peer\_disconnect**(struct pi\_device \*device, const char \*addr)

Close a connection to a peer.

This function closes an existing peer connection.

#### Parameters

- **device** – Pointer to the BLE device structure.
- **addr** – Address in string format.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

void **pi\_ble\_data\_send**(struct pi\_device \*device, uint8\_t \*buffer, uint32\_t size)

Send data to BLE device. Blocking API.

This function is used to send data to BLE device.

---

**Note:** This function is synchronous, caller is blocked until transfer is finished. The pending asynchronous function is below : [\*pi\\_ble\\_data\\_send\\_async\(\)\*](#).

---

#### Parameters

- **device** – Pointer to the BLE structure.
- **buffer** – Buffer to send.
- **size** – Size of data to send.

void **pi\_ble\_data\_send\_async**(struct pi\_device \*device, uint8\_t \*buffer, uint32\_t size, pi\_evt\_t \*task)

Send data to BLE device. Blocking API.

This function is used to send data to BLE device.

---

**Note:** This function is asynchronous. The pending synchronous function is : [\*pi\\_ble\\_data\\_send\(\)\*](#).

---

#### Parameters

- **device** – Pointer to the BLE structure.
- **buffer** – Buffer to send.
- **size** – Size of data to send.
- **task** – Event task used to check end of transfer.

`void pi_ble_data_get(struct pi_device *device, uint8_t *buffer, uint32_t size)`  
Get data from BLE device. Blocking API.

This function is used to retrieve data from BLE device.

---

**Note:** This function is synchronous, caller is blocked until transfer is finished. The pending asynchronous function is below : [\*pi\\_ble\\_data\\_get\\_async\(\)\*](#).

---

#### Parameters

- **device** – Pointer to the BLE structure.
- **buffer** – Buffer to store data.
- **size** – Size of data.

`void pi_ble_data_get_async(struct pi_device *device, uint8_t *buffer, uint32_t size, pi_evt_t *task)`  
Get data from BLE device. Non blocking API.

This function is used to retrieve data from BLE device.

---

**Note:** This function is asynchronous. The pending synchronous function is : [\*pi\\_ble\\_data\\_get\(\)\*](#).

---

#### Parameters

- **device** – Pointer to the BLE structure.
- **buffer** – Buffer to store data.
- **size** – Size of data.
- **task** – Event task used to check end of transfer.

`int8_t pi_ble_catch_peer_event_async(struct pi_device *device, char *resp, pi_evt_t *callback)`  
Manage events received from peer.

---

**Note:** Asynchronous.

---

#### Parameters

- **device** –
- **resp** – NOT NULL. Buffer to store the event.
- **callback** – NOT NULL. Triggered when we received an event.

**Returns 0** if success

**Returns -1** if error

**Returns** int8\_t Process status code.

```
struct pi_ble_api_s
    #include <ble.h> BLE specific API.
    Structure holding BLE specific API.
```

### Public Members

int32\_t (\***at\_cmd**)(struct pi\_device \*device, const char \*cmd, char \*resp, uint32\_t size)  
Function to send AT command.

int32\_t (\***peer\_connect**)(struct pi\_device \*device, const char \*addr)  
Function to connect BLE device to a remote peer.

int32\_t (\***peer\_disconnect**)(struct pi\_device \*device, const char \*addr)  
Function to disconnect BLE device to a remote peer.

## Nina B312

### group **NINA\_B312**

The nina\_b312 driver provides support for data transfer using a BLE module, here a NINA B312 BLE module. This module is interfaced on GAP9\_EVK through UART.

### Defines

```
PI_AT_RESP_ARRAY_LENGTH
    RESP array length.
```

### Enums

```
enum pi_ble_ioctl_cmd_e
    Values:
```

enumerator **PI\_NINA\_B112\_MODEL\_INFO** = 0  
BLE device info.

enumerator **PI\_NINA\_B112\_SERVER\_CONFIGURE**  
Configure BLE device as a server.

enumerator **PI\_NINA\_B112\_CLIENT\_CONFIGURE**  
Configure BLE device as a client.

enumerator **PI\_NINA\_B112\_UART\_CONFIGURE**  
Configure HCI UART.

---

enumerator **PI\_NINA\_B112\_DATA\_MODE\_ENTER**  
Enter Data Mode.

enumerator **PI\_NINA\_B112\_DATA\_MODE\_EXIT**  
Exit Data Mode.

enumerator **PI\_NINA\_B112\_WAIT\_FOR\_EVENT**  
Wait for some responses/events from BLE device.

enumerator **PI\_NINA\_B312\_MODEL\_INFO = 0**  
BLE device info.

enumerator **PI\_NINA\_B312\_SERVER\_CONFIGURE**  
Configure BLE device as a server.

enumerator **PI\_NINA\_B312\_CLIENT\_CONFIGURE**  
Configure BLE device as a client.

enumerator **PI\_NINA\_B312\_UART\_CONFIGURE**  
Configure HCI UART.

enumerator **PI\_NINA\_B312\_DATA\_MODE\_ENTER**  
Enter Data Mode.

enumerator **PI\_NINA\_B312\_DATA\_MODE\_EXIT**  
Exit Data Mode.

enumerator **PI\_NINA\_B312\_CATCH\_PEER\_EVENT**  
Catch events received from peer.

## Functions

void **pi\_ble\_nina\_b312\_conf\_init**(struct pi\_device \*device, struct *pi\_nina\_b312\_conf* \*conf)  
Initialize NINA\_B312 configuration structure.

### Parameters

- **device** – Pointer to the BLE device structure.
- **conf** – Pointer to NINA\_B312 configuration structure.

struct **pi\_nina\_b312\_conf**  
*#include <nina\_b312.h>* NINA\_B312 configuration structure.

This structure holds BLE configuration(interface used, baudrate,...).

**Public Members**

**uint8\_t uart\_if**  
UART interface used to connect BLE device.

**uint32\_t baudrate**  
UART baudrate.

**char local\_name[30]**  
BLE device name (visible by others).

**Camera****Common API***group Camera*

The camera driver provides support for capture data from an external camera and getting the data into the chip running this driver.

**Enums**

**enum pi\_camera\_cmd\_e**  
Command ID for pi\_camera\_control.

*Values:*

**enumerator PI\_CAMERA\_CMD\_PWM\_CLK**  
Clock into the camera.

**enumerator PI\_CAMERA\_CMD\_ON**  
Power-up the camera.

**enumerator PI\_CAMERA\_CMD\_OFF**  
Power-off the camera.

**enumerator PI\_CAMERA\_CMD\_START**  
Start the camera, i.e. it will start sending data on the interface.

**enumerator PI\_CAMERA\_CMD\_STOP**  
Stop the camera, i.e. it will stop sending data on the interface.

**enumerator PI\_CAMERA\_CMD\_CONTINUE\_MODE**  
Switch to continue mode in the camera.

**enumerator PI\_CAMERA\_CMD\_TRIGGER\_MODE**  
Switch to trigger mode in the camera.

**enumerator PI\_CAMERA\_CMD\_POWERDOWN\_MODE**  
Power-down the camera.

enumerator **PI\_CAMERA\_CMD\_AEG\_INIT**  
Enable and init the camera's AEG (Automatic Exposure and Gain).

enumerator **PI\_CAMERA\_CMD\_SNAPSHOT**  
snapshot mode.

enum **pi\_camera\_opts\_e**  
Camera command options.

This is the command ID option passed as third argument to pi\_camera\_control.

*Values:*

enumerator **PI\_CAMERA\_NO\_OPT**  
No option (for back compatibility).

enumerator **PI\_CAMERA\_OPT\_NO\_REG\_INIT**  
Do not initialize camera register, nor reset it.

enum **pi\_camera\_format\_e**  
Camera format.

This can be given within the camera configuration when it is opened.

*Values:*

enumerator **PI\_CAMERA\_HD1080**  
HD 1080 format (1920 x 1080)

enumerator **PI\_CAMERA\_HD720**  
HD 720 format (1280 x 720).

enumerator **PI\_CAMERA\_WXGA**  
WXGA format (1280 x 800).

enumerator **PI\_CAMERA\_QWXGA**  
QWXGA format (640 x 400).

enumerator **PI\_CAMERA\_VGA**  
VGA format (640 x 480).

enumerator **PI\_CAMERA\_QVGA**  
QVGA format (320 x 240).

enumerator **PI\_CAMERA\_QQVGA**  
QQVGA format (160 x 120).

enum **pi\_camera\_color\_mode\_e**  
Camera color mode.

This can be given within the camera configuration when it is opened.

*Values:*

enumerator **PI\_CAMERA\_GRAY8**  
8 bit grayscale.

enumerator **PI\_CAMERA\_RGB565**  
16 bit RGB .

enumerator **PI\_CAMERA\_RGB888**  
24 bit RGB .

enumerator **PI\_CAMERA\_YUV**  
24 bit YUV .

## Functions

`int32_t pi_camera_open(struct pi_device *device)`  
Open an image sensor device.

This function must be called before the Camera device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

### Parameters

- **device** – A pointer to the device structure of the device to open.

**Returns** 0 if the operation is successfull, -1 if there was an error.

`static inline int32_t pi_camera_control(struct pi_device *device, pi_camera_cmd_e cmd, void *arg)`  
Control the Camera device.

This function is used to control and configure the Camera device. For each command, the arguments necessary are listed below:

CMD	Type of argument
CMD_ON	NULL
CMD_OFF	NULL
CMD_START	NULL
CMD_STOP	NULL

### Parameters

- **device** – The device structure of the device to control.
- **cmd** – The command for controlling or configuring the camera. Check the description of `pi_camera_cmd_e` for further information.
- **\*arg** – A pointer to the arguments of the command.

**Returns** 0 if the operation is successfull, -1 if there was an error.

`void pi_camera_capture(struct pi_device *device, void *buffer, uint32_t size)`  
Capture a sequence of samples.

Queue a buffer that will receive Camera samples. The samples will start being stored in the provided buffer as soon as the camera is started. If it is already started, it starts storing them immediately. On some chips, the start of the sampling may be differed to the next start of frame, see chip-specific section for more details. The caller is blocked until the transfer is finished.

## Parameters

- **device** – The device structure of the device where to capture samples.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **size** – The size in bytes of the memory buffer.

```
static inline void pi_camera_capture_async(struct pi_device *device, void *buffer, uint32_t size, pi_evt_t *task)
```

Capture a sequence of samples.

Queue a buffer that will receive Camera samples. The samples will start being stored in the provided buffer as soon as the camera is started. If it is already started, it starts storing them immediately. On some chips, the start of the sampling may be differed to the next start of frame, see chip-specific section for more details. It is possible to call this function asynchronously and several times in order to queue several buffers. At a minimum 2 buffers should be queued to ensure that no data sampled is lost. This is also the most efficient way to retrieve data from the Camera device. You should always make sure that at least 2 buffers are always queued, by queuing a new one as soon as the current one is full. Can only be called from fabric-controller side. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

## Parameters

- **device** – The device structure of the device where to capture samples.
- **buffer** – The memory buffer where the captured samples will be transferred.
- **size** – The size in bytes of the memory buffer.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_camera_close(struct pi_device *device)
```

Close an opened Camera device.

This function can be called to close an opened Camera device once it is not needed anymore in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

## Parameters

- **device** – The device structure of the device to close.

```
static inline int32_t pi_camera_reg_set(struct pi_device *device, uint32_t reg_addr, uint8_t *value)
```

Set camera register.

This can be called to set a camera register. This must be used carefully as this can disturb the behavior of the other calls. The list of registers is specific to each camera, see the documentation of the camera for more information.

## Parameters

- **device** – The device structure of the camera.
- **reg\_addr** – The register address.
- **value** – A pointer to the value to be set. The size of this variable depends on the register being accessed.

```
static inline int32_t pi_camera_reg_get(struct pi_device *device, uint32_t reg_addr, uint8_t *value)
```

Get camera register.

This can be called to get a camera register. The list of registers is specific to each camera, see the documentation of the camera for more information.

### Parameters

- **device** – The device structure of the camera.
- **reg\_addr** – The register address.
- **value** – A pointer to the value where the read value will be stored. The size of this variable depends on the register being accessed.

```
struct pi_camera_slicing_conf_t
```

### Public Members

uint16\_t **x**

X coordinate of the ROI start

uint16\_t **y**

Y coordinate of the ROI start

uint16\_t **w**

Width of the ROI start

uint16\_t **h**

Height of the ROI start

uint8\_t **slice\_en**

Slice mode enable

```
struct pi_camera_slicing_conf
```

#include <camera.h> Camera slice mode configuration.

This structure is used to pass the desired ROI (region of interest) configuration to the runtime.

## Himax

group **Himax**

### Functions

```
void pi_himax_conf_init(struct pi_himax_conf *conf)
```

Initialize a camera configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened. Can only be called from fabric-controller side.

### Parameters

- **conf** – A pointer to the camera configuration.

```
void pi_ov7670_conf_init(struct pi_ov7670_conf *conf)
```

Initialize a camera configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened. Can only be called from fabric-controller side.

## Parameters

- **conf** – A pointer to the camera configuration.

```
struct pi_himax_conf
#include <himax.h> Himax configuration structure.
```

This structure is used to pass the desired Himax configuration to the runtime when opening the device.

## Public Members

```
struct pi_camera_conf camera
Generic camera configuration.
```

```
int cpi_itf
CPI interface where the camera is connected.
```

```
int i2c_itf
I2C interface where the camera control interface is connected.
```

```
char skip_pads_config
Skip pads configuration if set to 1.
```

```
pi_camera_format_e format
Camera image resolution.
```

```
pi_camera_slicing_conf_t roi
ROI (region of interest) of an image.
```

```
struct pi_ov7670_conf
#include <ov7670.h> Himax configuration structure.
```

This structure is used to pass the desired Himax configuration to the runtime when opening the device.

## Public Members

```
struct pi_camera_conf camera
Generic camera configuration.
```

```
int cpi_itf
CPI interface where the camera is connected.
```

```
int i2c_itf
I2C interface where the camera control interface is connected.
```

```
char skip_pads_config
Skip pads configuration if set to 1.
```

```
pi_camera_format_e format
Camera image resolution.
```

**Mt9v034**

group **Mt9v**

**Functions**

void **pi\_mt9v034\_conf\_init**(struct *pi\_mt9v034\_conf* \*conf)

Initialize a camera configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened. Can only be called from fabric-controller side.

**Parameters**

- **conf** – A pointer to the camera configuration.

struct **pi\_mt9v034\_conf**

#include <mt9v034.h> Mt9v034 configuration structure.

This structure is used to pass the desired mt9v034 configuration to the runtime when opening the device.

**Public Members**

struct *pi\_camera\_conf* **camera**

Generic camera configuration.

char **cpi\_itf**

CPI interface where the camera is connected.

char **i2c\_itf**

I2C interface where the camera control interface is connected.

char **power\_gpio**

GPIO number where the power pad of the camera is connected.

char **trigger\_gpio**

GPIO number where the trigger pad of the camera is connected.

char **column\_flip**

Flip columns if set to 1.

char **row\_flip**

Flip rows if set to 1.

char **skip\_pads\_config**

Skip pads configuration if set to 1.

*pi\_camera\_format\_e* **format**

Camera image resolution.

## OV5647

*group Ov5647*

### Functions

**void pi\_ov5647\_conf\_init(struct *pi\_ov5647\_conf* \*conf)**

Initialize a camera configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened. Can only be called from fabric-controller side.

#### Parameters

- **conf** – A pointer to the camera configuration.

**struct *pi\_ov5647\_conf***

#include <ov5647.h> Ov5647 configuration structure.

This structure is used to pass the desired Ov5647 configuration to the runtime when opening the device.

### Public Members

**struct *pi\_camera\_conf* *camera***

Generic camera configuration.

***pi\_camera\_format\_e* *format***

Camera image resolution.

**int *csi2\_itf***

CPI interface where the camera is connected.

**int *i2c\_itf***

I2C interface where the camera control interface is connected.

**char *vc***

support virtual channel, default is 0, can up to 2 virtual channel .

***pi\_camera\_slicing\_conf\_t* *roi***

ROI (region of interest) of an image.

## OV9281

*group Ov9281*

## Functions

void **pi\_ov9281\_conf\_init**(struct *pi\_ov9281\_conf* \*conf)

Initialize a camera configuration with default values.

The structure containing the configuration must be kept alive until the camera device is opened. Can only be called from fabric-controller side.

### Parameters

- **conf** – A pointer to the camera configuration.

struct **pi\_ov9281\_conf**

#include <ov9281.h> Ov9281 configuration structure.

This structure is used to pass the desired Ov9281 configuration to the runtime when opening the device.

## Public Members

struct *pi\_camera\_conf* **camera**

Generic camera configuration.

*pi\_camera\_format\_e* **format**

Camera image resolution.

int **csi2\_itf**

CPI interface where the camera is connected.

int **i2c\_itf**

I2C interface where the camera control interface is connected.

char **vc**

support virtual channel, default is 0, can up to 2 virtual channel .

*pi\_camera\_slicing\_conf\_t* **roi**

ROI (region of interest) of an image.

## CRC

### CRC32

This module can be enabled by defined CONFIG\_BSP\_DRIVER\_CRC\_CRC32=1 in a Makefile or by selecting the corresponding option in Kconfig.

group **CRC32**

The CRC32 module defines functions to compute CRC32 checksums.

## Functions

`uint32_t pi_crc32_compute(uint8_t *payload, uint32_t length)`

Computes a CRC32 checksum for the given payload. Uses the polynom 0x04C11DB7 and seed value 0xFFFFFFFF.

### Parameters

- **payload** – [in] payload on which the checksum should be computed
- **length** – [in] length of the payload

**Returns** the computed CRC32 checksum

## Display

### Display

#### group Display

The display driver provides support to write and print on LCD screens.

### Enums

`enum pi_display_ioctl_cmd_e`

*Values:*

enumerator `PI_DISPLAY_IOCTL_CUSTOM` = 0

## Functions

`int pi_display_open(struct pi_device *device)`

Open a display device.

This function must be called before the display device can be used. It does all the needed configuration to enable display device.

### Parameters

- **device** – A pointer to the device structure of the device to open.

**Returns** 0 Device opened successfully. error code otherwise.

`void pi_display_write(struct pi_device *device, pi_buffer_t *buffer, uint16_t x, uint16_t y, uint16_t w, uint16_t h)`

Write on a display device. Blocking API.

This function is called to print/write a buffer on a display device. It displays a whole frame of (width \* height) size on display at position (x, y).

---

**Note:** This function is synchronous, caller can not do anything else until transfer is finished. The pending asynchronous function is below : `pi_display_write_async()`.

### Parameters

- **device** – A pointer to the device structure of the device to open.
- **buffer** – Data buffer to write on display.
- **x** – X position on LCD.
- **y** – Y position on LCD.
- **w** – Width of buffer.
- **h** – Height of buffer.

```
static inline int32_t pi_display_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
Send display specific command.
```

This function is called to send a display device specific command. Arguments for commands can be given using

#### Parameters

- **arg** – parameter.
- **device** – A pointer to the device structure of the device to open.
- **cmd** – Command to send.
- **arg** – Parameters/arguments to send to device for the command.

**Returns** Value Depend on type of commands.

```
static inline void pi_display_write_async(struct pi_device *device, pi_buffer_t *buffer, uint16_t x,
                                         uint16_t y, uint16_t w, uint16_t h, pi_evt_t *task)
```

Write on a display device. Non blocking API.

This function is called to print/write a buffer on a display device. It displays a whole frame of (width \* height) size on display at position (x, y).

---

**Note:** This function is asynchronous, caller can do something else while transfer is enqueued. The pending synchronous function is below : [pi\\_display\\_write\(\)](#).

---

#### Parameters

- **device** – A pointer to the device structure of the device to open.
- **buffer** – Data buffer to write on display.
- **x** – X position on LCD.
- **y** – Y position on LCD.
- **w** – Width of buffer.
- **h** – Height of buffer.
- **task** – Task to use to check end of transfer.

## Flash

### Common API

#### group Flash

The flash driver provides support for transferring data between an external flash chip (e.g. Hyperflash or SPI flash) and the processor running this driver.

#### Enums

##### enum **pi\_flash\_ioctl\_e**

Command ID for pi\_flash\_ioctl.

*Values:*

##### enumerator **PI\_FLASH\_IOCTL\_INFO**

Command for getting flash information. The argument must be a pointer to a variable of type struct *pi\_flash\_info* so that the call is returning information there.

##### enumerator **PI\_FLASH\_IOCTL\_SET\_BAUDRATE**

Command for setting baudrate.

##### enumerator **PI\_FLASH\_IOCTL\_AES\_ENABLE**

Command for setting aes enable state

##### enumerator **PI\_FLASH\_IOCTL\_SET\_MBA**

Command for setting mapping base addr for flash cs.

#### Functions

##### int **pi\_flash\_open**(struct pi\_device \*device)

Open a flash device.

This function must be called before the flash device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. The configuration associated to the device must specify the exact model of flash which must be opened.

##### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

##### static inline void **pi\_flash\_close**(struct pi\_device \*device)

Close an opened flash device.

This function can be called to close an opened flash device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

##### Parameters

- **device** – The device structure of the device to close.

```
static inline int32_t pi_flash_ioctl(struct pi_device *device, uint32_t cmd, void *arg)
```

Control device.

This can be called to configure and control the device after it has been opened.

#### Parameters

- **device** – The device structure of the device to control.
- **cmd** – The command to execute on the device.
- **arg** – The argument to the command. The size and meaning of this parameter depends on the command which is passed.

```
static inline void pi_flash_reg_set(struct pi_device *device, uint32_t reg_addr, uint8_t *value)
```

Set flash register.

This can be called to set a flash register. This must be used carefully as this can disturb the behavior of the other calls. The list of registers is specific to each flash, see the documentation of the flash for more information.

#### Parameters

- **device** – The device structure of the flash.
- **reg\_addr** – The register address.
- **value** – A pointer to the value to be set. The size of this variable depends on the register being accessed.

```
static inline void pi_flash_reg_get(struct pi_device *device, uint32_t reg_addr, uint8_t *value)
```

Get flash register.

This can be called to get a flash register. The list of registers is specific to each flash, see the documentation of the flash for more information.

#### Parameters

- **device** – The device structure of the flash.
- **reg\_addr** – The register address.
- **value** – A pointer to the value where the read value will be stored. The size of this variable depends on the register being accessed.

```
static inline void pi_flash_reg_set_async(struct pi_device *device, uint32_t reg_addr, uint8_t *value,
```

pi\_evt\_t \*task)

Set flash register asynchronously.

This can be called to set a flash register. This must be used carefully as this can disturb the behavior of the other calls. The list of registers is specific to each flash, see the documentation of the flash for more information. A task must be specified in order to specify how the caller should be notified when the operation is finished.

#### Parameters

- **device** – The device structure of the flash.
- **reg\_addr** – The register address.
- **value** – A pointer to the value to be set. The size of this variable depends on the register being accessed.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

---

```
static inline void pi_flash_reg_get_async(struct pi_device *device, uint32_t reg_addr, uint8_t *value,
                                         pi_evt_t *task)
```

Get flash register asynchronously.

This can be called to get a flash register. The list of registers is specific to each flash, see the documentation of the flash for more information. A task must be specified in order to specify how the caller should be notified when the operation is finished.

#### Parameters

- **device** – The device structure of the flash.
- **reg\_addr** – The register address.
- **value** – A pointer to the value where the read value will be stored. The size of this variable depends on the register being accessed.
- **task** – The task used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

```
static inline void pi_flash_read(struct pi_device *device, uint32_t pi_flash_addr, void *data, uint32_t size)
```

Enqueue a read copy to the flash (from flash to processor).

The copy will make a transfer between the flash and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the copy.
- **pi\_flash\_addr** – The address of the copy in the flash.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
static inline void pi_flash_program(struct pi_device *device, uint32_t pi_flash_addr, const void *data,
                                   uint32_t size)
```

Enqueue a write copy to the flash (from processor to flash).

The copy will make a write transfer from one of the processor memory areas to the flash. The locations in the flash being written should have first been erased. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the copy.
- **pi\_flash\_addr** – The address of the copy in the flash.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
static inline void pi_flash_erase_chip(struct pi_device *device)
```

Erase the whole flash.

This will erase the entire flash. The duration of this operation may be long and may be retrieved from the datasheet. The caller is blocked until the operation is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.

```
static inline void pi_flash_erase_sector(struct pi_device *device, uint32_t pi_flash_addr)
Erase a sector.
```

This will erase one sector. The duration of this operation may be long and may be retrieved from the datasheet. The caller is blocked until the operation is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.
- **pi\_flash\_addr** – The address of the sector to be erased.

```
static inline void pi_flash_erase(struct pi_device *device, uint32_t pi_flash_addr, int size)
Erase an area in the flash.
```

This will erase the specified area. The duration of this operation may be long and may be retrieved from the datasheet. If the flash only supports sector erasing, all the sectors partially or entirely covered by this area will be erased. The caller is blocked until the operation is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.
- **pi\_flash\_addr** – The address of the area to be erased.
- **size** – The size of the area to be erased.

```
static inline void pi_flash_read_async(struct pi_device *device, uint32_t pi_flash_addr, void *data,
                                      uint32_t size, pi_evt_t *task)
```

Enqueue an asynchronous read copy to the flash (from flash to processor).

The copy will make a transfer between the flash and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the copy.
- **pi\_flash\_addr** – The address of the copy in the flash.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_flash_program_async(struct pi_device *device, uint32_t pi_flash_addr, const void
                                         *data, uint32_t size, pi_evt_t *task)
```

Enqueue an asynchronous write copy to the flash (from processor to flash).

The copy will make a write transfer from one of the processor memory areas to the flash. The locations in the flash being written should have first been erased. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the copy.
- **pi\_flash\_addr** – The address of the copy in the flash.
- **data** – The address of the copy in the processor.

- **size** – The size in bytes of the copy.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_flash_erase_chip_async(struct pi_device *device, pi_evt_t *task)
    Erase the whole flash asynchronously.
```

This will erase the entire flash. The duration of this operation may be long and may be retrieved from the datasheet. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_flash_erase_sector_async(struct pi_device *device, uint32_t pi_flash_addr, pi_evt_t
    *task)
```

Erase a sector asynchronously.

This will erase one sector. The duration of this operation may be long and may be retrieved from the datasheet. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.
- **pi\_flash\_addr** – The address of the sector to be erased.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_flash_erase_async(struct pi_device *device, uint32_t pi_flash_addr, int size, pi_evt_t
    *task)
```

Erase an area in the flash asynchronously.

This will erase the specified area. The duration of this operation may be long and may be retrieved from the datasheet. If the flash only supports sector erasing, all the sectors partially or entirely covered by this area will be erased. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

#### Parameters

- **device** – The device descriptor of the flash chip on which to do the operation.
- **pi\_flash\_addr** – The address of the area to be erased.
- **size** – The size of the area to be erased.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
struct pi_flash_conf
#include <flash.h> Flash configuration structure.
```

This structure is used to pass the desired flash configuration to the runtime when opening the device. This configuration should not be used directly as it is meant to be encapsulated into a specific device configuration.

## Public Members

`pi_flash_api_t *api`

Pointer to specific flash methods. Reserved for internal runtime usage.

`struct pi_flash_info`

`#include <flash.h>` Parameter for FLASH\_IOCTL\_INFO command.

This structure is used to return flash information when executing the FLASH\_IOCTL\_INFO ioctl command.

## Public Members

`uint32_t sector_size`

Size in bytes of a sector.

`uint32_t flash_start`

Start address in the flash. What is before is reserved for runtime usage and should not be accessed.

## Hyperflash

*group Hyperflash*

### Functions

`void pi_hyperflash_conf_init(struct pi_hyperflash_conf *conf)`

Initialize an Hyperflash configuration with default values.

The structure containing the configuration must be kept alive until the hyperflash device is opened.

#### Parameters

- `conf` – A pointer to the hyperflash configuration.

`void pi_hyperflash_deep_sleep_enter(pi_device_t *device)`

`void pi_hyperflash_deep_sleep_exit(pi_device_t *device)`

`struct pi_hyperflash_conf`

## Public Members

`struct pi_flash_conf flash`

Generic flash configuration.

`int hyper_itf`

Hyperbus interface where the flash is connected.

`int hyper_cs`

Chip select where the flash is connected.

`uint32_t baudrate`

Baudrate (in bytes/second).

`char skip_pads_config`

Skip pads configuration if set to 1.

`struct hyperflash_conf`

`#include <hyperflash.h>` Hyperflash configuration structure.

This structure is used to pass the desired Hyperflash configuration to the runtime when opening the device.

## FS

### group FS

The file-system driver provides support for accessing files on a flash. The following file-systems are available:

- Read-only file system. This file-system is very basic but quite-convenient to have access to input data. The open operation does not scale well when having lots of file so this file-system should only be used with few files.

### TypeDefs

`typedef struct pi_fs_info_s pi_fs_info_t`

`typedef struct pi_fs_file_s pi_fs_file_t`

FS file structure.

This structure is used by the runtime to store information about a file.

`typedef struct pi_fs_dir_s pi_fs_dir_t`

FS dir structure.

This structure is used by the runtime to store information about a directory.

`typedef struct pi_cl_fs_req_s pi_cl_fs_req_t`

FS cluster file request structure.

This structure is used by the runtime to manage a cluster remote operation with the FS. It must be instantiated once for each operation and must be kept alive until the operation is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

## Enums

enum **pi\_fs\_type\_e**

File-system type.

This can be used to select the type of file-system to mount.

*Values:*

enumerator **PI\_FS\_READ\_ONLY** = 0

Read-only file system.

enumerator **PI\_FS\_HOST** = 1

Host file system.

enumerator **PI\_FS\_LFS** = 2

LittleFS Filesystem.

enum **pi\_fs\_flags\_e**

File-system open flags.

This can be used to select the type of file-system to mount.

*Values:*

enumerator **PI\_FS\_FLAGS\_READ** = 0

File is opened for reading.

enumerator **PI\_FS\_FLAGS\_WRITE** = 1

File is opened for writing.

enumerator **PI\_FS\_FLAGS\_APPEND** = 2

File is opened for appending (writing at end of file).

enum **pi\_fs\_file\_type\_e**

*Values:*

enumerator **PI\_FS\_TYPE\_REG** = 1

enumerator **PI\_FS\_TYPE\_DIR** = 2

enum **pi\_fs\_error\_e**

*Values:*

enumerator **PI\_FS\_ERR\_OK** = 0

enumerator **PI\_FS\_ERR\_IO** = -5

enumerator **PI\_FS\_ERR\_CORRUPT** = -84

enumerator **PI\_FS\_ERR\_NOENT** = -2

enumerator **PI\_FS\_ERR\_EXIST** = -17

enumerator **PI\_FS\_ERR\_NOTDIR** = -20

enumerator **PI\_FS\_ERR\_ISDIR** = -21

---

```

enumerator PI_FS_ERR_NOTEEMPTY = -39
enumerator PI_FS_ERR_BADF = -9
enumerator PI_FS_ERR_FBIG = -27
enumerator PI_FS_ERR_INVAL = -22
enumerator PI_FS_ERR_NOSPC = -28
enumerator PI_FS_ERR_NOMEM = -12
enumerator PI_FS_ERR_NOATTR = -61
enumerator PI_FS_ERR_NAMETOOLONG = -36
enumerator PI_FS_ERR_UNSUPPORTED = -37
enumerator PI_FS_MOUNT_FLASH_ERROR = 1
enumerator PI_FS_MOUNT_MEM_ERROR = 2

```

## Functions

**void pi\_fs\_conf\_init(struct *pi\_fs\_conf* \*conf)**

Initialize a file-system configuration with default values.

The structure containing the configuration must be kept allocated until the file-system is mounted.

### Parameters

- **conf** – A pointer to the file-system configuration.

**int32\_t pi\_fs\_mount(struct pi\_device \*device)**

Mount a file-system.

This function must be called before the file-system device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

**void pi\_fs\_unmount(struct pi\_device \*device)**

Unmount a mounted file-system.

This function can be called to close a mounted file-system once it is not needed anymore, in order to free all allocated resources. Once this function is called, the file-system is not accessible anymore and must be mounted again before being used.

### Parameters

- **device** – The device structure of the FS to unmount.

**int32\_t pi\_fs\_mkdir(struct pi\_device \*device, const char \*path)**

Create a Directory (only for read/write fs i.e. little FS)

This function can be called to create a directory

### Parameters

- **device** – The device structure of the FS where to open the file.

- **path** – The path of the directory to be created

**Returns** 0 if successfull, other if error

```
int32_t pi_fs_remove(struct pi_device *device, const char *path)
    Remove a Directory or a file (only for read/write fs i.e. little FS)
```

This function can be called to remove a directory or a regular file.

#### Parameters

- **device** – The device structure of the FS where to open the file.
- **path** – The path of the directory or file to be remove

**Returns** 0 if successfull, other if error

```
int32_t pi_fs_ls(struct pi_device *device, const char *path)
    List the content of a directory.
```

This function can be called to list the content of directory which is printed to STDOUT. This is a debug function.

#### Parameters

- **device** – The device structure of the FS where to open the file.
- **path** – The path of the directory to be listed

```
pi_fs_file_t *pi_fs_dir_open(struct pi_device *device, const char *path)
    Open a diractory.
```

This function can be called to open a file on a file-system in order to read or write data to it.

#### Parameters

- **device** – The device structure of the FS where to open the file.
- **file** – The path to the directory to be opened.

**Returns** NULL if the directory is not found, or a handle identifying the directory which can be used with other functions.

```
void pi_fs_dir_close(pi_fs_file_t *file)
    Close a directory.
```

This function can be called to close an opened directory once it is not needed anymore in order to free the allocated resources.

#### Parameters

- **file** – The handle of the file to be closed.

```
int32_t pi_fs_dir_read(pi_fs_file_t *file, pi_fs_info_t *info)
    Read directory content.
```

This function can be called to read the content of a directory. The infos are read from the current position which is the beginning of the directory when the it is opened. The current position is incremented by the number of bytes read by the call to this function. The caller is blocked until the transfer is finished.

#### Parameters

- **file** – The handle of the file where to read directory information.
- **dir\_infos** – A structure with file information.

**Returns** 0 if end of directory, < 0 if error, > 0 otherwise be smaller than the requested size if the end of file is reached.

---

`pi_fs_file_t *pi_fs_open(struct pi_device *device, const char *file, int flags)`  
Open a file.

This function can be called to open a file on a file-system in order to read or write data to it.

#### Parameters

- **device** – The device structure of the FS where to open the file.
- **file** – The path to the file to be opened.
- **flags** – Optional flags to configure how the file is opened.

**Returns** NULL if the file is not found, or a handle identifying the file which can be used with other functions.

`void pi_fs_close(pi_fs_file_t *file)`  
Close a file.

This function can be called to close an opened file once it is not needed anymore in order to free the allocated resources.

#### Parameters

- **file** – The handle of the file to be closed.

`int32_t pi_fs_read(pi_fs_file_t *file, void *buffer, uint32_t size)`  
Read data from a file.

This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. The caller is blocked until the transfer is finished. Compared to `pi_fs_direct_read`, this functions can use a cache to optimize small transfers. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.

**Returns** The number of bytes actually read from the file. This can be smaller than the requested size if the end of file is reached.

`int32_t pi_fs_write(pi_fs_file_t *file, void *buffer, uint32_t size)`  
Write data to a file.

This function can be called to write data to an opened file. The data is written to the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes written by the call to this function. This function may not be supported by each file-system. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to write data.
- **buffer** – The memory location where the data to be written must be read.
- **size** – The size in bytes to write to the file.

**Returns** The number of bytes actually written to the file. This can be smaller than the requested size if the end of file is reached.

```
int32_t pi_fs_direct_read(pi_fs_file_t *file, void *buffer, uint32_t size)
```

Read data from a file with no intermediate cache.

This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. The caller is blocked until the transfer is finished. Compared to `pi_fs_read`, this function does direct read transfers from the flash without any cache. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.

**Returns** The number of bytes actually read from the file. This can be smaller than the requested size if the end of file is reached.

```
int32_t pi_fs_seek(pi_fs_file_t *file, unsigned int offset)
```

Reposition the current file position.

This function can be called to change the current position of a file. Note that this does not affect pending copies, but only the ones which will be enqueued after this call.

#### Parameters

- **file** – The handle of the file for which the current position is changed.
- **offset** – The offset where to set the current position. The offset can be between 0 for the beginning of the file and the file size.

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_seek_read(pi_fs_file_t *file, uint32_t offset, void *buffer, uint32_t size)
```

Reposition the current file position and do a read from there.

This function can be called to change the current position of a file. Note that this does not affect pending copies, but only the ones which will be enqueued after this call. It will then directly execute a read. Final position will be offset + size

#### Parameters

- **file** – The handle of the file for which the current position is changed.
- **offset** – The offset where to set the current position. The offset can be between 0 for the beginning of the file and the file size.
- **buffer** – Buffer into which data readen from file will be written
- **size** – Size to be readen from file

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_copy(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, int32_t ext2loc)
```

Copy data between a FS file and a chip memory.

This function can be called to transfer data between an opened file and a chip memory using a specified offset instead of a current position. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_copy_2d(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, uint32_t stride, uint32_t length, int32_t ext2loc)
```

Enqueue a 2D copy (rectangle area) between a FS file and a chip memory.

This function can be called to transfer data between an opened file and a chip memory using a specified offset instead of a current position. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_read_async(pi_fs_file_t *file, void *buffer, uint32_t size, pi_evt_t *task)
```

Read data from a file asynchronously.

This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Compared to pi\_fs\_direct\_read, this function can use a cache to optimize small transfers. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

**Returns** The number of bytes actually read from the file. This can be smaller than the requested size if the end of file is reached.

```
int32_t pi_fs_seek_read_async(pi_fs_file_t *file, uint32_t offset, void *buffer, uint32_t size, pi_evt_t *task)
```

Asynchronously Reposition the current file position and do a read from there.

This function can be called to change the current position of a file. Note that this does not affect pending copies, but only the ones which will be enqueued after this call. It will then directly execute a read. Final position will be offset + size

#### Parameters

- **file** – The handle of the file for which the current position is changed.
- **offset** – The offset where to set the current position. The offset can be between 0 for the beginning of the file and the file size.
- **buffer** – Buffer into which data readen from file will be written
- **size** – Size to be readen from file \paran task pi\_evt\_t to be pushed at the end of operation

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_write_async(pi_fs_file_t *file, void *buffer, uint32_t size, pi_evt_t *task)
```

Write data to a file asynchronously.

This function can be called to write data to an opened file. The data is written to the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes written by the call to this function. This functionmay not be supported by each file-system. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to write data.
- **buffer** – The memory location where the data to be written must be written.
- **size** – The size in bytes to write to the file.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

**Returns** The number of bytes actually written to the file. This can be smaller than the requested size if the end of file is reached.

```
int32_t pi_fs_direct_read_async(pi_fs_file_t *file, void *buffer, uint32_t size, pi_evt_t *task)
```

Read data from a file with no intermediate cache asynchronously.

This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Compared to pi\_fs\_read, this function does direct read transfers from the flash without any cache. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.
- **task** – The task used to notify the end of transfer.

**Returns** The number of bytes actually read from the file. This can be smaller than the requested size if the end of file is reached.

```
int32_t pi_fs_copy_async(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, int32_t ext2loc,
                         pi_evt_t *task)
```

Copy data between a FS file and a chip memory asynchronously.

This function can be called to transfer data between an opened file and a chip memory using a specified offset instead of a current position. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer.

**Returns** 0 if the operation was successful, -1 otherwise.

```
int32_t pi_fs_copy_2d_async(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, uint32_t stride,
                            uint32_t length, int32_t ext2loc, pi_evt_t *task)
```

Enqueue a 2D copy (rectangle area) between a FS file and a chip memory asynchronously.

This function can be called to transfer data between an opened file and a chip memory using a specified offset instead of a current position. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer.

**Returns** 0 if the operation was successful, -1 otherwise.

```
void pi_cl_fs_read(pi_fs_file_t *file, void *buffer, uint32_t size, pi_cl_fs_req_t *req)
```

Read data from a file from cluster side.

This function implements the same feature as pi\_fs\_read but can be called from cluster side in order to expose the feature on the cluster. This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. This operation is asynchronous and

its termination is managed through the request structure. Compared to pi\_fs\_direct\_read, this function can use a cache to optimize small transfers. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details. The only difference compared to pi\_cl\_fs\_read is that the file position is automatically set to 0 for the next transfer if the current transfer reaches the end of the file.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.
- **req** – The request structure used for termination.

`void pi_cl_fs_write(pi_fs_file_t *file, void *buffer, uint32_t size, pi_cl_fs_req_t *req)`  
Write data to a file from cluster side.

This function implements the same feature as pi\_fs\_write but can be called from cluster side in order to expose the feature on the cluster. This function can be called to write data to an opened file. The data is written to the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes written by the call to this function. This function may not be supported by each file-system. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to write data.
- **buffer** – The memory location where the data to be written must be read.
- **size** – The size in bytes to write to the file.
- **req** – The request structure used for termination.

**Returns** The number of bytes actually written to the file. This can be smaller than the requested size if the end of file is reached.

`void pi_cl_fs_direct_read(pi_fs_file_t *file, void *buffer, uint32_t size, pi_cl_fs_req_t *req)`  
Read data from a file with no intermediate cache from cluster side.

This function implements the same feature as pi\_fs\_direct\_read but can be called from cluster side in order to expose the feature on the cluster. This function can be called to read data from an opened file. The data is read from the current position which is the beginning of the file when the file is opened. The current position is incremented by the number of bytes read by the call to this function. This operation is asynchronous and its termination is managed through the request structure. Compared to pi\_cl\_fs\_read, this function does direct read transfers from the flash without any cache. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **buffer** – The memory location where the read data must be copied.
- **size** – The size in bytes to read from the file.
- **req** – The request structure used for termination.

`void pi_cl_fs_seek(pi_fs_file_t *file, uint32_t offset, pi_cl_fs_req_t *req)`  
Reposition the current file position from cluster side.

This function can be called from cluster side to change the current position of a file. Note that this does not affect pending copies, but only the ones which will be enqueued after this call. This operation is asynchronous and its termination is managed through the request structure.

#### Parameters

- **file** – The handle of the file for which the current position is changed.
- **offset** – The offset where to set the current position. The offset can be between 0 for the beginning of the file and the file size.
- **req** – The request structure used for termination.

```
void pi_cl_fs_seek_read(pi_fs_file_t *file, uint32_t offset, void *buffer, uint32_t size, pi_cl_fs_req_t *req)
```

Reposition the current file position and read from cluster side.

This function can be called from cluster side to change the current position of a file and do a combined read. Note that this does not affect pending copies, but only the ones which will be enqueued after this call. This operation is asynchronous and its termination is managed through the request structure. At the end, the file position will in fact be at offset+size

#### Parameters

- **file** – The handle of the file for which the current position is changed.
- **offset** – The offset where to set the current position. The offset can be between 0 for the beginning of the file and the file size.
- **buffer** – Buffer into which readen data will be written
- **size** – Size in bytes which will be readen from the file
- **req** – The request structure used for termination.

```
void pi_cl_fs_copy(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, int32_t ext2loc,
                   pi_cl_fs_req_t *req)
```

Copy data between a FS file and a chip memory from cluster side.

This function is a remote call that the cluster can do to transfer data between an opened file and a chip memory using a specified offset instead of a current position. This operation is asynchronous and its termination is managed through the request structure. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.
- **req** – The request structure used for termination.

```
void pi_cl_fs_copy_2d(pi_fs_file_t *file, uint32_t index, void *buffer, uint32_t size, uint32_t stride,
                      uint32_t length, int32_t ext2loc, pi_cl_fs_req_t *req)
```

Enqueue a 2D copy (rectangle area) between a FS file and a chip memory from cluster side.

This function is a remote call that the cluster can do to transfer data between an opened file and a chip memory using a specified offset instead of a current position. This operation is asynchronous and its termination is managed through the request structure. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

### Parameters

- **file** – The handle of the file where to read data.
- **index** – The offset in the file where to start accessing data.
- **buffer** – The memory location in the chip memory where the data is accessed.
- **size** – The size in bytes to transfer.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from file to the chip or 0 for the contrary.
- **req** – The request structure used for termination.

```
static inline int32_t pi_cl_fs_wait(pi_cl_fs_req_t *req)
```

Wait until the specified fs request has finished.

This blocks the calling core until the specified cluster remote copy is finished. As the remote copy is asynchronous, this also gives the number of bytes which was read.

### Parameters

- **req** – The request structure used for termination.

**Returns** The number of bytes actually read from the file. This can be smaller than the requested size if the end of file is reached.

```
struct pi_fs_info_s  
struct pi_fs_conf  
#include <fs.h> File-system configuration structure.
```

This structure is used to pass the desired file-system configuration to the runtime when mounting the file-system.

### Public Members

*pi\_fs\_type\_e type*

File-system type.

struct pi\_device \***flash**

Flash device. The flash device must be first opened and its device structure passed here.

char \***partition\_name**

useful if there are several partitions of this FS type. By default this field is set to null, which allows to find the first partition compatible with this type of FS.

bool **auto\_format**

Defined the behavior of the mount operation in case the file system could not be found in the partition. If auto\_format is set to false, An error is returned . In the opposite case, if auto\_format is set to true, the partition will be formated and ready to use. Not available in ReadFS.

---

`pi_fs_api_t *api`

Pointer to specific FS methods. Reserved for internal runtime usage.

## GPIO

### FXL6408

*group* **FXL6408**

I2C Controlled GPIO Expander (8 configurable IOs)

#### Defines

`FXL6408_IS_VALID_OUTPUT_STATE(state)`

#### Enums

enum **fxl6408\_gpio\_dir\_e**

Direction of a GPIO (input or output)

*Values:*

enumerator **FXL6408\_GPIO\_DIR\_INPUT** = 0x0

enumerator **FXL6408\_GPIO\_DIR\_OUTPUT** = 0x1

enum **fxl6408\_gpio\_output\_state\_e**

GPIO Output state

This has no effect in input mode.

*Values:*

enumerator **FXL6408\_GPIO\_OUTPUT\_STATE\_DISABLED** = 0x0

GPIO is in High-Z (impedance) mode

enumerator **FXL6408\_GPIO\_OUTPUT\_STATE\_LOW** = 0x1

GPIO is in low voltage level, or 0

enumerator **FXL6408\_GPIO\_OUTPUT\_STATE\_HIGH** = 0x2

GPIO is in high voltage level, or 1

enum **fxl6408\_gpio\_input\_trigger\_e**

GPIO Input trigger conditions (disabled, rising or falling edge)

This has no effect in output mode.

*Values:*

enumerator **FXL6408\_GPIO\_INPUT\_TRIGGER\_RISING** = 0x0

Input will trigger on a rising edge

enumerator **FXL6408\_GPIO\_INPUT\_TRIGGER\_FALLING** = 0x1  
Input will trigger on a falling edge

enumerator **FXL6408\_GPIO\_INPUT\_TRIGGER\_DISABLED** = 0x2  
Input will never trigger

enum **fxl6408\_gpio\_pull\_state\_e**  
GPIO Pull up/down state

This has no effect in output mode.

*Values:*

enumerator **FXL6408\_GPIO\_PULL\_STATE\_DOWN** = 0x0  
GPIO is set to pull down

enumerator **FXL6408\_GPIO\_PULL\_STATE\_UP** = 0x1  
GPIO is set to pull up

enumerator **FXL6408\_GPIO\_PULL\_STATE\_DISABLED** = 0x2  
GPIO pull is disabled

## Functions

**void pi\_fxl6408\_conf\_init(struct *pi\_fxl6408\_conf* \*conf)**

Initialize an FXL6408 configuration with default values.

The structure containing the configuration must be kept alive until the device is opened. It can only be called from fabric-controller side. It is not thread-safe and cannot be called from a pmsis task callback or interrupt handler.

### Parameters

- **conf** – [inout] Pointer to the device configuration.

**int pi\_fxl6408\_open(pi\_device\_t \*device)**

Open an FXL6408 device.

It can only be called from fabric-controller side. It is not thread-safe and cannot be called from a pmsis task callback or interrupt handler.

### Parameters

- **device** – [inout] Pointer to the device.

**Returns** 0 if successfull or any other value otherwise

**void pi\_fxl6408\_close(pi\_device\_t \*device)**

Close an FXL6408 device.

### Parameters

- **device** – [inout] device to be closed

**void pi\_fxl6408\_gpio\_conf\_init(pi\_fxl6408\_gpio\_conf\_t \*gpio\_conf)**

Initialize the configuration of a GPIO (Output, High-Z)

### Parameters

- **gpio\_conf** – [inout] configuration of the gpio

---

```
int pi_fxl6408_gpio_set(pi_device_t *device, pi_fxl6408_gpio_conf_t *gpio_conf)
Set a GPIO state.
```

It can only be called from fabric-controller side. It is not thread-safe and cannot be called from a pmsis task callback or interrupt handler.

#### Parameters

- **device** – [in] Pointer to the device.
- **gpio\_conf** – [in] GPIO configuration

**Returns** PI\_OK if successfull or any other value otherwise

```
int pi_fxl6408_input_status_get(pi_device_t *device, uint8_t *input_status)
Return the current status of inputs
```

Each bit of the input status is the status of the corresponding gpio input.

#### Parameters

- **device** – [in] pointer to the device
- **input\_status** – [out] value of the input status register

**Returns** PI\_OK if operation was successful, an error code otherwise.

```
int pi_fxl6408_interrupt_status_get(pi_device_t *device, uint8_t *interrupt_status)
Return the current status of interrupts
```

This will clear the interrupt status register.

Each bit of the interrupt status is the status of the corresponding gpio interrupt.

#### Parameters

- **device** – [in] pointer to the device
- **interrupt\_status** – [out] value of the interrupt status register

**Returns** PI\_OK if operation was successful, an error code otherwise.

```
struct pi_fxl6408_conf
#include <fxl6408.h> Struct holding FXL6408 display config.
```

### Public Members

```
int i2c_if
I2C interface number where the device is connected.
```

```
pi_gpio_e interrupt_pin
interrupt pin
```

```
struct pi_fxl6408_gpio_conf_t
#include <fxl6408.h> Structure holding the configuration of a FXL6408 GPIO
```

## Public Members

`uint8_t id`  
GPIO id(from 0 to 7)

`fxl6408_gpio_dir_e direction`  
Direction (input or output)

`fxl6408_gpio_output_state_e output_state`  
Output State (disabled/High-Z, 0 or 1)

`fxl6408_gpio_input_trigger_e input_trigger`  
Input trigger (trigger on falling edge, rising edge or disabled)

`fxl6408_gpio_pull_state_e pull_state`  
pull state (pull-up, pull-down, disabled)

`pi_evt_t *irq_task`  
task executed when an irq is detected

## RAM

### Common API

#### group Ram

The RAM driver provides support for transferring data between an external RAM chip (e.g. Hyperram or SPI ram) and the processor running this driver.

#### TypeDefs

`typedef struct pi_cl_ram_req_s pi_cl_ram_req_t`  
RAM cluster copy request structure.

This structure is used by the runtime to manage a cluster remote copy with the RAM. It must be instantiated once for each copy and must be kept alive until the copy is finished. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

`typedef struct pi_cl_ram_alloc_req_s pi_cl_ram_alloc_req_t`  
RAM cluster alloc request structure.

This structure is used by the runtime to manage a cluster remote allocation in the RAM. It must be instantiated once for each allocation and must be kept alive until the allocation is done. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

`typedef struct pi_cl_ram_free_req_s pi_cl_ram_free_req_t`  
RAM cluster free request structure.

This structure is used by the runtime to manage a cluster remote free in the RAM. It must be instantiated once for each free and must be kept alive until the free is done. It can be instantiated as a normal variable, for example as a global variable, a local one on the stack, or through a memory allocator.

## Enums

enum **pi\_ram\_ioctl\_e**

*Values:*

enumerator **PI\_RAM\_IOCTL\_INFO**

Command for getting ram information. The argument must be a pointer to a variable of type struct *pi\_ram\_info* so that the call is returning information there.

enumerator **PI\_RAM\_IOCTL\_AES\_ENABLE**

Command for setting aes enable state

enumerator **PI\_RAM\_IOCTL\_SET\_MBA**

Command for setting mapping base addr for flash cs.

## Functions

void **pi\_spiram\_conf\_init**(struct *pi\_spiram\_conf* \*conf)

Initialize an SPI ram configuration with default values.

The structure containing the configuration must be kept alive until the SPI ram device is opened.

### Parameters

- **conf** – A pointer to the SPI ram configuration.

int32\_t **pi\_ram\_open**(struct pi\_device \*device)

Open a RAM device.

This function must be called before the RAM device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions. The configuration associated to the device must specify the exact model of RAM which must be opened.

### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

static inline void **pi\_ram\_close**(struct pi\_device \*device)

Close an opened RAM device.

This function can be called to close an opened RAM device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

### Parameters

- **device** – The device structure of the device to close.

static inline int **pi\_ram\_alloc**(struct pi\_device \*device, uint32\_t \*addr, uint32\_t size)

Allocate RAM memory.

The allocated memory is 4-bytes aligned. The allocator uses some meta-data stored in the chip memory for every allocation so it is advisable to do as few allocations as possible to lower the memory overhead.

### Parameters

- **device** – The device structure of the device where to allocate the memory.

- **addr** – A pointer to the variable where the allocated address must be returned.
- **size** – The size in bytes of the memory to allocate

**Returns** 0 if the allocation succeeded, -1 if not enough memory was available.

```
static inline int pi_ram_free(struct pi_device *device, uint32_t addr, uint32_t size)
Free RAM memory.
```

The allocator does not store any information about the allocated chunks, thus the size of the allocated chunk to be freed must be provided by the caller.

#### Parameters

- **device** – The device structure of the device where to free the memory.
- **addr** – The allocated chunk to free
- **size** – The size in bytes of the memory chunk which was allocated

**Returns** 0 if the operation was successful, -1 otherwise

```
static inline void pi_ram_read(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t size)
Enqueue a read copy to the RAM (from RAM to processor).
```

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
static inline void pi_ram_write(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t size)
Enqueue a write copy to the RAM (from processor to RAM).
```

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

```
static inline void pi_ram_copy(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t size, int
ext2loc)
```

Enqueue a copy with the RAM.

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.

- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.

```
static inline void pi_ram_read_2d(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t size,
                                 uint32_t stride, uint32_t length)
```

Enqueue a 2D read copy (rectangle area) to the RAM (from RAM to processor).

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.

```
static inline void pi_ram_write_2d(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t
                                   size, uint32_t stride, uint32_t length)
```

Enqueue a 2D write copy (rectangle area) to the RAM (from processor to RAM).

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.

```
static inline void pi_ram_copy_2d(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t size,
                                 uint32_t stride, uint32_t length, int ext2loc)
```

Enqueue a 2D copy (rectangle area) with the RAM.

The copy will make a transfer between the RAM and one of the processor memory areas. The caller is blocked until the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.

```
static inline void pi_ram_read_async(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t  
size, pi_evt_t *task)
```

Enqueue an asynchronous read copy to the RAM (from RAM to processor).

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_ram_write_async(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t  
size, pi_evt_t *task)
```

Enqueue an asynchronous write copy to the RAM (from processor to RAM).

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_ram_copy_async(struct pi_device *device, uint32_t pi_ram_addr, void *data, uint32_t  
size, int ext2loc, pi_evt_t *task)
```

Enqueue an asynchronous copy with the RAM.

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished.

Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_ram_read_2d_async(struct pi_device *device, uint32_t pi_ram_addr, void *data,
                                       uint32_t size, uint32_t stride, uint32_t length, pi_evt_t *task)
```

Enqueue an asynchronous 2D read copy (rectangle area) to the RAM (from RAM to processor).

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **task** – The task used to notify the end of transfer. See the documentation of pi\_evt\_t for more details.

```
static inline void pi_ram_write_2d_async(struct pi_device *device, uint32_t pi_ram_addr, void *data,
                                         uint32_t size, uint32_t stride, uint32_t length, pi_evt_t *task)
```

Enqueue an asynchronous 2D write copy (rectangle area) to the RAM (from processor to RAM).

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy

- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **task** – The task used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

```
static inline void pi_ram_copy_2d_async(struct pi_device *device, uint32_t pi_ram_addr, void *data,  
                                     uint32_t size, uint32_t stride, uint32_t length, int ext2loc, pi_evt_t  
                                     *task)
```

Enqueue an asynchronous 2D copy (rectangle area) with the RAM.

The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A task must be specified in order to specify how the caller should be notified when the transfer is finished. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **data** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.
- **task** – The task used to notify the end of transfer. See the documentation of `pi_evt_t` for more details.

```
void pi_cl_ram_alloc(struct pi_device *device, uint32_t size, pi_cl_ram_alloc_req_t *req)
```

Allocate RAM memory from cluster side.

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM allocation. The allocated memory is 4-bytes aligned. The allocator uses some meta-data stored in the fabric controller memory for every allocation so it is advisable to do as few allocations as possible to lower the memory overhead.

#### Parameters

- **device** – The device descriptor of the RAM chip for which the memory must be allocated.
- **size** – The size in bytes of the memory to allocate.
- **req** – The request structure used for termination.

```
void pi_cl_ram_free(struct pi_device *device, uint32_t chunk, uint32_t size, pi_cl_ram_free_req_t *req)
```

Free RAM memory from cluster side.

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM allocation. The allocator does not store any information about the allocated chunks, thus the size of the allocated chunk to be freed must be provided by the caller.

#### Parameters

- **device** – The device descriptor of the RAM chip for which the memory must be freed.
- **chunk** – The allocated chunk to free.
- **size** – The size in bytes of the memory chunk which was allocated.
- **req** – The request structure used for termination.

```
static inline int32_t pi_cl_ram_alloc_wait(pi_cl_ram_alloc_req_t *req, uint32_t *addr)
    Wait until the specified RAM alloc request has finished.
```

This blocks the calling core until the specified cluster RAM allocation is finished.

#### Parameters

- **req** – The request structure used for termination.
- **addr** – A pointer to the variable where the allocated address must be returned.

**Returns** 0 if the allocation succeeded, -1 if not enough memory was available.

```
static inline int32_t pi_cl_ram_free_wait(pi_cl_ram_free_req_t *req)
    Wait until the specified RAM free request has finished.
```

This blocks the calling core until the specified cluster RAM free is finished.

#### Parameters

- **req** – The request structure used for termination.

**Returns** 0 if the operation was successful, -1 otherwise

```
static inline void pi_cl_ram_read(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t size,
                                 pi_cl_ram_req_t *req)
```

Enqueue a read copy to the RAM from cluster side (from RAM to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM read copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_ram_write(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t
                                  size, pi_cl_ram_req_t *req)
```

Enqueue a write copy to the RAM from cluster side (from RAM to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM write copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
void pi_cl_ram_copy(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t size, int ext2loc,  
                     pi_cl_ram_req_t *req)
```

Enqueue a copy with the RAM from cluster side.

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_ram_read_2d(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t  
                                    size, uint32_t stride, uint32_t length, pi_cl_ram_req_t *req)
```

Enqueue a 2D read copy (rectangle area) to the RAM from cluste side (from RAM to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM read copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy.
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_ram_write_2d(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t
size, uint32_t stride, uint32_t length, pi_cl_ram_req_t *req)
```

Enqueue a 2D write copy (rectangle area) to the RAM from cluster side (from RAM to processor).

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM write copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
void pi_cl_ram_copy_2d(struct pi_device *device, uint32_t pi_ram_addr, void *addr, uint32_t size, uint32_t
stride, uint32_t length, int ext2loc, pi_cl_ram_req_t *req)
```

Enqueue a 2D copy (rectangle area) with the RAM from cluster side.

This function is a remote call that the cluster can do to the fabric-controller in order to ask for a RAM copy. The copy will make an asynchronous transfer between the RAM and one of the processor memory areas. A pointer to a request structure must be provided so that the runtime can properly do the remote call. Depending on the chip, there may be some restrictions on the memory which can be used. Check the chip-specific documentation for more details.

#### Parameters

- **device** – The device descriptor of the RAM chip on which to do the copy.
- **pi\_ram\_addr** – The address of the copy in the RAM.
- **addr** – The address of the copy in the processor.
- **size** – The size in bytes of the copy
- **stride** – 2D stride, which is the number of bytes which are added to the beginning of the current line to switch to the next one.
- **length** – 2D length, which is the number of transferred bytes after which the driver will switch to the next line.
- **ext2loc** – 1 if the copy is from RAM to the chip or 0 for the contrary.
- **req** – A pointer to the RAM request structure. It must be allocated by the caller and kept alive until the copy is finished.

```
static inline void pi_cl_ram_read_wait(pi_cl_ram_req_t *req)
```

Wait until the specified RAM request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_ram_write_wait(pi_cl_ram_req_t *req)
```

Wait until the specified RAM request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

### Parameters

- **req** – The request structure used for termination.

```
static inline void pi_cl_ram_copy_wait(pi_cl_ram_req_t *req)
```

Wait until the specified RAM request has finished.

This blocks the calling core until the specified cluster remote copy is finished.

### Parameters

- **req** – The request structure used for termination.

```
struct pi_spiram_conf
```

#include <spiram.h> SPI ram configuration structure.

This structure is used to pass the desired SPI ram configuration to the runtime when opening the device.

### Public Members

```
struct pi_ram_conf ram
```

Generic RAM configuration.

```
int spi_itf
```

SPI interface where the RAM is connected.

```
int spi_cs
```

Chip select where the RAM is connected.

```
char skip_pads_config
```

Skip pads configuration if set to 1.

```
int ram_start
```

SPI ram start address.

```
int ram_size
```

SPI ram size.

```
uint32_t baudrate
```

Baudrate (in bytes/second).

```
struct pi_ram_conf
```

#include <ram.h> RAM configuration structure.

This structure is used to pass the desired RAM configuration to the runtime when opening the device. This configuration should be not be used directly as it is meant to be encapsulated into a specific device configuration.

## Public Members

`pi_ram_api_t *api`

Pointer to specific RAM methods. Reserved for internal runtime usage.

`struct pi_ram_info`

## Hyperram

*group Hyperram*

### Functions

`void pi_hyperram_conf_init(struct pi_hyperram_conf *conf)`

Initialize an Hyperram configuration with default values.

The structure containing the configuration must be kept alive until the hyperram device is opened.

#### Parameters

- `conf` – A pointer to the hyperram configuration.

`struct pi_hyperram_conf`

`#include <hyperram.h>` Hyperram configuration structure.

This structure is used to pass the desired Hyperram configuration to the runtime when opening the device.

## Public Members

`struct pi_ram_conf ram`

Generic RAM configuration.

`int hyper_itf`

Hyperbus interface where the RAM is connected.

`int hyper_cs`

Chip select where the RAM is connected.

`char skip_pads_config`

Skip pads configuration if set to 1.

`int xip_en`

enable xip mode if set to 1.

`int ram_start`

Hyperram start address.

`int ram_size`

Hyperram size.

`uint32_t baudrate`

Baudrate (in bytes/second).

`int reserve_addr_0`

Reserve address 0 and never return a chunk with address 0.

## Audio

### TLV320

*group* **TLV320**

ADC TLV320.

#### Defines

**TLV320\_IS\_INCLUDED**(val, min, max)

Generic inclusion test. Helper on register value validness tests.

#### Parameters

- **val** – [in] Value to test.
- **min** – [in] Range's minimal value (included).
- **max** – [in] Range's maximal value (included).

**TLV320\_SLAVE\_ADDRESS\_1**

Default slave address that match the following configuration : ADDR1\_MISO = 0 & ADDR0\_SCLK = 0.

**TLV320\_SLAVE\_ADDRESS\_2**

Slave address that match the following configuration : ADDR1\_MISO = 0 & ADDR0\_SCLK = 1.

**TLV320\_SLAVE\_ADDRESS\_3**

Slave address that match the following configuration : ADDR1\_MISO = 1 & ADDR0\_SCLK = 0.

**TLV320\_SLAVE\_ADDRESS\_4**

Slave address that match the following configuration : ADDR1\_MISO = 1 & ADDR0\_SCLK = 1.

**TLV320\_ADDR0\_SLAVE\_ADDR\_BITSHIFT**

TLV320 addr0 pin impact on slave address value bit n°0.

**TLV320\_ADDR1\_SLAVE\_ADDR\_BITSHIFT**

TLV320 addr1 pin impact on slave address value bit n°1.

**TLV320\_PAGE\_CONFIG\_REG**

Register page about configuration register.

**TLV320\_PAGE\_COEF\_BIQUAD\_1\_6**

Register page about programmable coefficients for the biquad 7 to biquad 12 filters.

**TLV320\_PAGE\_COEF\_BIQUAD\_7\_12**

Register page about programmable coefficients for the biquad 7 to biquad 12 filters.

**TLV320\_PAGE\_COEF\_MIXER\_1\_4**

Register page about Programmable coefficients for mixer 1 to mixer 4.

**TLV320\_I2C\_MSG\_SIZE**

TLV320 typical message size (adress & data)

**TLV320\_I2C\_WRITE\_FLAGS**

TLV320 I2C write flags. (One start bit & one stop bit)

**TLV320\_I2C\_READ\_TX\_SIZE**

TLV320 I2C read tx size.

**TLV320\_I2C\_READ\_RX\_SIZE**

TLV320 I2C read rx size.

**TLV320\_I2C\_MAX\_BAUDRATE**

TLV320 I2C max baudrate.

**TLV320\_NO\_ERROR**

TLV320 Error codes.

Code returned when nothing wrong happened

**TLV320\_GENERIC\_ERROR**

Code returned when a non specified error occurred.

**TLV320\_WRITE\_ERROR**

Code returned when an i2C write error occurred.

**TLV320\_SLAVEADDRESS\_ERROR**

Code returned when the slave adress has not been recognized.

**TLV320\_READ\_ERROR**

Code returned when a read operation failed (mostly due to a page cfg write error)

**TLV320\_SLEEP\_MODE\_WAKEUP\_DELAY**

Delay needed after recovering from sleep mode. 1ms.

**TLV320\_SLEEP\_MODE\_SLEEP\_DELAY**

Delay needed after entering to sleep mode. 10ms.

**TLV320\_CHANNEL\_1**

Channel 1 id.

**TLV320\_CHANNEL\_2**

Channel 2 id.

**TLV320\_CHANNEL\_3**

Channel 3 id.

**TLV320\_CHANNEL\_4**

Channel 4 id.

**TLV320\_CHANNEL\_5**

Channel 5 id.

**TLV320\_CHANNEL\_6**

Channel 6 id.

**TLV320\_CHANNEL\_7**

Channel 7 id.

**TLV320\_CHANNEL\_8**

Channel 8 id.

**TLV320\_CHANNEL\_NB**

Total amount of channels.

**TLV320\_ANALOG\_CHANNEL\_NB**

Number of analog channels.

**TLV320\_IS\_VALID\_ANALOG\_CHANNEL\_ID(id)**

TLV320 analog channel id validness test.

---

**Note:** TLV320 device only have 4 analog channels.

---

**Parameters**

- **id** – [in] Analog channel id to test. Accepted values:
  - It can be a value from [TLV320\\_CHANNEL\\_1](#) and [TLV320\\_CHANNEL\\_4](#).

**TLV320\_IS\_VALID\_CHANNEL\_ID(id)**

TLV320 channel id validness test.

---

**Note:** TLV320 device only have 4 analog channels.

---

**Parameters**

- **id** – [in] channel id to test. Accepted values:
  - It can be a value from [TLV320\\_CHANNEL\\_1](#) and [TLV320\\_CHANNEL\\_8](#).

**TLV320\_IS\_VALID\_STATE(state)**

**TLV320\_IS\_VALID\_HW\_MODE(state)**

**TLV320\_IS\_VALID\_CONFIG\_REG(reg)**  
Configuration register code validness test.

#### Parameters

- **reg** – [in] Register value to verify.

**TLV320\_SW\_RESET\_RESET**  
Software Reset.

**TLV320\_SLEEP\_CFG\_AREG\_SELECT\_EXTERNAL**  
AREG\_SELECT bit setting for an external 1.8-V AREG supply.

**TLV320\_SLEEP\_CFG\_AREG\_SELECT\_INTERNAL**  
AREG\_SELECT bit setting for an internally generated 1.8-V AREG supply using an on-chip regulator.

**TLV320\_SLEEP\_CFG\_VREF\_QCHG\_3\_5\_MS**  
VREF\_QCHG bits setting for a VREF quick-charge duration of 3.5 ms.

**TLV320\_SLEEP\_CFG\_VREF\_QCHG\_10\_MS**  
VREF\_QCHG bits setting for a VREF quick-charge duration of 10 ms.

**TLV320\_SLEEP\_CFG\_VREF\_QCHG\_50\_MS**  
VREF\_QCHG bits setting for a VREF quick-charge duration of 50 ms.

**TLV320\_SLEEP\_CFG\_VREF\_QCHG\_100\_MS**  
VVREF\_QCH bits setting for a GREF quick-charge duration of 100 ms.

**TLV320\_SLEEP\_CFG\_I2C\_BRDCAST\_EN\_DISABLED**  
I2C\_BRDCAST\_EN bit setting for I2C broadcast mode disabled; the I2C slave address is determined based on ADDR pins.

**TLV320\_SLEEP\_CFG\_I2C\_BRDCAST\_EN\_ENABLED**  
I2C\_BRDCAST\_EN bit setting for I2C broadcast mode enabled; the I2C slave address is fixed at 1001100.

**TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_ENABLED**  
SLEEP\_ENZ bit setting to set the device in sleep mode.

**TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_DISABLED**  
SLEEP\_ENZ bit setting to not set the device in sleep mode (active mode)

**TLV320\_IS\_VALID\_SLEEP\_MODE(mode)**  
TLV320 sleep mode validness test.

#### Parameters

- **mode** – [in] Sleep mode value to test. Accepted values:
  - *TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_ENABLED*
  - *TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_DISABLED*

**TLV320\_ASI\_CFG0\_ASI\_FORMAT\_TDM**  
ASI\_FORMAT bits setting of Time division multiplexing mode.

**TLV320\_ASICFG0\_ASIFORMAT\_I2S**

ASI\_FORMAT bits setting of Inter IC sound mode.

**TLV320\_ASICFG0\_ASIFORMAT\_LJ**

ASI\_FORMAT bits setting of Left-justified mode.

**TLV320\_ASICFG0\_ASILEN\_16BITS**

ASI\_WLEN bits settings for 16 bits word lenght.

**TLV320\_ASICFG0\_ASILEN\_20BITS**

ASI\_WLEN bits settings for 20 bits word lenght.

**TLV320\_ASICFG0\_ASILEN\_24BITS**

ASI\_WLEN bits settings for 24 bits word lenght.

**TLV320\_ASICFG0\_ASILEN\_32BITS**

ASI\_WLEN bits settings for 32 bits word lenght.

**TLV320\_ASICFG0\_FSYNC\_POL\_NORMAL**

FSYNC\_POL bit setting for a normal FSYNC polarity.

**TLV320\_ASICFG0\_FSYNC\_POL\_INVERTED**

FSYNC\_POL bit setting for a inverted FSYNC polarity.

**TLV320\_ASICFG0\_BCLK\_POL\_NORMAL**

BCLK\_POL bit setting for a normal bit clock polarity.

**TLV320\_ASICFG0\_BCLK\_POL\_INVERTED**

BCLK\_POL bit setting for an inverted bit clock polarity.

**TLV320\_ASICFG0\_TX\_EDGE\_NORMAL**

TX\_EDGE bit setting for a default edge matching BCLK\_POL bit setting.

**TLV320\_ASICFG0\_TX\_EDGE\_INVERTED**

TX\_EDGE bit setting for an inverted following edge (half cycle delay) with respect to the default edge setting.

**TLV320\_ASICFG0\_TX\_FILL\_ZERO**

TX\_FILL bit setting to always transmit 0 for unused cycles.

**TLV320\_ASICFG0\_TX\_FILL\_HI\_Z**

TX\_FILL bit setting to use High-Z for unused cycles.

**TLV320\_IS\_VALID\_ASIFORMAT(format)**

TLV320 ASI format validness test

**Parameters**

- **format** – [in] ASI format to test. Accepted values:
  - *TLV320\_ASICFG0\_ASIFORMAT\_TDM*
  - *TLV320\_ASICFG0\_ASIFORMAT\_I2S*

- *TLV320\_ASI\_CFG0\_ASI\_FORMAT\_LJ*

**TLV320\_IS\_VALID\_ASI\_WLEN**(wlen)  
TLV320 ASI word lenght validness

#### Parameters

- **wlen** – [in] word lenght to test. Accepted values:
  - *TLV320\_ASI\_CFG0\_ASI\_WLEN\_16BITS*
  - *TLV320\_ASI\_CFG0\_ASI\_WLEN\_20BITS*
  - *TLV320\_ASI\_CFG0\_ASI\_WLEN\_24BITS*
  - *TLV320\_ASI\_CFG0\_ASI\_WLEN\_32BITS*

**TLV320\_IS\_VALID\_ASI\_FSYNC\_POL**(pol)  
TLV320 FSync polarity validness test.

#### Parameters

- **pol** – [in] polarity to test. Accepted values:
  - *TLV320\_ASI\_CFG0\_FSYNC\_POL\_NORMAL*
  - *TLV320\_ASI\_CFG0\_FSYNC\_POL\_INVERTED*

**TLV320\_IS\_VALID\_ASI\_BCLK\_POL**(pol)  
TLV320 Bit clock polarity validness test.

#### Parameters

- **pol** – [in] polarity to test. Accepted values:
  - *TLV320\_ASI\_CFG0\_BCLK\_POL\_NORMAL*
  - *TLV320\_ASI\_CFG0\_BCLK\_POL\_INVERTED*

**TLV320\_IS\_VALID\_ASI\_TX\_EDGE**(edge)  
TLV320 ASI output transmission edge validness test.

#### Parameters

- **edge** – [in] Tx edge to test. Accepted values:
  - *TLV320\_ASI\_CFG0\_TX\_EDGE\_NORMAL*
  - *TLV320\_ASI\_CFG0\_TX\_EDGE\_INVERTED*

**TLV320\_IS\_VALID\_ASI\_TX\_FILL**(edge)  
TLV320 data output for any unused cycles validness test.

#### Parameters

- **edge** – [in] Tx fill value to test. Accepted values:
  - *TLV320\_ASI\_CFG0\_TX\_FILL\_ZERO*
  - *TLV320\_ASI\_CFG0\_TX\_FILL\_HI\_Z*

**TLV320\_ASI\_CFG1\_TX\_LSB\_FULL\_CYCLE**  
TX\_LSB bit setting to transmit the LSB for a full cycle.

**TLV320\_ASI\_CFG1\_TX\_LSB\_HALF\_CYCLE**  
TX\_LSB bit setting to transmit the LSB for the first half cycle and Hi-Z for the second half cycle.

**TLV320\_ASI\_CFG1\_TX\_KEEPER\_DISABLED**

TX\_KEEPER bits setting to always disable the bus keeper.

**TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED**

TX\_KEEPER bits setting to always enable the bus keeper.

**TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_CYCLE**

TX\_KEEPER bits setting to enable the bus keeper during LSB transmissions only for one cycle.

**TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_5\_CYCLES**

TX\_KEEPER bits setting to enable the bus keeper during LSB transmissions only for one and half cycle.

**TLV320\_ASI\_CFG1\_TX\_OFFSET\_MIN**

TX\_OFFSET minimal accepted value.

**TLV320\_ASI\_CFG1\_TX\_OFFSET\_DISABLED**

TX\_OFFSET bits setting to disable the offset feature.

**TLV320\_ASI\_CFG1\_TX\_OFFSET\_SIZE**

TX\_OFFSET setting bits number.

**TLV320\_ASI\_CFG1\_TX\_OFFSET\_MAX**

TX\_OFFSET maximal accepted value.

**TLV320\_ASI\_CFG1\_TX\_OFFSET\_DEFAULT**

TX\_OFFSET default value.

**TLV320\_IS\_VALID\_ASI\_TX\_LSB\_CYCLE**(cycle)

TLV320 ASI data output for LSB transmission validness test.

**Parameters**

- **cycle** – [in] Value to test. Accepted values:
  - *TLV320\_ASI\_CFG1\_TX\_LSB\_FULL\_CYCLE*
  - *TLV320\_ASI\_CFG1\_TX\_LSB\_HALF\_CYCLE*

**TLV320\_IS\_VALID\_ASI\_TX\_KEEPER**(keeper)

TLV320 ASI data output bus keeper validness test.

**Parameters**

- **keeper** – [in] Value to test. Accepted values:
  - *TLV320\_ASI\_CFG1\_TX\_KEEPER\_DISABLED*
  - *TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED*
  - *TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_CYCLE*
  - *TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_5\_CYCLES*

**TLV320\_IS\_VALID\_TX\_OFFSET**(offset)

TLV320 ASI data MSB slot 0 offset validness test.

**Parameters**

- **offset** – [in] Value to test. Accepted values:

- Between *TLV320\_ASI\_CFG1\_TX\_OFFSET\_MIN* and *TLV320\_ASI\_CFG1\_TX\_OFFSET\_MAX*

**TLV320\_ASI\_CHx\_OUTPUT\_PRIMARY**

Channel output is on the ASI primary output pin (SDOUT)

**TLV320\_ASI\_CHx\_OUTPUT\_SECONDARY**

Channel output is on the ASI secondary output pin (GPIO1 or GPOx)

**TLV320\_TDM\_SLOT\_MIN**

Minimal acceptable slot value.

**TLV320\_TDM\_SLOT\_MAX**

Maximal acceptable slot value.

**TLV320\_TDM\_SLOT\_DEFAULT**

Default slot value.

**TLV320\_IS\_VALID\_ASI\_CHx\_OUTPUT**(output)

TLV320 ASI data output for LSB transmission validness test.

**Parameters**

- **output** – [in] ASI output type value to test. Accepted values:
  - *TLV320\_ASI\_CHx\_OUTPUT\_PRIMARY*
  - *TLV320\_ASI\_CHx\_OUTPUT\_SECONDARY*

**TLV320\_IS\_VALID\_TDM\_SLOT\_ID**(slot)

Slot ID validness test.

**Parameters**

- **slot** – [in] Value to test. Accepted values:
  - Between TLV320\_SLOT\_MIN and TLV320\_SLOT\_MAX

**TLV320\_MSTCFG0\_SLAVE**

Device is in slave mode (both BCLK and FSYNC are inputs to the device)

**TLV320\_MSTCFG0\_MASTER**

Device is in master mode (both BCLK and FSYNC are generated from the device)

**TLV320\_MSTCFG0\_AUTOCLK\_ENABLE**

Auto clock configuration is enabled (all internal clock divider and PLL configurations are auto derived)

**TLV320\_MSTCFG0\_AUTOCLK\_DISABLE**

Auto clock configuration is disabled (custom mode and device GUI must be used for the device configuration settings)

**TLV320\_MSTCFG0\_PLLAUTOMODE\_ENABLE**

PLL is enabled in auto clock configuration.

**TLV320\_MSTCFG0\_PLLAUTOMODE\_DISABLE**

PLL is disabled in auto clock configuration.

**TLV320\_MSTCFG0\_BCLKFSYNC\_GATE\_DISABLE**

Do not gate BCLK and FSYNC.

**TLV320\_MSTCFG0\_BCLKFSYNC\_GATE\_ENABLE**

Force gate BCLK and FSYNC when being transmitted from the device in master mode.

**TLV320\_MSTCFG0\_FSMODE\_48KHZ**

FS is a multiple (or submultiple) of 48 kHz.

**TLV320\_MSTCFG0\_FSMODE\_44\_1KHZ**

FS is a multiple (or submultiple) of 44.1 kHz.

**TLV320\_MSTCFG0\_MCLK\_12\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 12 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_12\_288\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 12,288 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_13\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 13 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_16\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 16 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_19\_2\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 19,2 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_19\_68\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 19,68 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_24\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 24 MHz for the PLL source clock input.

**TLV320\_MSTCFG0\_MCLK\_24\_576\_MHZ**

MCLK (GPIO or GPIx) frequency is set to 24,576 MHz for the PLL source clock input.

**TLV320\_IS\_VALID\_ASI\_MASTER\_SLAVE\_Cfg(cfg)**

ASI master or slave configuration setting validness test.

**Parameters**

- **cfg** – [in] ASI master slave configuration to test. Accepted values:

- *TLV320\_MSTCFG0\_SLAVE*
- *TLV320\_MSTCFG0\_MASTER*

**TLV320\_IS\_VALID\_AUTO\_CLK(cfg)**

Automatic clock configuration setting validness test.

**Parameters**

- **cfg** – [in] ASI master slave configuration to test. Accepted values:

- *TLV320\_MSTCFG0\_AUTOCLK\_DISABLE*
- *TLV320\_MSTCFG0\_AUTOCLK\_ENABLE*

**TLV320\_IS\_VALID\_PLLAUTOMODE(mode)**

Automatic mode PLL setting validness test.

**Parameters**

- **mode** – [in] ASI master slave configuration to test. Accepted values:
  - *TLV320\_MSTCFG0\_PLLAUTOMODE\_DISABLE*
  - *TLV320\_MSTCFG0\_PLLAUTOMODE\_ENABLE*

**TLV320\_IS\_VALID\_BCLKFSYNCGATE(gate)**

BCLK and FSYNC clock gate setting validness test.

**Parameters**

- **gate** – [in] ASI master slave configuration to test. Accepted values:
  - *TLV320\_MSTCFG0\_BCLKFSYNCGATE\_DISABLE*
  - *TLV320\_MSTCFG0\_BCLKFSYNCGATE\_ENABLE*

**TLV320\_IS\_VALID\_FSMODE(freq)**

Sample rate setting validness test.

**Parameters**

- **freq** – [in] ASI master slave configuration to test. Accepted values:
  - *TLV320\_MSTCFG0\_FSMODE\_48KHZ*
  - *TLV320\_MSTCFG0\_FSMODE\_44\_1KHZ*

**TLV320\_IS\_VALID\_MCLK(freq)**

MCLK frequency validness test.

**Parameters**

- **freq** – [in] Master clk to test. Accepted values:
  - *TLV320\_MSTCFG0\_MCLK\_12\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_12\_288\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_13\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_16\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_19\_2\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_19\_68\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_24\_MHZ*
  - *TLV320\_MSTCFG0\_MCLK\_24\_576\_MHZ*

**TLV320\_MSTCFG1\_FSRATE\_7\_35\_OR\_8\_KHZ**

7.35 kHz or 8 kHz

**TLV320\_MSTCFG1\_FSRATE\_14\_7\_OR\_16\_KHZ**

14.7 kHz or 16 kHz

**TLV320\_MSTCFG1\_FSRATE\_22\_05\_OR\_24\_KHZ**

22.05 kHz or 24 kHz

**TLV320\_MSTCFG1\_FSRATE\_29\_4\_OR\_32\_KHZ**

29.4 kHz or 32 kHz

**TLV320\_MSTCFG1\_FSRATE\_44\_1\_OR\_48\_KHZ**

44.1 kHz or 48 kHz

**TLV320\_MSTCFG1\_FSRATE\_88\_2\_OR\_96\_KHZ**

88.2 kHz or 96 kHz

**TLV320\_MSTCFG1\_FSRATE\_176\_4\_OR\_192\_KHZ**

176.4 kHz or 192 kHz

**TLV320\_MSTCFG1\_FSRATE\_352\_8\_OR\_384\_KHZ**

352.8 kHz or 384 kHz

**TLV320\_MSTCFG1\_FSRATE\_705\_6\_OR\_768\_KHZ**

705.6 kHz or 768 kHz

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_16**

Ratio of 16.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_24**

Ratio of 24.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_32**

Ratio of 32.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_48**

Ratio of 48.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_64**

Ratio of 64.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_96**

Ratio of 96.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_128**

Ratio of 128.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_192**

Ratio of 192.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_256**

Ratio of 256.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_384**

Ratio of 384.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_512**

Ratio of 512.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_1024**

Ratio of 1024.

**TLV320\_MSTCFG1\_FSBCLK\_RATIO\_2048**

Ratio of 2048.

**TLV320\_IS\_VALID\_FSRATE(freq)**

ASI bus sample rate validness test.

**Parameters**

- **freq** – [in] Frequency to test. Accepted values:
  - *TLV320\_MSTCFG1\_FSRATE\_7\_35\_OR\_8\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_14\_7\_OR\_16\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_22\_05\_OR\_24\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_29\_4\_OR\_32\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_44\_1\_OR\_48\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_88\_2\_OR\_96\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_176\_4\_OR\_192\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_352\_8\_OR\_384\_KHZ*
  - *TLV320\_MSTCFG1\_FSRATE\_705\_6\_OR\_768\_KHZ*

**TLV320\_IS\_VALID\_FSBCLK\_RATIO(ratio)**

BCLK to FSYNC frequency ratio validness test.

**Parameters**

- **ratio** – [in] FS/BCLK Ratio to test. Accepted values:
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_16*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_24*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_32*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_48*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_64*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_96*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_128*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_192*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_256*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_384*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_512*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_1024*
  - *TLV320\_MSTCFG1\_FSBCLK\_RATIO\_2048*

**TLV320\_CLKSRC\_AUDIOROOT\_BCLK**

BCLK is used as the audio root clock source.

**TLV320\_CLKSRC\_AUDIOROOT\_MCLK**

MCLK is used as the audio root clock source.

**TLV320\_CLKSRC\_MCLK\_MODE\_FREQ\_SEL**

MCLK frequency is based on the MCLK\_FREQ\_SEL.

**TLV320\_CLKSRC\_MCLK\_MODE\_FSYNC\_RATIO**

MCLK frequency is specified as a multiple of FSYNC in the MCLK\_RATIO\_SEL configuration.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_64**

Set a MCLK to FSYNC ratio to 64.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_256**

Set a MCLK to FSYNC ratio to 256.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_384**

Set a MCLK to FSYNC ratio to 384.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_512**

Set a MCLK to FSYNC ratio to 512.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_768**

Set a MCLK to FSYNC ratio to 768.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_1024**

Set a MCLK to FSYNC ratio to 1024.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_1536**

Set a MCLK to FSYNC ratio to 1536.

**TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_2304**

Set a MCLK to FSYNC ratio to 2304.

**TLV320\_IS\_VALID\_AUDIOROOT\_CLK\_SRC(bclk)**

Audio root clock source setting validness test.

**Parameters**

- **bclk** – [in] clock source to test. Accepted values:
  - *TLV320\_CLKSRC\_AUDIOROOT\_BCLK*
  - *TLV320\_CLKSRC\_AUDIOROOT\_MCLK*

**TLV320\_IS\_VALID\_MCLK\_FREQ\_MODE(mode)**

Audio root clock source setting validness test.

**Parameters**

- **bclk** – [in] clock source to test. Accepted values:
  - *TLV320\_CLKSRC\_MCLK\_MODE\_FREQ\_SEL*
  - *TLV320\_CLKSRC\_MCLK\_MODE\_FSYNC\_RATIO*

**TLV320\_IS\_VALID\_MCLK\_RATIO\_SEL(ratio)**

Audio root clock source setting validness test.

**Parameters**

- **ratio** – [in] clock source to test. Accepted values:
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_64*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_256*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_384*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_512*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_768*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_1024*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_1536*
  - *TLV320\_CLKSRC\_MCLK\_FS\_RATIO\_2304*

**TLV320\_IS\_VALID\_CLK\_MODE\_CFG(mode)***tlv320\_clk\_mode\_t* structure validness test.**Parameters**

- **mode** – [in] *tlv320\_clk\_mode\_t* structure to test.

**TLV320\_IS\_VALID\_CLK\_FS\_CFG(fs)***tlv320\_fs\_t* structure validness test.**Parameters**

- **fs** – [in] *tlv320\_fs\_t* structure to test.

**TLV320\_IS\_VALID\_CLK\_MCLK\_CFG(mclk)***tlv320\_mclk\_t* structure validness test.**Parameters**

- **mclk** – [in] *tlv320\_mclk\_t* structure to test.

**TLV320\_IS\_VALID\_CLK\_CFG(clk)***tlv320\_clk\_cfg\_t* structure validness test.**Parameters**

- **clk** – [in] *tlv320\_clk\_cfg\_t* structure to test.

**TLV320\_GPOx\_CFG\_DISABLED**

GPOx is disabled.

**TLV320\_GPOx\_CFG\_GPO**

GPOx is configured as a general-purpose output (GPO)

**TLV320\_GPOx\_CFG\_DIO**

GPOx is configured as a device interrupt output (IRQ)

**TLV320\_GPOx\_CFG\_ASI**

GPOx is configured as a secondary ASI output (SDOUT2)

**TLV320\_GPOx\_CFG\_PDMCLK**

GPOx is configured as a PDM clock output (PDMCLK)

**TLV320\_GPOx\_DRV\_HI\_Z**

Hi-Z output.

**TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_ACTIVE\_HIGH**

Drive active low and active high.

**TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_WEAK\_HIGH**

Drive active low and weak high.

**TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_HI\_Z**

Drive active low and Hi-Z.

**TLV320\_GPOx\_DRV\_WEAK\_LOW\_ACTIVE\_HIGH**

Drive weak low and active high.

**TLV320\_GPOx\_DRV\_HI\_Z\_ACTIVE\_HIGH**

Drive Hi-Z and active high.

**TLV320\_IS\_VALID\_GPO\_CFG(cfg)**

TLV320 General purpose output configuration validness test.

**Parameters**

- **cfg** – [in] General purpose output configuration to test. Accepted values:
  - *TLV320\_GPOx\_CFG\_DISABLED*
  - *TLV320\_GPOx\_CFG\_GPO*
  - *TLV320\_GPOx\_CFG\_DIO*
  - *TLV320\_GPOx\_CFG\_ASI*
  - *TLV320\_GPOx\_CFG\_PDMCLK*

**TLV320\_IS\_VALID\_GPO\_DRV(drv)**

TLV320 General purpose output drive validness test.

**Parameters**

- **drv** – [in] General purpose output drive to test. Accepted values :
  - *TLV320\_GPOx\_DRV\_HI\_Z*
  - *TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_ACTIVE\_HIGH*
  - *TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_WEAK\_HIGH*
  - *TLV320\_GPOx\_DRV\_ACTIVE\_LOW\_HI\_Z*
  - *TLV320\_GPOx\_DRV\_WEAK\_LOW\_ACTIVE\_HIGH*
  - *TLV320\_GPOx\_DRV\_WEAK\_LOW\_ACTIVE\_HIGH*

**TLV320\_GPIx\_CFG\_DISABLED**

GPOx is disabled.

**TLV320\_GPIx\_CFG\_GPI**

GPOx is configured as a general-purpose output (GPO)

**TLV320\_GPIx\_CFG\_MCLK**

GPOx is configured as a device interrupt output (IRQ)

**TLV320\_GPIx\_CFG\_SDIN**

GPOx is configured as a secondary ASI output (SDOUT2)

**TLV320\_GPIx\_CFG\_PDMIN1**

GPOx is configured as a PDM clock output (PDMCLK)

**TLV320\_GPIx\_CFG\_PDMIN2**

GPOx is configured as a PDM clock output (PDMCLK)

**TLV320\_GPIx\_CFG\_PDMIN3**

GPOx is configured as a PDM clock output (PDMCLK)

**TLV320\_GPIx\_CFG\_PDMIN4**

GPOx is configured as a PDM clock output (PDMCLK)

**TLV320\_IS\_VALID\_GPI\_CFG(cfg)**

TLV320 General purpose input configuration validness test.

**Parameters**

- **cfg** – [in] General purpose input configuration to test. Accepted values:
  - *TLV320\_GPIx\_CFG\_DISABLED*
  - *TLV320\_GPIx\_CFG\_GPI*
  - *TLV320\_GPIx\_CFG\_MCLK*
  - *TLV320\_GPIx\_CFG\_SDIN*
  - *TLV320\_GPIx\_CFG\_PDMIN1*
  - *TLV320\_GPIx\_CFG\_PDMIN2*
  - *TLV320\_GPIx\_CFG\_PDMIN3*
  - *TLV320\_GPIx\_CFG\_PDMIN4*

**TLV320\_BIASCFG\_BIAS\_VREF**

Microphone bias is set to VREF (2.750 V, 2.500 V, or 1.375 V)

**TLV320\_BIASCFG\_BIAS\_1\_096\_VREF**

Microphone bias is set to  $VREF \times 1.096$  (3.014 V, 2.740 V, or 1.507 V)

**TLV320\_BIASCFG\_BIAS\_AVDD**

Microphone bias is set to AVDD.

**TLV320\_BIASCFG\_ADCFScale\_VREF\_2\_75**

VREF is set to 2.75 V to support 2 VRMS for the differential input or 1 VRMS for the single-ended input.

**TLV320\_BIASCFG\_ADCFScale\_VREF\_2\_5**

VREF is set to 2.5 V to support 1.818 VRMS for the differential input or 0.909 VRMS for the single-ended input.

**TLV320\_BIASCFG\_ADCFSSCALE\_VREF\_1\_375**

VREF is set to 1.375 V to support 1 VRMS for the differential input or 0.5 VRMS for the single-ended input.

**TLV320\_IS\_VALID\_ADC\_FSCALE(fscale)**

TLV320 ADC full scale validness test.

**Parameters**

- **fscale** – [in] ADC Full scale configuration to test. Accepted value:
  - *TLV320\_BIASCFG\_ADCFSSCALE\_VREF\_2\_75*
  - *TLV320\_BIASCFG\_ADCFSSCALE\_VREF\_2\_5*
  - *TLV320\_BIASCFG\_ADCFSSCALE\_VREF\_1\_375*

**TLV320\_IS\_VALID\_MICBIAS(bias)**

TLV320 Microphone bias validness test.

**Parameters**

- **bias** – [in] Microphone bias configuration to test. Accepted value:
  - *TLV320\_BIASCFG\_BIAS\_VREF*
  - *TLV320\_BIASCFG\_BIAS\_1\_096\_VREF*
  - *TLV320\_BIASCFG\_BIAS\_AVDD*

**TLV320\_CHx\_CFG0\_INTYP\_MICRO**

Configure the channel as a microphone input.

**TLV320\_CHx\_CFG0\_INTYP\_LINE**

Configure the channel as a line input.

**TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_DIFF**

Configure the channel as a analog differential input

**TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_SINGLE**

Configure the channel as a analog single-ended input.

**TLV320\_CHx\_CFG0\_INSRC\_DIGITAL**

Configure the channel as a digital input.

**TLV320\_CHx\_CFG0\_AC\_COUPLED**

AC-coupled input.

**TLV320\_CHx\_CFG0\_DC\_COUPLED**

DC-coupled input.

**TLV320\_CHx\_CFG0\_Z\_2\_5\_KOHMS**

Typical 2.5-kOhm input impedance.

**TLV320\_CHx\_CFG0\_Z\_10\_KOHMS**

Typical 10-kOhm input impedance.

**TLV320\_CHx\_CFG0\_Z\_20\_KOHMS**

Typical 20-kOhm input impedance.

**TLV320\_CHx\_CFG0\_DRE\_AGC\_DISABLED**

DRE and AGC disabled.

**TLV320\_CHx\_CFG0\_DRE\_AGC\_ENABLE**

DRE or AGC enabled based on the configuration of bit 3 in register DSP\_CFG1.

**TLV320\_CHx\_GAIN\_MIN**

Minimal acceptable channel gain (0dB)

**TLV320\_CHx\_GAIN\_MAX**

Maximal acceptable channel gain (42dB)

**TLV320\_CHx\_GAIN\_DEFAULT**

Default channel gain.

**TLV320\_CHx\_VOLUME\_MIN**

Minimal volume (dB)

**TLV320\_CHx\_VOLUME\_MAX**

Maximal volume (dB)

**TLV320\_CHx\_VOLUME\_DEFAULT**

Default volume (dB)

**TLV320\_CHx\_VOLUME\_CODE\_0\_DB**

Code value to set the digital volume to 0dB.

**TLV320\_CHx\_VOLUME\_STEP**

Step between two volume settings.

**TLV320\_CHx\_VOLUME\_MUTED**

Float code to mute the channel.

**TLV320\_CHx\_GAIN\_CALIB\_MIN**

Minimal gain calibration (dB)

**TLV320\_CHx\_GAIN\_CALIB\_MAX**

Maximal gain calibration (dB)

**TLV320\_CHx\_GAIN\_CALIB\_DEFAULT**

Default gain calibration (dB)

**TLV320\_CHx\_GAIN\_CALIB\_NB**

Number of gain calibration settings.

**TLV320\_CHx\_GAIN\_CALIB\_CODE\_0\_DB**

Value to set to get a calibration gain of 0dB

**TLV320\_CHx\_GAIN\_CALIB\_STEP**

Step between two gain calibration settings.

**TLV320\_CHx\_PHASE\_CALIB\_MIN**

Minimal gain calibration (n cycles)

**TLV320\_CHx\_PHASE\_CALIB\_MAX**

Maximal gain calibration (n cycles)

**TLV320\_CHx\_PHASE\_CALIB\_DEFAULT**

Default phase calibration (n cycles)

**TLV320\_IS\_VALID\_CH\_GAIN(gain)**

Channel gain validness test.

**Parameters**

- **gain** – [in] Value to test. Accepted values:
  - Between *TLV320\_CHx\_GAIN\_MIN* and *TLV320\_CHx\_GAIN\_MAX*

**TLV320\_IS\_VALID\_CH\_VOLUME(vol)**

Channel volume validness test.

**Parameters**

- **vol** – [in] Value to test. Accepted values:
  - Between *TLV320\_CHx\_VOLUME\_MUTED* and *TLV320\_CHx\_VOLUME\_MAX*

**TLV320\_IS\_VALID\_CH\_GAIN\_CALIB(gain)**

Channel gain calibration validness test.

**Parameters**

- **gain** – [in] Value to test. Accepted values:
  - between *TLV320\_CHx\_GAIN\_CALIB\_MIN* and *TLV320\_CHx\_GAIN\_CALIB\_MAX*

**TLV320\_IS\_VALID\_CH\_PHASE\_CALIB(phase)**

Channel phase calibration validness test.

**Parameters**

- **phase** – [in] Value to test. Accepted values:
  - between *TLV320\_CHx\_PHASE\_CALIB\_MIN* and *TLV320\_CHx\_PHASE\_CALIB\_MAX*

**TLV320\_IS\_VALID\_INTYP(intyp)**

TLV320 input type validness test.

**Parameters**

- **intyp** – [in] Value to test. Accepted values:
  - *TLV320\_CHx\_CFG0\_INTYP\_MICRO*
  - *TLV320\_CHx\_CFG0\_INTYP\_LINE*

**TLV320\_IS\_VALID\_INSRC(insrc)**

TLV320 input configuration validness test.

**Parameters**

- **insrc** – [in] Value to test. Accepted values:

- *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_DIFF*
- *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_SINGLE*
- *TLV320\_CHx\_CFG0\_INSRC\_DIGITAL*

**TLV320\_IS\_VALID\_ANALOG\_CH**(insrc)

TLV320 input analog configuration validness test.

**Parameters**

- **an** – [in] Value to test. Accepted values:
  - *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_DIFF*
  - *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_SINGLE*

**TLV320\_IS\_VALID\_COUPLING**(coupling)

TLV320 input coupling validness test.

**Parameters**

- **coupling** – [in] Value to test. Accepted values:
  - *TLV320\_CHx\_CFG0\_AC\_COUPLED*
  - *TLV320\_CHx\_CFG0\_DC\_COUPLED*

**TLV320\_IS\_VALID\_INPUT\_Z**(z)

TLV320 input impedance validness test.

**Parameters**

- **z** – [in] Value to test. Accepted values:
  - *TLV320\_CHx\_CFG0\_Z\_2\_5\_KOHMS*
  - *TLV320\_CHx\_CFG0\_Z\_10\_KOHMS*
  - *TLV320\_CHx\_CFG0\_Z\_20\_KOHMS*

**TLV320\_IS\_VALID\_DRE\_AGC**(dreagc)

TLV320 input DRE & AGC validness test.

**Parameters**

- **dreagc** – [in] Value to test. Accepted values:
  - *TLV320\_CHx\_CFG0\_DRE\_AGC\_DISABLED*
  - *TLV320\_CHx\_CFG0\_DRE\_AGC\_ENABLE*

**TLV320\_IN\_CH\_EN\_DISABLED**

Value to disable an input channel.

**TLV320\_IN\_CH\_EN\_ENABLED**

Value to enable an input channel.

**TLV320\_ASI\_OUT\_CH\_EN\_TRISTATE**

Value that disable an output channel.

**TLV320\_ASI\_OUT\_CH\_EN\_ENABLED**

Value that enable an output channel.

**TLV320\_PWR\_CFG\_MICBIAS\_POWER\_DOWN**

Power down MICBIAS.

**TLV320\_PWR\_CFG\_MICBIAS\_POWER\_UP**

Power up MICBIAS.

**TLV320\_PWR\_CFG\_ADC\_POWER\_DOWN**

Power down all ADC and PDM channels.

**TLV320\_PWR\_CFG\_ADC\_POWER\_UP**

Power up all enabled ADC and PDM channels

**TLV320\_PWR\_CFG\_PLL\_POWER\_DOWN**

Power down the PLL

**TLV320\_PWR\_CFG\_PLL\_POWER\_UP**

Power up the PLL.

**TLV320\_PWR\_CFG\_DYN\_CH\_PUPD\_DISABLED**

Channel power-up, power-down is not supported if any channel recording is on.

**TLV320\_PWR\_CFG\_DYN\_CH\_PUPD\_ENABLED**

Channel can be powered up or down individually, even if channel recording is on.

**TLV320\_PWR\_CFG\_DYN\_MAXCH\_SEL\_1\_2**

Channel 1 and channel 2 are used with dynamic channel power-up, power-down feature enabled.

**TLV320\_PWR\_CFG\_DYN\_MAXCH\_SEL\_1\_4**

Channel 1 and channel 4 are used with dynamic channel power-up, power-down feature enabled.

**TLV320\_PWR\_CFG\_DYN\_MAXCH\_SEL\_1\_6**

Channel 1 and channel 6 are used with dynamic channel power-up, power-down feature enabled.

**TLV320\_PWR\_CFG\_DYN\_MAXCH\_SEL\_1\_8**

Channel 1 and channel 8 are used with dynamic channel power-up, power-down feature enabled.

## Enums

**enum tlv320\_state\_t**

State command definition.

*Values:*

enumerator **TLV320\_DISABLE** = 0

Generic value used to disable a feature.

enumerator **TLV320\_ENABLE** = 1

Generic value used to enable a feature.

---

enum **tlv320\_hw\_mode\_t**  
Device's hardware functional mode.

*Values:*

enumerator **TLV320\_SHUTDOWN** = 0  
Generic value used to disable a feature.

enumerator **TLV320\_ACTIVE** = 1  
Generic value used to enable a feature.

enum **tlv320\_config\_register\_t**  
Enumeration of TLV320's configuration registers.

*Values:*

enumerator **TLV320\_PAGE\_CFG** = 0x00  
Device page register address.

enumerator **TLV320\_SW\_RESET** = 0x01  
Software reset register.

enumerator **TLV320\_SLEEP\_CFG** = 0x02  
Sleep mode register.

enumerator **TLV320\_SHDN\_CFG** = 0x05  
Shutdown configuration register.

enumerator **TLV320\_ASI\_CFG0** = 0x07  
ASI configuration register 0.

enumerator **TLV320\_ASI\_CFG1** = 0x08  
ASI configuration register 1.

enumerator **TLV320\_ASI\_CFG2** = 0x09  
ASI configuration register 2.

enumerator **TLV320\_ASI\_CH1** = 0x0B  
Channel 1 ASI slot configuration register.

enumerator **TLV320\_ASI\_CH2** = 0x0C  
Channel 2 ASI slot configuration register.

enumerator **TLV320\_ASI\_CH3** = 0x0D  
Channel 3 ASI slot configuration register.

enumerator **TLV320\_ASI\_CH4** = 0x0E  
Channel 4 ASI slot configuration register.

enumerator **TLV320\_ASI\_CH5** = 0x0F  
Channel 5 ASI slot configuration register.

- enumerator **TLV320\_ASI\_CH6** = 0x10  
Channel 6 ASI slot configuration register.
- enumerator **TLV320\_ASI\_CH7** = 0x11  
Channel 7 ASI slot configuration register.
- enumerator **TLV320\_ASI\_CH8** = 0x12  
Channel 8 ASI slot configuration register.
- enumerator **TLV320\_MST\_CFG0** = 0x13  
ASI master mode configuration register 0.
- enumerator **TLV320\_MST\_CFG1** = 0x14  
ASI master mode configuration register 1.
- enumerator **TLV320\_ASI\_STS** = 0x15  
ASI bus clock monitor status register.
- enumerator **TLV320\_CLK\_SRC** = 0x16  
Clock source configuration register 0.
- enumerator **TLV320\_PDMCLK\_CFG** = 0x1F  
PDM clock generation configuration register.
- enumerator **TLV320\_PDMIN\_CFG** = 0x20  
PDM DINx sampling edge register.
- enumerator **TLV320\_GPIO\_CFG0** = 0x21  
GPIO configuration register 0.
- enumerator **TLV320\_GPO\_CFG0** = 0x22  
GPO configuration register 0.
- enumerator **TLV320\_GPO\_CFG1** = 0x23  
GPO configuration register 1.
- enumerator **TLV320\_GPO\_CFG2** = 0x24  
GPO configuration register 2.
- enumerator **TLV320\_GPO\_CFG3** = 0x25  
GPO configuration register 3.
- enumerator **TLV320\_GPO\_VAL** = 0x29  
GPIO, GPO output value register.
- enumerator **TLV320\_GPIO\_MON** = 0x2A  
GPIO monitor value register.
- enumerator **TLV320\_GPI\_CFG0** = 0x2B  
GPI configuration register 0.

enumerator **TLV320\_GPI\_CFG1** = 0x2C  
GPI configuration register 1.

enumerator **TLV320\_GPI\_MON** = 0x2F  
GPI monitor value register.

enumerator **TLV320\_INT\_CFG** = 0x32  
Interrupt configuration register.

enumerator **TLV320\_INT\_MASK0** = 0x33  
Interrupt mask register 0.

enumerator **TLV320\_INT\_LTCH0** = 0x36  
Latched interrupt readback register 0.

enumerator **TLV320\_BIAS\_CFG** = 0x3B  
Bias and ADC configuration register.

enumerator **TLV320\_CH1\_CFG0** = 0x3C  
Channel 1 configuration register 0.

enumerator **TLV320\_CH1\_CFG1** = 0x3D  
Channel 1 configuration register 1.

enumerator **TLV320\_CH1\_CFG2** = 0x3E  
Channel 1 configuration register 2.

enumerator **TLV320\_CH1\_CFG3** = 0x3F  
Channel 1 configuration register 3.

enumerator **TLV320\_CH1\_CFG4** = 0x40  
Channel 1 configuration register 4.

enumerator **TLV320\_CH2\_CFG0** = 0x41  
Channel 2 configuration register 0.

enumerator **TLV320\_CH2\_CFG1** = 0x42  
Channel 2 configuration register 1.

enumerator **TLV320\_CH2\_CFG2** = 0x43  
Channel 2 configuration register 2.

enumerator **TLV320\_CH2\_CFG3** = 0x44  
Channel 2 configuration register 3.

enumerator **TLV320\_CH2\_CFG4** = 0x45  
Channel 2 configuration register 4.

enumerator **TLV320\_CH3\_CFG0** = 0x46  
Channel 3 configuration register 0.

enumerator **TLV320\_CH3\_CFG1** = 0x47  
Channel 3 configuration register 1.

enumerator **TLV320\_CH3\_CFG2** = 0x48  
Channel 3 configuration register 2.

enumerator **TLV320\_CH3\_CFG3** = 0x49  
Channel 3 configuration register 3.

enumerator **TLV320\_CH3\_CFG4** = 0x4A  
Channel 3 configuration register 4.

enumerator **TLV320\_CH4\_CFG0** = 0x4B  
Channel 4 configuration register 0.

enumerator **TLV320\_CH4\_CFG1** = 0x4C  
Channel 4 configuration register 1.

enumerator **TLV320\_CH4\_CFG2** = 0x4D  
Channel 4 configuration register 2.

enumerator **TLV320\_CH4\_CFG3** = 0x4E  
Channel 4 configuration register 3.

enumerator **TLV320\_CH4\_CFG4** = 0x4F  
Channel 4 configuration register 4.

enumerator **TLV320\_CH5\_CFG2** = 0x52  
Channel 5 (PDM only) configuration register 2.

enumerator **TLV320\_CH5\_CFG3** = 0x53  
Channel 5 (PDM only) configuration register 3.

enumerator **TLV320\_CH5\_CFG4** = 0x54  
Channel 5 (PDM only) configuration register 4.

enumerator **TLV320\_CH6\_CFG2** = 0x57  
Channel 6 (PDM only) configuration register 2.

enumerator **TLV320\_CH6\_CFG3** = 0x58  
Channel 6 (PDM only) configuration register 3.

enumerator **TLV320\_CH6\_CFG4** = 0x59  
Channel 6 (PDM only) configuration register 4.

enumerator **TLV320\_CH7\_CFG2** = 0x5C  
Channel 7 (PDM only) configuration register 2.

enumerator **TLV320\_CH7\_CFG3** = 0x5D  
Channel 7 (PDM only) configuration register 3.

---

enumerator **TLV320\_CH7\_CFG4** = 0x5E  
     Channel 7 (PDM only) configuration register 4.

enumerator **TLV320\_CH8\_CFG2** = 0x61  
     Channel 8 (PDM only) configuration register 2.

enumerator **TLV320\_CH8\_CFG3** = 0x62  
     Channel 8 (PDM only) configuration register 3.

enumerator **TLV320\_CH8\_CFG4** = 0x63  
     Channel 8 (PDM only) configuration register 4.

enumerator **TLV320\_DSP\_CFG0** = 0x6B  
     DSP configuration register 0.

enumerator **TLV320\_DSP\_CFG1** = 0x6C  
     DSP configuration register 1.

enumerator **TLV320\_DRE\_CFG0** = 0x6D  
     DRE configuration register 0.

enumerator **TLV320\_AGC\_CFG0** = 0x70  
     AGC configuration register 0.

enumerator **TLV320\_IN\_CH\_EN** = 0x73  
     Input channel enable configuration register.

enumerator **TLV320\_ASI\_OUT\_CH\_EN** = 0x74  
     ASI output channel enable configuration register.

enumerator **TLV320\_PWR\_CFG** = 0x75  
     Power up configuration register.

enumerator **TLV320\_DEV\_STS0** = 0x76  
     Device status value register 0.

enumerator **TLV320\_DEV\_STS1** = 0x77  
     Device status value register 1.

enumerator **TLV320\_I2C\_CKSUM** = 0x7E  
     I2C checksum register.

enum **tlv320\_asi\_format\_t**  
     Enumeration of ASI protocol format.  
     Values:  
         enumerator **TLV320\_TDM** = *TLV320\_ASI\_CFG0\_ASI\_FORMAT\_TDM*  
             Time division multiplexing mode.

enumerator **TLV320\_I2S** = *TLV320\_ASI\_CFG0\_ASI\_FORMAT\_I2S*  
Inter IC sound mode

enumerator **TLV320\_LJ** = *TLV320\_ASI\_CFG0\_ASI\_FORMAT\_LJ*  
Left-justified mode.

enum **tlv320\_asi\_wlen\_t**  
Enumeration of audio output channel data word-length.

*Values:*

enumerator **TLV320\_WLEN\_16** = *TLV320\_ASI\_CFG0\_ASI\_WLEN\_16BITS*  
Output channel data word-length set to 16 bits.

enumerator **TLV320\_WLEN\_20** = *TLV320\_ASI\_CFG0\_ASI\_WLEN\_20BITS*  
Output channel data word-length set to 20 bits.

enumerator **TLV320\_WLEN\_24** = *TLV320\_ASI\_CFG0\_ASI\_WLEN\_24BITS*  
Output channel data word-length set to 24 bits.

enumerator **TLV320\_WLEN\_32** = *TLV320\_ASI\_CFG0\_ASI\_WLEN\_32BITS*  
Output channel data word-length set to 32 bits.

enum **tlv320\_asi\_fsync\_pol\_t**  
Enumeration of audio serial data interface bus frame synchronization polarity.

*Values:*

enumerator **TLV320\_FSYNCPOL\_NORMAL** = *TLV320\_ASI\_CFG0\_FSYNC\_POL\_NORMAL*  
Default polarity as per standard protocol.

enumerator **TLV320\_FSYNCPOL\_INVERTED** = *TLV320\_ASI\_CFG0\_FSYNC\_POL\_INVERTED*  
Inverted polarity with respect to standard protocol.

enum **tlv320\_asi\_bclk\_pol\_t**  
Enumeration of audio serial data interface bus bit clock polarity.

*Values:*

enumerator **TLV320\_BCLKPOL\_NORMAL** = *TLV320\_ASI\_CFG0\_BCLK\_POL\_NORMAL*  
Default polarity as per standard protocol.

enumerator **TLV320\_BCLKPOL\_INVERTED** = *TLV320\_ASI\_CFG0\_BCLK\_POL\_INVERTED*  
Inverted polarity with respect to standard protocol.

enum **tlv320\_asi\_tx\_edge\_t**  
Enumeration of ASI data output transmit edge.

*Values:*

enumerator **TLV320\_TXEDGE\_NORMAL** = *TLV320\_ASI\_CFG0\_TX\_EDGE\_NORMAL*  
Default edge as per the protocol configuration setting : *tlv320\_asi\_bclk\_pol\_t*.

---

enumerator **TLV320\_TXEDGE\_INVERTED** = *TLV320\_ASI\_CFG0\_TX\_EDGE\_INVERTED*  
Inverted following edge (half cycle delay) with respect to the default edge setting.

enum **tlv320\_asi\_tx\_fill\_t**

Enumeration of ASI data output selection for any unused cycles.

*Values:*

enumerator **TLV320\_TXFILL\_ZERO** = *TLV320\_ASI\_CFG0\_TX\_FILL\_ZERO*  
Always transmit 0 for unused cycles.

enumerator **TLV320\_TXFILL\_HI\_Z** = *TLV320\_ASI\_CFG0\_TX\_FILL\_HI\_Z*  
Always use Hi-Z for unused cycles.

enum **tlv320\_tx\_lsb\_t**

Enumeration of ASI data output (on the primary and secondary data pin) for LSB transmissions

*Values:*

enumerator **TLV320\_TX\_LSB\_FULLCYCLE** = *TLV320\_ASI\_CFG1\_TX\_LSB\_FULL\_CYCLE*  
Transmit the LSB for a full cycle.

enumerator **TLV320\_TX\_LSB\_HALFCYCLE** = *TLV320\_ASI\_CFG1\_TX\_LSB\_HALF\_CYCLE*  
Transmit the LSB for the first half cycle and Hi-Z for the second half cycle.

enum **tlv320\_tx\_keeper\_t**

Enumeration of ASI data output (on the primary and secondary data pin) bus keeper.

*Values:*

enumerator **TLV320\_TX\_KEEPER\_DISABLED** = *TLV320\_ASI\_CFG1\_TX\_KEEPER\_DISABLED*  
Bus keeper is always disabled.

enumerator **TLV320\_TX\_KEEPER\_EN** = *TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED*  
Bus keeper is always enabled.

enumerator **TLV320\_TX\_KEEPER\_1CYCLE** = *TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_CYCLE*  
Bus keeper is enabled during LSB transmissions only for one cycle.

enumerator **TLV320\_TX\_KEEPER\_1\_5\_CYCLES** =  
*TLV320\_ASI\_CFG1\_TX\_KEEPER\_ENABLED\_1\_5\_CYCLES*

Bus keeper is enabled during LSB transmissions only for one and half cycles.

enum **tlv320\_ch\_intyp\_t**

Channel input type possible configurations.

*Values:*

enumerator **TLV320\_INTYP\_MICRO** = *TLV320\_CHx\_CFG0\_INTYP\_MICRO*

enumerator **TLV320\_INTYP\_LINE** = *TLV320\_CHx\_CFG0\_INTYP\_LINE*

enum **tlv320\_ch\_insrc\_t**

Channel input possible configuration.

*Values:*

enumerator **TLV320\_INSRC\_ANALOG\_DIFF** = *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_DIFF*

enumerator **TLV320\_INSRC\_ANALOG\_SINGLE** = *TLV320\_CHx\_CFG0\_INSRC\_ANALOG\_SINGLE*

enumerator **TLV320\_INSRC\_DIGITAL** = *TLV320\_CHx\_CFG0\_INSRC\_DIGITAL*

enum **tlv320\_ch\_input\_coupling\_t**

Channel input coupling possible configurations.

*Values:*

enumerator **TLV320\_INPUT\_AC\_COUPLED** = *TLV320\_CHx\_CFG0\_AC\_COUPLED*

enumerator **TLV320\_INPUT\_DC\_COUPLED** = *TLV320\_CHx\_CFG0\_DC\_COUPLED*

enum **tlv320\_ch\_input\_z\_t**

Channel input impedance possible configurations.

*Values:*

enumerator **TLV320\_INPUT\_Z\_2\_5\_KOHM** = *TLV320\_CHx\_CFG0\_Z\_2\_5\_KOHMS*

enumerator **TLV320\_INPUT\_Z\_10\_KOHM** = *TLV320\_CHx\_CFG0\_Z\_10\_KOHMS*

enumerator **TLV320\_INPUT\_Z\_20\_KOHM** = *TLV320\_CHx\_CFG0\_Z\_20\_KOHMS*

enum **tlv320\_ch\_input\_agcdre\_t**

Channel DRE & AGC possible configurations.

*Values:*

enumerator **TLV320\_AGCDRE\_DISABLED** = *TLV320\_CHx\_CFG0\_DRE\_AGC\_DISABLED*

enumerator **TLV320\_AGCDRE\_ENABLED** = *TLV320\_CHx\_CFG0\_DRE\_AGC\_ENABLE*

enum **tlv320\_ch\_output\_line\_t**

Channel output possible configurations.

*Values:*

enumerator **TLV320\_OUTPUT\_PRIMARY** = *TLV320\_ASI\_CHx\_OUTPUT\_PRIMARY*

enumerator **TLV320\_OUTPUT\_SECONDARY** = *TLV320\_ASI\_CHx\_OUTPUT\_SECONDARY*

enum **tlv320\_micbias\_t**

Microphone bias possible configurations.

*Values:*

enumerator **TLV320\_MICBIAS\_VREF** = *TLV320\_BIASCFG\_BIAS\_VREF*

enumerator **TLV320\_MICBIAS\_VREF\_1\_096** = *TLV320\_BIASCFG\_BIAS\_1\_096\_VREF*

enumerator **TLV320\_MICBIAS\_VREF\_AVDD** = *TLV320\_BIASCFG\_BIAS\_AVDD*

enum **tlv320\_afc\_fscale\_t**

ADC full scale possible configurations.

*Values:*

enumerator **TLV320\_ADC\_FSCALE\_VREF\_2\_75** = *TLV320\_BIASCFG\_ADCFSCALE\_VREF\_2\_75*

---

```
enumerator TLV320_ADC_FSCALE_VREF_2_5 = TLV320_BIASCFG_ADCFSCALE_VREF_2_5
enumerator TLV320_ADC_FSCALE_1_375 = TLV320_BIASCFG_ADCFSCALE_VREF_1_375
```

## Functions

`uint8_t pi_tlv320_write_config_register(pi_device_t *tlv320_dev, tlv320_config_register_t reg, uint8_t value)`

TLV320 write operation dedicated to configuration registers. This function automatically set the right page to write a configuration register.

### Parameters

- **tlv320\_dev** – [in] TLV320 device instance
- **reg** – [in] Configuration register to write.
- **value** – [in] Value to write into the register

**Returns** `uint8_t` Error management. `TLV320_NO_ERROR` if no error occurred.  
`TLV320_WRITE_ERROR` if an i2C write operation failed

`uint8_t pi_tlv320_read_config_register(pi_device_t *tlv320_dev, tlv320_config_register_t reg, uint8_t *read_value)`

TLV320 read operation dedicated to configuration registers. This function automatically set the right page before reading a configuration register.

### Parameters

- **tlv320\_dev** – [in] TLV320 device instance
- **reg** – [in] Configuration register to write.
- **read\_value** – [inout] Pointer to the value read from the register

**Returns** Error management. The function can return the following error codes:

- `TLV320_READ_ERROR` Failed to read register value.
- `TLV320_NO_ERROR` No error occurred.

`void pi_tlv320_conf_init(struct pi_tlv320_conf *conf)`

Initialize an tlv320 configuration with default values.

The structure containing the configuration must be kept alive until the device is opened. It can only be called from fabric-controller side.

### Parameters

- **conf** – [in] Pointer to the device configuration.

`int pi_tlv320_open(pi_device_t *tlv320_dev)`

TLV320 initialization. This function must be called after loading a configuration with `pi_open_from_conf`. These will set the module's configuration (`struct pi_tlv320_conf`) before initializing it.

### Parameters

- **tlv320\_dev** – [in] `pi_device` that store the tlv320 configuration

**Returns** `int` Error management.

`void pi_tlv320_close(pi_device_t *tlv320_dev)`

TLV320 deallocation.

**Parameters**

- **tlv320\_dev** – [in] TLV320 instance pointer

void **pi\_tlv320\_reset**(pi\_device\_t \*tlv320\_dev)

Shuts down the device, and restores all device configuration registers and programmable coefficients to their default values.

**Parameters**

- **tlv320\_dev** – [in] tlv320 device instance.

void **pi\_tlv320\_set\_sleep\_mode**(pi\_device\_t \*tlv320\_dev, uint8\_t mode)

Put the device into sleep or active mode.

---

**Note:** While putting the device into active, the analog supply selection is reseted to “internal” with AREG\_SELECT bit setted to *TLV320\_SLEEP\_CFG\_AREG\_SELECT\_INTERNAL*. In case mode is set to *TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_DISABLED*, The function insert a delay of *TLV320\_SLEEP\_MODE\_WAKEUP\_DELAY* to wait for TLV320 internal wake-up sequence.

---

**Parameters**

- **tlv320\_dev** – [in] TLV320 device instance pointer
- **mode** – [in] Mode in which to set the TLV320. It can be a value of :  
*TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_DISABLED* Set the device into active mode.  
*TLV320\_SLEEP\_CFG\_SLEEP\_ENZ\_ENABLED* Set the device into sleep mode.

void **pi\_tlv320\_set\_asi**(pi\_device\_t \*tlv320\_dev, *tlv320\_asi\_t* asi)

Program the device’s serial audio interface according to *tlv320\_asi\_t* data structure.

**Parameters**

- **tlv320\_dev** – [in] TLV320 instance pointer
- **asi** – [in] Audio serial interface to program into the device

uint8\_t **pi\_tlv320\_get\_asi**(pi\_device\_t \*tlv320\_dev, *tlv320\_asi\_t* \*asi)

Return the audio serial interface programmed into the device.

**Parameters**

- **tlv320\_dev** – [in] TLV320 instance pointer
- **asi** – [inout] Audio serial interface structure pointer

**Returns** Error management

- *TLV320\_NO\_ERROR* is everything went well
- *TLV320\_READ\_ERROR* if read operation failed

*tlv320\_asi\_t* **pi\_tlv320\_build\_default\_asi**(void)

Return a default, ready to be written structure about the TLV320 serial audio interface. Use *pi\_tlv320\_set\_asi* to send this configuration to the device.

**Parameters**

- **tlv320\_dev** – [in] TLV320 instance pointer

**Returns** *tlv320\_asi\_t* Default serial audio interface data structure.

---

*tlv320\_analog\_ch\_t* **pi\_tlv320\_build\_default\_analog\_ch**(uint8\_t channel\_id, uint8\_t slot\_id)  
 Return a default analog channel configuration structure ready to be written. It is configured as followed:

- a. Micro input
- b. Analog differential
- c. AC coupled
- d. 2.5 kOhm input impedance
- e. No Digital range enhancer nor Automatic gain controller activated
- f. The channel output is set on its primary output (SDOUT)
- g. Channel gain, volume, gain calib and phase calib are set to their default value.
- h. The channel input as the output slot are specified by parameters.

#### Parameters

- **tlv320\_dev** – [in] TLV320 instance pointer
- **channel\_id** – [in] Channel selection. It can be a value of :
  - *TLV320\_CHANNEL\_1* Channel 1 selected
  - *TLV320\_CHANNEL\_2* Channel 2 selected
  - *TLV320\_CHANNEL\_3* Channel 3 selected
  - *TLV320\_CHANNEL\_4* Channel 4 selected
- **slot\_id** – [in] Output slot selection. It can be a value between *TLV320\_TDM\_SLOT\_MIN* and *TLV320\_TDM\_SLOT\_MAX*.

**Returns** *tlv320\_analog\_ch\_t* Filled analog channel configuration.

*uint8\_t* **pi\_tlv320\_write\_analog\_ch**(pi\_device\_t \*tlv320\_dev, *tlv320\_analog\_ch\_t* channel)  
 Configure a device's channel as an analog one from a configuration. The structure *tlv320\_analog\_ch\_t* allows to set the following parameters :

- 1. Channel ID. See *TLV320\_IS\_VALID\_ANALOG\_CHANNEL\_ID* for accepted values.
- 2. Input type. See *TLV320\_IS\_VALID\_INTYP* for accepted values.
- 3. Input source. See *TLV320\_IS\_VALID\_INSRC* for accepted values.
- 4. Input coupling. See *TLV320\_IS\_VALID\_COUPLING* for accepted values.
- 5. Input impedance. See *TLV320\_IS\_VALID\_INPUT\_Z* for accepted values.
- 6. DRE and AGC setting. See *TLV320\_IS\_VALID\_DRE\_AGC* for accepted values.
- 7. Slot Id assignement. See *TLV320\_IS\_VALID\_TDM\_SLOT\_ID* for accepted values.
- 8. Output line. See *TLV320\_IS\_VALID\_ASI\_CHx\_OUTPUT* for accepted values.
- 9. Channel gain. See *TLV320\_IS\_VALID\_CH\_GAIN* for accepted values.
- 10. Channel Volume. See *TLV320\_IS\_VALID\_CH\_VOLUME* for accepted values.
- 11. Gain calibration. See *TLV320\_IS\_VALID\_CH\_GAIN\_CALIB* for accepted values.
- 12. Phase calibration. See *TLV320\_IS\_VALID\_CH\_PHASE\_CALIB* for accepted values.

**Parameters**

- **tlv320\_dev** – [inout] TLV320 instance pointer
- **channel** – [in] Channel configuration structure. It must already be initialized

**Returns Error** management. The function can return the following error codes:

- *TLV320\_READ\_ERROR* Failed to read GPI register value.
- *TLV320\_WRITE\_ERROR* Failed to write at least one register.
- *TLV320\_NO\_ERROR* No error occurred.

```
uint8_t pi_tlv320_read_analog_ch(pi_device_t *tlv320_dev, uint8_t channel_id, tlv320_analog_ch_t  
*an_channel)
```

Read an analog channel configuration from the device.

**Parameters**

- **tlv320\_dev** – [inout] TLV320 instance pointer
- **channel\_id** – [in] Analog channel to read
- **an\_channel** – [inout] Analog channel data structure pointer

**Returns Error** management. The function can return the following error codes:

- *TLV320\_READ\_ERROR* Failed to read register value
- *TLV320\_NO\_ERROR* No error occurred.

```
void pi_tlv320_set_bias_cfg(pi_device_t *tlv320_dev, tlv320_micbias_t micbias, tlv320_afc_fscale_t  
adc_fscale)
```

Configure the microphone bias and the ADC full scale.

**Parameters**

- **tlv320\_dev** – [inout] TLV320 instance pointer
- **micbias** – [in] Microphone bias value
- **adc\_fscale** – [in] ADC full scale

**Returns** *tlv320\_analog\_ch\_t* Analog channel read

```
uint8_t pi_tlv320_set_channel_input(pi_device_t *tlv320_dev, uint8_t channel, tlv320_state_t state)
```

Enable/disable an input channel.

---

**Note:** Enabling a channel is different than powering up channels !

---

**Parameters**

- **tlv320\_dev** – TLV320 instance pointer
- **channel** – Channel input to enable/disable
- **state** – State to apply

**Returns Error** management. The function can return the following error codes:

- *TLV320\_READ\_ERROR* Failed to read register value.
- *TLV320\_WRITE\_ERROR* Failed to write at least one register.
- *TLV320\_NO\_ERROR* No error occurred.

---

`uint8_t pi_tlv320_set_channel_output(pi_device_t *tlv320_dev, uint8_t channel, tlv320_state_t state)`  
Enable/disable an output channel.

---

**Note:** Enabling a channel is different than powering up channels !

---

#### Parameters

- **tlv320\_dev** – TLV320 instance pointer
- **channel** – Channel output to enable/disable
- **state** – State to apply

**Returns Error** management. The function can return the following error codes:

- *TLV320\_READ\_ERROR* Failed to read register value.
- *TLV320\_WRITE\_ERROR* Failed to write at least one register.
- *TLV320\_NO\_ERROR* No error occurred.

`uint8_t pi_tlv320_power(pi_device_t *tlv320_dev, tlv320_state_t state)`

General power control function. Power control for ADC and PDM channels. Power control for the microphone bias output. Power control for the PLL.

---

**Note:** All the programmable coefficient values must be written before powering up the respective channel.

---

**Note:** MicBias setting is ignored if the GPIO1 or GPIx pin is configured to set the microphone bias on or off

- *TLV320\_READ\_ERROR* Failed to read register value.
- *TLV320\_WRITE\_ERROR* Failed to write at least one register.
- *TLV320\_NO\_ERROR* No error occurred.

---

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.
- **state** – [in] Desired state.
  - *TLV320\_DISABLE* Power down
  - *TLV320\_ENABLE* Power up

`void pi_tlv320_set_slave_clk_pll(pi_device_t *tlv320_dev)`

TLV320 clock configuration. Set the device's clock into slave mode based on internal PLL mechanism.

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.

`void pi_tlv320_set_slave_clk_bclk(pi_device_t *tlv320_dev)`

TLV320 clock configuration. Set the device's clock into slave mode based on Bit clock signal.

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.

```
void pi_tlv320_set_slave_clk_mclk(pi_device_t *tlv320_dev, uint8_t ratio)
    TLV320 clock configuration. Set the device's clock into slave mode based on a ratio from FSYNC signal.
```

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.
- **ratio** – [in] Ratio from FSYNC signal.
  - See [TLV320\\_IS\\_VALID\\_MCLK\\_RATIO\\_SEL](#) for accepted values.

```
void pi_tlv320_set_master_clk(pi_device_t *tlv320_dev, tlv320_fs_t fs, tlv320_mclk_t mclk)
    TLV320 clock configuration. Set the device's clock into master mode.
```

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.
- **fs** – [in] ASI bus frame synchronization signal configuration
- **mclk** – [in] Master clock configuration structure

```
void pi_tlv320_set_clk(pi_device_t *tlv320_dev, tlv320_clk_cfg_t clk_cfg)
```

TLV320 clock configuration. This function allows to set all clocks parameters from main clock configuration structure.

#### Parameters

- **tlv320\_dev** – [inout] TLV320 instance pointer.
- **clk\_cfg** – [in] Clock configuration parameters structure.

```
void pi_tlv320_set_shutdown_mode(pi_device_t *tlv320_dev, tlv320_hw_mode_t state)
```

Control TLV320 functional state by setting its SHDNZ pin to LOW (for shutdown state) or HIGH (for active state).

#### Parameters

- **tlv320\_dev** – TLV320 instance pointer
- **state** – Enable or disable hardware shutdown mode
  - [TLV320\\_SHUTDOWN](#) Set the device into shutdown mode
  - [TLV320\\_ACTIVE](#) Set the device into active mode.

```
struct tlv320_asci_t
```

#include <tlv320.h> Audio serial interface configuration structure.

#### Public Members

```
tlv320_asci_format_t format
    ASI protocol format selection.
```

```
tlv320_asci_wlen_t wlen
    ASI word/slot lenght.
```

```
tlv320_asci_fsync_pol_t fsync_pol
    ASI FSYNC polarity.
```

*tlv320\_asi\_bclk\_pol\_t bclk\_pol*  
ASI BCLK polarity.

*tlv320\_asi\_tx\_edge\_t tx\_edge*  
ASI data output transmit edge.

*tlv320\_asi\_tx\_fill\_t tx\_fill*  
ASI data output for any unused cycles.

*tlv320\_tx\_lsb\_t tx\_lsb*  
ASI data output for LSB transmissions.

*tlv320\_tx\_keeper\_t tx\_keeper*  
ASI data output bus keeper.

*uint8\_t tx\_offset*  
ASI data MSB slot 0 offset.

```
struct tlv320_ch_input_t
#include <tlv320.h> Analog channel input parameters structure definition.
```

## Public Members

*tlv320\_ch\_intyp\_t type*  
Channel's input type.

*tlv320\_ch\_insrc\_t src*  
Channel's input configuration.

*tlv320\_ch\_input\_coupling\_t coupling*  
Channel's input coupling.

*tlv320\_ch\_input\_z\_t z*  
Channel's input impedance.

*tlv320\_ch\_input\_agcdre\_t agcdre*  
Channel's Dynamic range enhancer and automatic gain controller setting.

```
struct tlv320_ch_output_t
#include <tlv320.h> Analog channel output parameters structure definition.
```

**Public Members**

**uint8\_t slot**  
Channel Slot assignement.

*tlv320\_ch\_output\_line\_t* **line**  
Channel output selection.

struct **tlv320\_analog\_ch\_t**  
*#include <tlv320.h>* Analog channel structure definition.

**Public Members**

**uint8\_t id**  
Analog channel id.

*tlv320\_ch\_input\_t* **input**  
Input parameters.

*tlv320\_ch\_output\_t* **output**  
Output parameters.

**uint8\_t gain**  
Channel gain.

**float volume**  
Channel digital volume control.

**float gain\_calib**  
Channel gain calibration.

**uint8\_t phase\_calib**  
Channel phase calibration.

struct **tlv320\_clk\_mode\_t**  
*#include <tlv320.h>* TLV320 clk main configuration.

**Public Members**

**uint8\_t master**  
ASI master or slave configuration.

- See *TLV320\_IS\_VALID\_ASI\_MASTER\_SLAVE\_CFG* for possible values

**uint8\_t auto\_clk**  
Automatic clock configuration.

- See *TLV320\_IS\_VALID\_AUTO\_CLK* for possible values

**uint8\_t auto\_pll**  
Automatic mode PLL setting.

- See [TLV320\\_IS\\_VALID\\_PLLAUTOMODE](#) for possible values

`uint8_t clk_src`

Audio root clock source setting.

- See [TLV320\\_IS\\_VALID\\_AUDIOROOT\\_CLK\\_SRC](#) for possible values

`struct tlv320_fs_t`

`#include <tlv320.h>` TLV320 Frame synchronisation (FS) bus configuration.

### Public Members

`uint8_t mode`

Sample rate setting. This parameter is valid when the device is in master mode.

- See [TLV320\\_IS\\_VALID\\_FSMODE](#) for possible values

`uint8_t rate`

Programmed sample rate of the ASI bus. This parameter is not used when the device is configured in slave mode auto clock configuration. See

- [tlv320\\_clk\\_mode\\_t.auto\\_clk](#) for more details.
- See [TLV320\\_IS\\_VALID\\_FSRATE](#) for possible values

`uint8_t gate`

BCLK and FSYNC clock gate. This parameter is valid when the device is in master mode.

- See [TLV320\\_IS\\_VALID\\_BCLKFSYNC\\_GATE](#) for possible values

`uint8_t bclk_ratio`

Programmed BCLK to FSYNC frequency ratio of the ASI bus. This parameter is not used when the device is configured in slave mode auto clock configuration. See

- [tlv320\\_clk\\_mode\\_t.auto\\_clk](#) for more details.
- See [TLV320\\_IS\\_VALID\\_FS\\_BCLK\\_RATIO](#) for possible values

`struct tlv320_mclk_t`

`#include <tlv320.h>` TLV320 Master clock configuration.

### Public Members

`uint8_t mode`

Master mode MCLK frequency selection mode.

Select the way the master clock is defined between from a predefined list represented by [tlv320\\_mclk\\_t::select](#) member or from a multiple of the frame synchronization signal FS represented by fs\_ratio member.

This member is only valid when the device is in auto clock configuration. [tlv320\\_clk\\_mode\\_t::auto\\_pll](#) for more details.

- See [TLV320\\_IS\\_VALID\\_MCLK\\_FREQ\\_MODE](#) for possible values

`uint8_t fs_ratio`

Select the MCLK to FSYNC ratio.

This parameter is valid when the device is in master mode and when `tlv320_mclk_t::mode` is set to `TLV320_CLKSRC_MCLK_FREQ_MODE_FSYNC_RATIO`

This parameter is valid when the device is in slave mode and master clock MCLK is used as the audio clock source in slave mode. See `tlv320_clk_mode_t::clk_src` for more info.

- See `TLV320_IS_VALID_MCLK_RATIO_SEL` for possible values

**uint8\_t select**

Select the MCLK frequency for the PLL source.

This parameter is valid when the device is in master mode and when `tlv320_mclk_t::mode` is set to `TLV320_CLKSRC_MCLK_FREQ_MODE_FREQ_SEL`

- See `TLV320_IS_VALID_MCLK` for possible values

**struct tlv320\_clk\_cfg\_t**

#include <tlv320.h> TLV320 main structure to set ASI clock configuration.

**Public Members****`tlv320_clk_mode_t mode`**

Set TLV320 clock modes.

**`tlv320_fs_t fs`**

Frame synchronization bus (FSYNC) configuration.

**`tlv320_mclk_t mclk`**

Master clock parameters.

**union tlv320\_sw\_reset\_t**

#include <tlv320.h> SW\_RESET (0x01) register definition.

**Public Members****uint8\_t sw\_reset**

Software reset.

**uint8\_t reserved**

Reserved.

**struct `tlv320_sw_reset_t`.[anonymous] [anonymous]****uint8\_t raw****union tlv320\_sleep\_cfg\_t**

#include <tlv320.h> SLEEP\_CFG (0x02) register definition.

---

## Public Members

`uint8_t sleep_enz`  
Sleep mode setting.

`uint8_t reserved_1`  
Reserved.

`uint8_t i2c_brdcast_en`  
I2C broadcast addressing setting.

`uint8_t vref_qchg`  
Quick charge duration.

`uint8_t reserved_5_6`  
Reserved.

`uint8_t areg_select`  
The analog supply selection.

`struct tlv320_sleep_cfg_t.[anonymous] [anonymous]`

`uint8_t raw`  
Register content raw version.

`union tlv320_asi_cfg0_t`  
`#include <tlv320.h>` ASI\_CFG0 (0x07) register definition.

## Public Members

`uint8_t tx_fill`  
ASI data output for any unused cycles.

`uint8_t tx_edge`  
ASI data output transmit edge.

`uint8_t bclk_pol`  
ASI BCLK polarity.

`uint8_t fsync_pol`  
ASI FSYNC polarity.

`uint8_t asi_wlen`  
ASI word or slot length.

`uint8_t asi_format`  
ASI protocol format.

`struct tlv320_asi_cfg0_t.[anonymous] [anonymous]`

```
uint8_t raw
    Register content raw version.

union tlv320_asi_cfg1_t
    #include <tlv320.h> ASI_CFG1 (0x08) register definition.
```

### Public Members

```
uint8_t tx_offset
    ASI data MSB slot 0 offset.

uint8_t tx_keeper
    ASI data output bus keeper.

uint8_t tx_lsb
    ASI data output for LSB transmissions.

struct tlv320_asi_cfg1_t.[anonymous] [anonymous]
    uint8_t raw
        Register content raw version.

union tlv320_asi_ch_t
    #include <tlv320.h> ASI_CHx (0x0B to 0x12) register definition.
```

### Public Members

```
uint8_t slot
    Channel slot assignment.

uint8_t output
    Channel output line.

uint8_t reserved
    Reserved.

struct tlv320_asi_ch_t.[anonymous] [anonymous]
    uint8_t raw
        Register content raw version.

union tlv320_mst_cfg0_t
    #include <tlv320.h> MST_CFG0 (0x13) register definition.
```

## Public Members

`uint8_t mclk_freq_sel`

MCLK (GPIO or GPIx) frequency for the PLL source Selection.

`uint8_t fs_mode`

Sample rate setting (valid when the device is in master mode).

`uint8_t bclk_fsync_gate`

BCLK and FSYNC clock gate (valid when the device is in master mode).

`uint8_t auto_mode_pll_dis`

Automatic mode PLL setting.

`uint8_t auto_clk_cfg`

Automatic clock configuration setting.

`uint8_t mst_slv_cfg`

ASI master or slave configuration register setting.

`struct tlv320_mst_cfg0_t.[anonymous] [anonymous]`

`uint8_t raw`

Register content raw version.

`union tlv320_mst_cfg1_t`

`#include <tlv320.h>` MST\_CFG1 (0x14) register definition.

## Public Members

`uint8_t fs_bclk_ratio`

Programmed sample rate of the ASI bus.

`uint8_t fs_rate`

Programmed BCLK to FSYNC frequency ratio of the ASI bus.

`struct tlv320_mst_cfg1_t.[anonymous] [anonymous]`

`uint8_t raw`

Register content raw version.

`union tlv320_clk_src_t`

`#include <tlv320.h>` CLK\_SRC (0x16) register definition.

### Public Members

`uint8_t reserved`

Reserved.

`uint8_t mclk_ratio_sel`

Select the MCLK (GPIO or GPIx) to FSYNC ratio for master mode or when MCLK is used as the audio root clock source in slave mode.

`uint8_t mclk_freq_sel_mode`

Master mode MCLK (GPIO or GPIx) frequency selection mode (valid when the device is in auto clock configuration)

`uint8_t dis_pll_slv_clk_src`

Audio root clock source setting when the device is configured with the PLL disabled in the auto clock configuration for slave mode.

struct *tlv320\_clk\_src\_t*.[anonymous] **[anonymous]**

`uint8_t raw`

Register content raw version.

union **tlv320\_gpo\_cfg\_t**

#include <tlv320.h> GPO\_CFGx (0x22 to 0x25) register structure definition.

### Public Members

`uint8_t drv`

GPO output drive configuration.

`uint8_t reserved`

Reserved.

`uint8_t cfg`

GPO Configuration.

struct *tlv320\_gpo\_cfg\_t*.[anonymous] **[anonymous]**

`uint8_t raw`

Register content raw version.

union **tlv320\_gpi\_cfg\_t**

#include <tlv320.h> GPI\_CFGx (0x2B & 0x2C) register structure definition.

**Public Members**

**uint8\_t cfg\_y**  
GPI (2 or 4) Configuration.

**uint8\_t reserved1**  
Reserved.

**uint8\_t cfg\_x**  
GPI (1 or 3) Configuration.

**uint8\_t reserved2**  
Reserved.

struct *tlv320\_gpi\_cfg\_t*.[anonymous] **[anonymous]**

**uint8\_t raw**  
Register content raw version.

union **tlv320\_bias\_cfg\_t**  
*#include <tlv320.h>* BIAS\_CFG (0x3B) register definition.

**Public Members**

**uint8\_t adc\_fscale**  
ADC full-scale setting

**uint8\_t reserved1**  
Bits 2 & 3 reserved.

**uint8\_t mbias\_val**  
MICBIAS setting

**uint8\_t reserved2**  
Bit 7 reserved.

struct *tlv320\_bias\_cfg\_t*.[anonymous] **[anonymous]**

**uint8\_t raw**

union **tlv320\_ch\_cfg0\_t**  
*#include <tlv320.h>* CHx\_CFG0 (0x3C, 0x41, 0x46, 0x4A) register definition.

**Public Members**

**uint8\_t ch\_dreen**  
Channel DRE and AGC setting.

**uint8\_t reserved**  
Reserved.

**uint8\_t ch\_imp**  
Channel input impedance.

**uint8\_t ch\_dc**  
Channel input coupling.

**uint8\_t ch\_insrc**  
Channel input configuration.

**uint8\_t ch\_inttyp**  
Channel input type.

struct *tlv320\_ch\_cfg0\_t*.[anonymous] [**anonymous**]

**uint8\_t raw**  
Register content raw version.

union **tlv320\_ch\_cfg1\_t**  
#include <tlv320.h> CHx\_CFG1 (0x3D, 0x42, 0x47, 0x4B) register definition.

**Public Members**

**uint8\_t reserved**  
Reserved.

**uint8\_t ch\_gain**  
Channel gain.

struct *tlv320\_ch\_cfg1\_t*.[anonymous] [**anonymous**]

**uint8\_t raw**  
Register content raw version.

union **tlv320\_ch\_cfg2\_t**  
#include <tlv320.h> CHx\_CFG2 (0x3E, 0x43, 0x48, 0x4C) register definition.

---

## Public Members

```
uint8_t ch_dvol  
Channel gain.
```

```
struct tlv320_ch_cfg2_t.[anonymous] [anonymous]
```

```
uint8_t raw  
Register content raw version.
```

```
union tlv320_ch_cfg3_t
```

```
#include <tlv320.h> CHx_CFG3 (0x3F, 0x44, 0x49, 0x4D) register definition.
```

## Public Members

```
uint8_t reserved  
Reserved.
```

```
uint8_t ch_gcal  
Channel gain calibration.
```

```
struct tlv320_ch_cfg3_t.[anonymous] [anonymous]
```

```
uint8_t raw  
Register content raw version.
```

```
union tlv320_ch_cfg4_t
```

```
#include <tlv320.h> CHx_CFG4 (0x3F, 0x44, 0x49, 0x4D) register definition.
```

## Public Members

```
uint8_t ch_pcal  
Channel gain calibration.
```

```
struct tlv320_ch_cfg4_t.[anonymous] [anonymous]
```

```
uint8_t raw  
Register content raw version.
```

```
union tlv320_in_ch_en_t
```

```
#include <tlv320.h> IN_CH_EN (0x73) register definition.
```

**Public Members**

**uint8\_t in\_ch8\_en**  
Input channel 8 enable setting.

**uint8\_t in\_ch7\_en**  
Input channel 7 enable setting.

**uint8\_t in\_ch6\_en**  
Input channel 6 enable setting.

**uint8\_t in\_ch5\_en**  
Input channel 5 enable setting.

**uint8\_t in\_ch4\_en**  
Input channel 4 enable setting.

**uint8\_t in\_ch3\_en**  
Input channel 3 enable setting.

**uint8\_t in\_ch2\_en**  
Input channel 2 enable setting.

**uint8\_t in\_ch1\_en**  
Input channel 1 enable setting.

struct *tlv320\_in\_ch\_en\_t*.[anonymous] [**anonymous**]

**uint8\_t raw**  
Register content raw version.

union **tlv320\_asi\_out\_ch\_en\_t**  
#include <tlv320.h> ASI\_OUT\_CH\_EN (0x74) register structure definition.

**Public Members**

**uint8\_t asi\_out\_ch8\_en**  
ASI output channel 8 enable setting.

**uint8\_t asi\_out\_ch7\_en**  
ASI output channel 7 enable setting.

**uint8\_t asi\_out\_ch6\_en**  
ASI output channel 6 enable setting.

**uint8\_t asi\_out\_ch5\_en**  
ASI output channel 5 enable setting.

**uint8\_t asi\_out\_ch4\_en**  
ASI output channel 4 enable setting.

```

uint8_t asi_out_ch3_en
ASI output channel 3 enable setting.

uint8_t asi_out_ch2_en
ASI output channel 2 enable setting.

uint8_t asi_out_ch1_en
ASI output channel 1 enable setting.

struct tlv320_asi_out_ch_en_t.[anonymous] [anonymous]

uint8_t raw

union tlv320_pwr_cfg_t
#include <tlv320.h> PWR_CFG (0x75) register structure definition.

```

### Public Members

```

uint8_t reserved
Reserved.

uint8_t dyn_maxch_sel
Dynamic mode maximum channel select configuration.

uint8_t dyn_ch_pupd_en
Dynamic channel power-up, power-down enable.

uint8_t pll_pdz
Power control for the PLL.

uint8_t adc_pdz
Power control for ADC and PDM channels.

uint8_t micbias_pdz
Power control for MICBIAS.

struct tlv320_pwr_cfg_t.[anonymous] [anonymous]

uint8_t raw
Register content raw version.

struct pi_tlv320_shdnz_api_t
#include <tlv320.h> API that control TLV320 SHDNZ pin. TLV320 module abstract this pin control in
order to be used in several ways. Either by direct GPIO control from the GAP or through a GPIO expander
via i2c.

```

**Public Members**

```
int (*set)(void*)
BSP provided. Set the GPIO that power up the TLV320.
```

```
int (*reset)(void*)
BSP provided. Reset the GPIO to power down the TLV320.
```

```
struct bsp_tlv320_shdnz_t
```

```
#include <tlv320.h> SHDNZ abstraction structure. This object handles the SHDNZ pin:
```

- What triggers it (GAP9 GPIO, GPIO expander via i2C, other...)
- Its features (enable/disable the device)

```
struct pi_tlv320_conf
```

```
#include <tlv320.h> TLV320 configuration structure. Contains all hardware related parameters of the TLV320.
```

**Public Members**

```
int i2c_itf
```

```
I2C interface ID.
```

```
uint32_t i2c_max_baudrate
```

```
I2C baudrate.
```

```
uint8_t addr1_miso
```

```
TLV320 ADDR1 pin state.
```

```
uint8_t addr0_sclk
```

```
TLV320 ADDR0 pin state.
```

```
bsp_tlv320_shdnz_t shdnz
```

```
TLV320 shdnz interface.
```

**DAC****AK4332**

*group AK4332*  
DAC AK4332.

This is the API of the AK4332 driver.

## Driver Description

The AK4332 is a 32-bits mono audio DAC that can accept PCM, PDM and DSD data. It consists in 21 registers to program via I2C in order to generate audio signal. These registers are organized in three parts :

1. Configuration registers These are registers that needs to be configured before starting the device. They set the AK4332's way to operate. For example, they set the device clock tree, the type of data that is received... These registers are available from the API.
2. Power management registers These are registers used to power-up and power-down the device. They must be driven in a specific way according to the driver documentation. That is why they are packaged into a simple enable/disable API.
3. Error report registers The driver offer the possibility to detect a fullscale error in PDM mode. This information is available in the only register of this kind : PDMERR. See [\*pi\\_ak4332\\_is\\_pdm\\_fullscale\*](#) for more details.

This API provides a way to drive this device through a few functions.

## How to use this driver

This driver is highly dependant on the hardware. The BSP (Board Support Package) must provide its configuration that describe which I2C and I2S interface to use plus the GPIO instance that will enable the AK4332 and finally the AK4332 Dynamic API. See [\*pi\\_ak4332\\_conf\\_t\*](#) for more details.

This Dynamic API contains functions pointer that represent the main AK4332 features.

- init
- start
- stop
- set volume
- enable
- disable

This AK4332 driver provides the standard implementation of these function but the BSP has to provide adapted functions pointed by this API because they are hardware dependant. (Meanwhile, only the standard implementation used).

As an example, the board Audio Addon V1.1 can drive one or two AK4332. Depending on that point, the implementation of these features is different. Again, the enable/disable features can be handled by a GAP pad or another device controlled by GPIO such as the FXL6408 on the audio addon V1.1. See [\*pi\\_ak4332\\_api\\_t\*](#) for more details.

The I2C part is fully handled by the driver in the [\*pi\\_ak4332\\_open\*](#) function. Meanwhile, the I2S part needs to be initialized before opening the AK4332 device. Here is an example how to initialize I2S peripheral to generate PDM data:

### *AK4332 PDM example*

Here is an example how to initialize I2S peripheral to generate PCM data:

### AK4332 PCM example

After being opened, this driver provides PDM and PCM presets that can help you to configure the device. These presets fill a structure object type named `ak4332_init_struct_t` which is an organized description of all AK4332 configuration registers. It is organized in 4 sub structures that represents the main part of the device:

- Clock
- Serial audio interface
- DAC
- HeadPhone

This presets are described by the enum `ak4332_preset_e` and can be loaded with `pi_ak4332_load_init_preset`.

To write these settings into the device, it must be powered up first with the “enable” function. Then, the initialization structure needs to be written with the “init” function into the AK4332 registers. Finally, use the “start” function to reach the AK4332’s normal operation mode.

I2S flow can be enable at this point.

The volume can be adjusted by the “set\_volume” function.

The audio stream generated by the AK4332 can be stopped by using the “stop” function. it is similar to a pause feature (It can be relaunch with the “start” function). To fully stop the driver, use the disable function. WARNING, this function reset all registers.

### Typedefs

`typedef struct pi_ak4332_api pi_ak4332_api_t`

AK4332 dynamic API. This structure offers slots to control the AK4332. The AK4332 should be provided a configuration with appropriate implementations before being opened. The driver can handle empty start, stop and init function by calling its std API (See but not enable and disable features since it is fully dependant on the BSP).

`typedef struct pi_ak4332_conf pi_ak4332_conf_t`

AK4332 hardware configuration structure. Contains all hardware related parameters of the AK4332. The BSP must define at least one static AK4332 configuration then, it must be loaded with `pi_open_from_conf` function before opening the AK4332 device.

### Enums

`enum ak4332_preset_e`

List of preset integrated in this AK4332 driver.

*Values:*

`enumerator AK4332_PCM_SLAVE_PLL_BCLK_32WS_48KHZ`

PCM Data, Clock used : BitClock (slave), 32 bits, Sampling frequency : 48kHz.

`enumerator AK4332_PDM_DSD_SLAVE_PLL_BCLK_32WS_48KHZ`

DSD Data, Clock used : BitClock (slave), 32 bits, Sampling frequency : 48kHz.

enumerator **AK4332\_PRESETS\_NUMBER**

Number of known presets.

enum **ak4332\_cfg\_register\_t**

Enumeration of all AK4332 registers addresses.

*Values:*

enumerator **AK4332\_REG\_PM1** = 0x00

Power Management 1.

enumerator **AK4332\_REG\_PM2** = 0x01

Power Management 2.

enumerator **AK4332\_REG\_PM3** = 0x02

Power Management 3.

enumerator **AK4332\_REG\_PM4** = 0x03

Power Management 4.

enumerator **AK4332\_REG\_OMS** = 0x04

Output Mode Setting.

enumerator **AK4332\_REG\_CMS** = 0x05

Clock Mode Selection.

enumerator **AK4332\_REG\_DFS** = 0x06

Digital Filter Selection.

enumerator **AK4332\_REG\_DMM** = 0x07

DAC Mono Mixing.

enumerator **AK4332\_REG\_PIC** = 0x08

PDM Interface Control.

enumerator **AK4332\_REG\_DOV** = 0x0B

DAC Output Volume.

enumerator **AK4332\_REG\_HPVC** = 0x0D

Headphone Volume Control.

enumerator **AK4332\_REG\_PCSS** = 0x0E

PLL Clock Source Selection.

enumerator **AK4332\_REG\_PRCD1** = 0x0F

PLL Reference Clock Divider 1.

enumerator **AK4332\_REG\_PRCD2** = 0x10

PLL Reference Clock Divider 2.

enumerator **AK4332\_REG\_PFCD1** = 0x11  
PLL Feedback Clock Divider 1.

enumerator **AK4332\_REG\_PFCD2** = 0x12  
PLL Feedback Clock Divider 2.

enumerator **AK4332\_REG\_DCS** = 0x13  
DAC CLK Source.

enumerator **AK4332\_REG\_DCD** = 0x14  
DAC CLK Divider.

enumerator **AK4332\_REG\_AIF** = 0x15  
Audio Interface Format.

enumerator **AK4332\_REG\_PDMERR** = 0x17  
PDMERR.

enumerator **AK4332\_REG\_DA1** = 0x26  
DAC Adjustment 1.

enumerator **AK4332\_REG\_DA2** = 0x27  
DAC Adjustment 2.

enum **ak4332\_pm\_t**

Generic type definition for power management control.

*Values:*

enumerator **AK4332\_PM\_DOWN** = AK4332\_POWER\_DOWN  
Generic value to Power down an hardware element.

enumerator **AK4332\_PM\_UP** = AK4332\_POWER\_UP  
Generic value to Power up an hardware element.

enum **ak4332\_en\_t**

Generic type definition for enabling/disabling features.

*Values:*

enumerator **AK4332\_DISABLE** = AK4332\_DISABLE\_VALUE  
Generic value to disable.

enumerator **AK4332\_ENABLE** = AK4332\_ENABLE\_VALUE  
Generic value to enable.

enum **ak4332\_bit\_t**

Generic type definition for register bit tuning.

*Values:*

enumerator **AK4332\_RESET**  
Generic value to reset a bit in a register.

enumerator **AK4332\_SET**

Generic value to set a bit in a register.

enum **ak4332\_pll\_en\_t**

Control the status of the AK4332 PLL.

*Values:*

enumerator **PLL\_DISABLE** = AK4332\_POWER\_DOWN

PLL Power Management Power-Down.

enumerator **PLL\_ENABLE** = AK4332\_POWER\_UP

PLL Power Management Power-Up.

enum **ak4332\_lvdtm\_t**

Detection time setting for ClassG 1/2VDD mode.

*Values:*

enumerator **LVDTM\_64** = ((uint8\_t)0x0)

64/fs detection time

enumerator **LVDTM\_128** = ((uint8\_t)0x1)

128/fs detection time

enumerator **LVDTM\_256** = ((uint8\_t)0x2)

256/fs detection time

enumerator **LVDTM\_512** = ((uint8\_t)0x3)

512/fs detection time

enumerator **LVDTM\_1024** = ((uint8\_t)0x4)

1024/fs detection time

enumerator **LVDTM\_2048** = ((uint8\_t)0x5)

2048/fs detection time

enumerator **LVDTM\_4096** = ((uint8\_t)0x6)

4096/fs detection time

enumerator **LVDTM\_8192** = ((uint8\_t)0x7)

8192/fs detection time

enum **ak4332\_cpmode\_t**

Charge pump Mode setting.

*Values:*

enumerator **CPMODE\_CLASSG** = ((uint8\_t)0x0)

Operation voltage : Automatic switching.

enumerator **CPMODE\_VDD** = ((uint8\_t)0x1)

Operation voltage : +/- VDD..

enumerator **Cplode\_Half\_Vdd** = ((uint8\_t)0x2)  
Operation voltage : +/- 1/2 VDD.

enum **ak4332\_lvdsel\_t**

Switching Threshold between VDD Mode and 1/2VDD Mode of Charge pump 2.

*Values:*

enumerator **Lvdsel\_16** = ((uint8\_t)0x0)  
Class-G Switching Level 1.05mW(PCM) / 1.61(PDM) @ 16 Ohms.

enumerator **Lvdsel\_32** = ((uint8\_t)0x1)  
Class-G Switching Level 1.05mW(PCM) / 1.41(PDM) @ 32 Ohms.

enumerator **Lvdsel\_11** = ((uint8\_t)0x2)  
Class-G Switching Level 1.05mW(PCM) / 1.65(PDM) @ 11 Ohms.

enumerator **Lvdsel\_8** = ((uint8\_t)0x3)  
Class-G Switching Level 1.05mW(PCM) / 1.86(PDM) @ 8 Ohms.

enum **ak4332\_vddtm\_t**

Class-G VDD Hold Time Setting.

*Values:*

enumerator **Vddtm\_1024** = ((uint8\_t)0x0)  
VDD Mode Holding period = 1024/fs.

enumerator **Vddtm\_2048** = ((uint8\_t)0x1)  
VDD Mode Holding period = 2048/fs.

enumerator **Vddtm\_4096** = ((uint8\_t)0x2)  
VDD Mode Holding period = 4096/fs.

enumerator **Vddtm\_8192** = ((uint8\_t)0x3)  
VDD Mode Holding period = 8192/fs.

enumerator **Vddtm\_16384** = ((uint8\_t)0x4)  
VDD Mode Holding period = 16384/fs.

enumerator **Vddtm\_32768** = ((uint8\_t)0x5)  
VDD Mode Holding period = 32768/fs.

enumerator **Vddtm\_65536** = ((uint8\_t)0x6)  
VDD Mode Holding period = 65535/fs.

enumerator **Vddtm\_131072** = ((uint8\_t)0x7)  
VDD Mode Holding period = 131072/fs.

enumerator **Vddtm\_262144** = ((uint8\_t)0x8)  
VDD Mode Holding period = 262144/fs.

---

enum **ak4332\_hpohz\_t**  
GND Switch Setting for Headphone Amplifier Output.

*Values:*

enumerator **HPOHZ\_4** = ((uint8\_t)0x0)  
Headphone amplifier output pull down by 4 ohm.

enumerator **HPOHZ\_95** = ((uint8\_t)0x1)  
Headphone amplifier output pull down by 95 kohm.

enum **ak4332\_fs\_t**

*Values:*

enumerator **FS\_8\_KHZ** = ((uint8\_t)0x00)  
Sampling frequency set to 8kHz.

enumerator **FS\_11\_025\_KHZ** = ((uint8\_t)0x01)  
Sampling frequency set to 11,025kHz.

enumerator **FS\_12\_KHZ** = ((uint8\_t)0x02)  
Sampling frequency set to 12kHz.

enumerator **FS\_16\_KHZ** = ((uint8\_t)0x04)  
Sampling frequency set to 16kHz.

enumerator **FS\_22\_05\_KHZ** = ((uint8\_t)0x05)  
Sampling frequency set to 22,05kHz.

enumerator **FS\_24\_KHZ** = ((uint8\_t)0x06)  
Sampling frequency set to 24kHz.

enumerator **FS\_32\_KHZ** = ((uint8\_t)0x08)  
Sampling frequency set to 32kHz.

enumerator **FS\_44\_1\_KHZ** = ((uint8\_t)0x09)  
Sampling frequency set to 44,1kHz.

enumerator **FS\_48\_KHZ** = ((uint8\_t)0x0A)  
Sampling frequency set to 48kHz.

enumerator **FS\_64\_KHZ** = ((uint8\_t)0x0C)  
Sampling frequency set to 64kHz.

enumerator **FS\_88\_2\_KHZ** = ((uint8\_t)0x0D)  
Sampling frequency set to 88,2kHz.

enumerator **FS\_96\_KHZ** = ((uint8\_t)0x0E)  
Sampling frequency set to 96kHz.

enumerator **FS\_128\_KHZ** = ((uint8\_t)0x10)  
Sampling frequency set to 128kHz.

enumerator **FS\_176\_4\_KHZ** = ((uint8\_t)0x11)  
Sampling frequency set to 176,4kHz.

enumerator **FS\_192\_KHZ** = ((uint8\_t)0x12)  
Sampling frequency set to 192kHz.

enum **ak4332\_cm\_t**  
Master Clock frequency setting.

*Values:*

enumerator **CM\_256\_FS** = ((uint8\_t)0x0)  
Master clock frequency = 256 times Fs / Sampling frequency range = 8 to 96kHz.

enumerator **CM\_512\_FS** = ((uint8\_t)0x1)  
Master clock frequency = 512 times Fs / Sampling frequency range = 8 to 48kHz.

enumerator **CM\_1024\_FS** = ((uint8\_t)0x2)  
Master clock frequency = 1024 times Fs / Sampling frequency range = 8 to 24kHz.

enumerator **CM\_128\_FS** = ((uint8\_t)0x3)  
Master clock frequency = 128 times Fs / Sampling frequency range = 128 to 192kHz.

enum **ak4332\_filter\_t**  
DAC Filter setting.

*Values:*

enumerator **SHARP\_ROLL\_OFF** = ((uint8\_t)0x0)  
Sharp Roll-Off Filter.

enumerator **SLOW\_ROLL\_OFF** = ((uint8\_t)0x1)  
Slow Roll-Off Filter.

enumerator **SHORT\_DELAY\_SHARP\_ROLL\_OFF** = ((uint8\_t)0x2)  
Short Delay Sharp Roll-Off Filter.

enumerator **SHORT\_DELAY\_SLOW\_ROLL\_OFF** = ((uint8\_t)0x3)  
Short Delay Slow Roll-Off Filter.

enum **ak4332\_dac\_input\_t**  
DAC Input Data selection.

*Values:*

enumerator **DAC\_INPUT\_MUTE** = ((uint8\_t)0x0)  
Nothing sent to SDTI pin in PCM Mode.

enumerator **DAC\_INPUT\_LEFT** = ((uint8\_t)0x1)  
Send left channel to SDTI pin in PCM mode.

---

enumerator **DAC\_INPUT\_RIGHT** = ((uint8\_t)0x2)  
Send right channel to SDTI pin in PCM Mode.

enumerator **DAC\_INPUT\_LEFT\_AND\_RIGHT** = ((uint8\_t)0x3)  
Send left plus right channel to SDTI pin in PCM Mode.

enumerator **DAC\_INPUT\_HALF\_LEFT** = ((uint8\_t)0x5)  
Send left channel divided by 2 to SDTI pin in PCM Mode.

enumerator **DAC\_INPUT\_HALF\_RIGHT** = ((uint8\_t)0x6)  
Send right channel divided by 2 to SDTI pin in PCM Mode.

enumerator **DAC\_INPUT\_AVERAGE\_LEFT\_RIGHT** = ((uint8\_t)0x7)  
Send Left and Right channel average value to SDTI pin in PCM Mode.

enum **ak4332\_dac\_polarity\_t**  
DAC Input Data polarity.

*Values:*

enumerator **DAC\_POLARITY\_NORMAL** = ((uint8\_t)0x0)  
DAC input Signal Polarity is normal.

enumerator **DAC\_POLARITY\_INVERTED** = ((uint8\_t)0x1)  
DAC input Signal Polarity is inverted.

enum **ak4332\_datatype\_t**  
DAC Input datatype.

*Values:*

enumerator **AK4332\_PCM** = 0  
Pulse coded modulation mode.

enumerator **AK4332\_PDM** = 1  
Pulse-density modulation mode.

enumerator **AK4332\_DSD** = 2  
Direct Stream digital mode.

enum **ak4332\_dsdclk\_pol\_t**  
DSD Clock polarity setting.

*Values:*

enumerator **AK4332\_DSDCLK\_F\_EDGE** = ((uint8\_t)0x0)  
DSD Data is output on a DSDCLK falling edge.

enumerator **AK4332\_DSDCLK\_R\_EDGE** = ((uint8\_t)0x1)  
DSD Data is output on a DSDCLK rising edge.

enum **ak4332\_pdmclock\_pol\_t**  
PDM Clock polarity.

*Values:*

enumerator **AK4332\_PDMCLK\_R\_EDGE** = ((uint8\_t)0x0)  
PDM Data is output on a PDMCLK falling edge.

enumerator **AK4332\_PDMCLK\_F\_EDGE** = ((uint8\_t)0x1)  
PDM Data is output on a PDMCLK rising edge.

enum **ak4332\_pdm\_mute\_t**  
PDM Mute feature activation.

*Values:*

enumerator **AK4332\_PDM\_DATA\_MUTE\_ENABLE** = ((uint8\_t)0x0)  
Enable PDM data / DSD data output mute function.

enumerator **AK4332\_PDM\_DATA\_MUTE\_DISABLE** = ((uint8\_t)0x1)  
Disable PDM data / DSD data output mute function.

enum **ak4332\_hptm\_t**  
Zero Cross Time Output Period Setting for Analog Volume of Headphone Amplifier.

*Values:*

enumerator **HPTM\_128\_FS** = ((uint8\_t)0x0)  
Headphone volume zero cross timeout set to 128/Fs.,

enumerator **HPTM\_256\_FS** = ((uint8\_t)0x1)  
Headphone volume zero cross timeout set to 256/Fs.,

enumerator **HPTM\_512\_FS** = ((uint8\_t)0x2)  
Headphone volume zero cross timeout set to 512/Fs.,

enumerator **HPTM\_1024\_FS** = ((uint8\_t)0x3)  
Headphone volume zero cross timeout set to 1024/Fs.,

enumerator **HPTM\_2048\_FS** = ((uint8\_t)0x4)  
Headphone volume zero cross timeout set to 2048/Fs.

enum **ak4332\_pll\_clksrc\_t**  
PLL Clock source selection.

*Values:*

enumerator **PLL\_CLKSRC\_MCKI** = ((uint8\_t)0x0)  
MCKI pin set as PLL CLock Source.,

enumerator **PLL\_CLKSRC\_BCLK** = ((uint8\_t)0x1)  
BCLK pin set as PLL CLock Source.

enum **ak4332\_pll\_mode\_t**  
PLL Mode Setting.

*Values:*

---

enumerator **PLLMD\_HIGH\_F\_MODE** = ((uint8\_t)0x0)  
Must be set to 0 when the REFCLK is higher than 256kHz.

enumerator **PLLMD\_LOW\_F\_MODE** = ((uint8\_t)0x1)  
Must be set to 1 when the REFCLK is higher than 256kHz.

enum **ak4332\_dac\_clksrc\_t**  
DAC Clock source setting.

*Values:*

enumerator **DAC\_CLKSRC\_MCKI** = ((uint8\_t)0x0)  
MCKI set as DAC clock source.

enumerator **DAC\_CLKSRC\_PLLCLK** = ((uint8\_t)0x1)  
PLL CLK set as DAC clock source.

enum **ak4332\_wordsize\_t**  
DAC input data size.

*Values:*

enumerator **DATALENGTH\_24** = ((uint8\_t)0x0)  
24 bit linear / Slave mode: >= 48 fs / Master Mode N/A

enumerator **DATALENGTH\_16** = ((uint8\_t)0x1)  
16 bit linear / Slave mode: >= 32 fs / Master Mode: 32fs (BCKO bit = 1)

enumerator **DATALENGTH\_32** = ((uint8\_t)0x2)  
32 bit linear / Slave mode: >= 64 fs / Master Mode: 64fs (BCKO bit = 0)

enum **ak4332\_sai\_format\_t**  
Input data format.

*Values:*

enumerator **FORMAT\_I2S** = ((uint8\_t)0x0)  
Interface format set to I2S compatible.

enumerator **FORMAT\_MSB\_JUSTIFIED** = ((uint8\_t)0x1)  
Interface format set to MSB justified.

enum **ak4332\_sai\_mode\_t**  
Clock provider setting.

*Values:*

enumerator **AK4332\_SLAVE** = ((uint8\_t)0x0)  
Slave mode.,

enumerator **AK4332\_MASTER** = ((uint8\_t)0x1)  
Master mode.

enum **ak4332\_sai\_bclk\_out\_t**

Values:

enumerator **AK4332\_BCLK\_64FS** = ((uint8\_t)0x0)

Bitclock set to 64 Fs.

enumerator **AK4332\_BCLK\_32FS** = ((uint8\_t)0x1)

Bitclock set to 32 Fs.

enum **ak4332\_pdm\_fs\_t**

Values:

enumerator **AK4332\_PDM\_NO\_FULL\_SCALE** = ((uint8\_t)0x0)

enumerator **AK4332\_PDM\_FULL\_SCALE** = ((uint8\_t)0x1)

## Functions

*ak4332\_init\_struct\_t* **pi\_ak4332\_load\_init\_preset**(*ak4332\_preset\_e* preset)

Return a filled initialization structure corresponding to a preset.

### Parameters

- **preset** – Enum value that represent a preset.

**Returns** *ak4332\_init\_struct\_t* Filled initialization structure.

void **pi\_ak4332\_init\_std**(*pi\_device\_t* \*ak4332\_dev, *ak4332\_init\_struct\_t* init)

Standard implementation of the initialization process. Initialize the AK4323 by converting a initialization structure to a Register structure (driver internal mechanism) then write it into AK4332 registers. The device must be enabled before writting this implementation.

**Warning:** This function is used by the AK4323 dynamic API depending on BSP constraints. To properly initialize the device, please use the *pi\_ak4332\_init* function. See the driver description at the top of this documentation to set it correctly.

### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **init** – AK4332 Initialization structure to convert.

void **pi\_ak4332\_start\_std**(*pi\_device\_t* \*ak4332\_dev)

This function is the standard implementation of the starting process. It initiates the AK4332 internal powerup sequence. Before using this function, make sure the device has been enabled and initialized to the right configuration.

**Warning:** This function is used by the AK4323 dynamic API depending on BSP constraints. To properly start the device, please use the *pi\_ak4332\_start* function. See the driver description at the top of this documentation to set it correctly.

### Parameters

- **ak4332\_dev** – AK4332 device instance.

```
void pi_ak4332_stop_std(pi_device_t *ak4332_dev)
```

This function is the standard implementation of the stop process. Stop the AK4332 device from normal operating mode. This function execute the AK4332 internal power down sequence.

**Warning:** This function is used by the AK4323 dynamic API depending on BSP constraints. To properly stop the device, please use the pi\_ak4332\_stop function. See the driver description at the top of this documentation to set it correctly.

#### Parameters

- **ak4332\_dev** – AK4332 device instance.

```
void pi_ak4332_set_volume_std(pi_device_t *ak4332_dev, uint8_t volume)
```

Modify AK4332's global volume. AK4332's volume setting represents the dynamic offered by digital and analog gains :

- Digital gain : OVC bits in DOV register
- Analog gain : HPG bits in HPVC register This function can be used while the AK4332 is running or not. It must not be used during power up/down sequences. This function is the standard implementation of the starting process. It is used by the AK4323 dynamic API depending on BSP constraints.

---

**Note:** The volume given to this function is expressed in percentage. It affect both DOV and HPVC registers.

---



---

**Note:** In order to preserve the SNR, this implementation does not set the digital gain over 0dB.

---

**Note:** The following value shows how the volume affect gains:

- 0% : The AK4332 is muted
  - >0% : Starts from -22 dB (OVC minimum setting & HPG minimum setting)
  - 100% : +4dB (HPG maximum setting, OVC is set to 0dB)
- 

**Warning:** This function is the standard implementation of the set\_volume process. This function is used by the AK4323 dynamic API depending on BSP constraints. To properly stop the device, please use the pi\_ak4332\_set\_volume function. See the driver description at the top of this documentation to set it correctly.

#### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **volume** – Volume to set expressed in percentage.

```
uint8_t pi_ak4332_is_pdm_fullscale(pi_device_t *ak4332_dev)
```

Full scale detection bit read accessor. This feature is only available in PDM/DSD mode.

---

**Note:** When the AK4332 detects full scale signal while PDMMUTEEN bit = “0”, the analog output is muted. (Popnoise may occur on a switching timing to the mute state.) When setting PDMMUTEEN bit = “1”, full scale detection function is available but the analog output will not be muted.

---

**Note:** Todo: If necessary, this function can be included in the dynamic API.

---

#### Parameters

- **ak4332\_dev** – AK4332 device instance.

#### Returns uint8\_t Full scale detection status

- 0: Full scale not detected
- 1: Full scale detected

```
static inline void pi_ak4332_init(pi_device_t *ak4332_dev, ak4332_init_struct_t init)
```

AK4332 Init function. This function call for the AK4332 API’s init function. If it has not been set from an AK4332 configuration, it calls for the standard implementation *pi\_ak4332\_init\_std*. Otherwise the implementation is provided by the BSP.

---

**Note:** Please, refer to *pi\_ak4332\_init\_std* for more information about the standard initialization process.

---

#### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **init** – AK4332 initialization structure.

```
static inline void pi_ak4332_start(pi_device_t *ak4332_dev)
```

AK4332 Start function. This function call for the AK4332 API’s start function. If it has not been set from an AK4332 configuration, it calls for the standard implementation *pi\_ak4332\_start\_std*. Otherwise the implementation is provided by the BSP.

---

**Note:** Please, refer to *pi\_ak4332\_start\_std* for more information about the standard starting process.

---

#### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **init** – AK4332 initialization structure.

```
static inline void pi_ak4332_stop(pi_device_t *ak4332_dev)
```

AK4332 stop function. This function call for the AK4332 API’s stop function. If it has not been set from an AK4332 configuration, it calls for the standard implementation *pi\_ak4332\_stop\_std*. Otherwise the implementation is provided by the BSP.

---

**Note:** Please, refer to [pi\\_ak4332\\_stop\\_std](#) for more information about the standard stop process.

---

### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **init** – AK4332 initialization structure.

static inline void **pi\_ak4332\_set\_volume**(pi\_device\_t \*ak4332\_dev, uint8\_t volume)

AK4332 set volume function. This function update both DAC and HeadPhone gain. If it has not been set from an AK4332 configuration, it calls for the standard implementation [pi\\_ak4332\\_set\\_volume\\_std](#). Otherwise the implementation is provided by the BSP.

---

**Note:** See [pi\\_ak4332\\_set\\_volume\\_std](#) for more informations

---

### Parameters

- **ak4332\_dev** – AK4332 device instance.
- **volume** – Volume to apply. expressed as a percentage.

static inline int **pi\_ak4332\_enable**(pi\_device\_t \*ak4332\_dev)

Enable the device by powering up its power supply. This function calls for the API's enable function.

**Warning:** The BSP must provide an implementation to power up the device. See [pi\\_ak4332\\_api\\_t::enable](#)

### Parameters

- **ak4332\_dev** – AK4332 device instance.

**Returns** int Error management.

- 0 : No issue
- -1: The instance that handle this feature returned an error

static inline int **pi\_ak4332\_disable**(pi\_device\_t \*ak4332\_dev)

Disable the device by powering down its power supply. This function calls for the API's disable function.

**Warning:** The BSP must provide an implementation to power down the device. See [pi\\_ak4332\\_api\\_t::disable](#)

### Parameters

- **ak4332\_dev** – AK4332 device instance.

**Returns** int Error management.

- 0 : No issue
- -1: The instance that handle this feature returned an error

```
int pi_ak4332_open(pi_device_t *device)
    Open a ak4332 device.
```

This function must be called before the ak4332 device can be used. It will do all the needed configuration to make it usable and initialize the handle used to refer to this opened device when calling other functions.

#### Parameters

- **device** – A pointer to the device structure of the device to open. This structure is allocated by the called and must be kept alive until the device is closed.

**Returns** 0 if the operation is successfull, -1 if there was an error.

```
void pi_ak4332_close(struct pi_device *device)
    Close an opened ak4332 device.
```

This function can be called to close an opened ak4332 device once it is not needed anymore, in order to free all allocated resources. Once this function is called, the device is not accessible anymore and must be opened again before being used.

#### Parameters

- **device** – The device structure of the device to close.

```
struct ak4332_pll_t
```

#include <ak4332.h> Sub structure of the clock configuration structure (*ak4332\_clk\_t*) that gather options related to the PLL.

#### Public Members

```
ak4332_pll_en_t en
    PLL Status.
```

```
ak4332_pll_clksrc_t src
    PLL Clock source.
```

```
uint16_t pld
    PLL Reference clock divider.
```

```
uint16_t plm
    PLL Feedback clock divider.
```

```
uint8_t mdiv
    PLLCLK Divider.
```

```
ak4332_pll_mode_t mode
    PLL Mode.
```

```
struct ak4332_clk_t
```

#include <ak4332.h> Sub structure of the main initialization structure (*ak4332\_init\_struct\_t*) dedicated to the device's clock configuration.

## Public Members

*ak4332\_pll\_t* **pll**

PLL Configuration structure.

*ak4332\_dac\_clksrc\_t* **dac\_clksrc**

DAC Clock source.

*ak4332\_fs\_t* **fs**

Sampling frequency.

*ak4332\_cm\_t* **cm**

Master clock configuration.

struct **ak4332\_pdm\_options\_t**

#include <ak4332.h> Sub structure of the Serial Audio interface structure (*ak4332\_sai\_t*) that gather options related to PDM.

## Public Members

*ak4332\_pdm\_mute\_t* **data\_mute**

PDM Data / DSD Data Output Mute Function.

*ak4332\_dsdclk\_pol\_t* **dsd\_clk\_pol**

Polarity of DSDCLK

*ak4332\_pdmclock\_pol\_t* **pdm\_clk\_pol**

Polarity of PDMCLK.

*ak4332\_pdm\_fs\_t* **pdm\_fullscale**

PDM full scale setting.

struct **ak4332\_sai\_t**

#include <ak4332.h> Sub structure of the main initialization structure (*ak4332\_init\_struct\_t*) dedicated to the Serial Audio interface.

## Public Members

*ak4332\_datatype\_t* **data\_type**

Select I2S or MSB justified.

*ak4332\_sai\_mode\_t* **mode**

Master or Slave mode.

*ak4332\_sai\_format\_t* **format**

I2S or MSB justified.

*ak4332\_wordsize\_t* **word\_size**

Wordsize 16, 24 or 32 bits.

***ak4332\_sai\_bclk\_out\_t bclk\_master***

Set the bitclock frequency depending on Fs and Datalength.

***ak4332\_pdm\_options\_t pdm\_options***

PDM configuration structure.

**struct *ak4332\_dac\_t***

#include <ak4332.h> Sub structure of the main initialization structure (*ak4332\_init\_struct\_t*) dedicated to the DAC.

**Public Members*****ak4332\_dac\_input\_t input***

Input signal selection.

***ak4332\_dac\_polarity\_t polarity***

Signal polarity.

***ak4332\_filter\_t filter***

Digital filter mode.

**float *volume***

DAC output volume

**struct *ak4332\_hp\_amp\_t***

#include <ak4332.h> Sub structure of the main initialization structure (*ak4332\_init\_struct\_t*) dedicated to the Headphone amplifier.

**Public Members*****ak4332\_cpmode\_t cpmode***

Charge pump mode.

***ak4332\_lvdsel\_t lvdsel***

Class-G Switching Level Setting.

***ak4332\_vddtm\_t vddtm***

VDD Hold time.

***ak4332\_lvdtm\_t lvdtm***

1/2VDD Mode detection time

**float *volume***

Headphone amplifier analog volume control.

***ak4332\_hptm\_t hptm***

Zero Cross Time Output Period.

*ak4332\_hpohz\_t* **hpohz**  
GND Pull down resistor value.

```
struct ak4332_init_struct_t
    #include <ak4332.h> Driver's main initialization structure. This structure represents the main control panel of the AK4332 driver. It has to be set after the driver being opened and used in the initialization process to be programmed. Please see pi_ak4332_init_std for more details.
```

### Public Members

*ak4332\_clk\_t* **clk**  
Clock configuration structure.

*ak4332\_sai\_t* **sai**  
Serial Audio interface configuration structure.

*ak4332\_dac\_t* **dac**  
DAC configuration structure.

*ak4332\_hp\_amp\_t* **hp**  
Headphone amplifier configuration structure.

```
struct bsp_ak4332_pdn_t
    #include <ak4332.h> PDN pin abstraction structure. This object describes the PDN pin (id) and what controls it (context). It can be the GAP itslef or an intermediate component such as a GPIO extender.
```

### Public Members

**int id**  
GPIO id.

**void \*context**  
Device that controls the GPIO.

```
struct pi_ak4332_api
    #include <ak4332.h> AK4332 dynamic API. This structure offers slots to control the AK4332. The AK4332 should be provided a configuration with appropriate implementations before being opened. The driver can handle empty start, stop and init function by calling its std API (See but not enable and disable features since it is fully dependant on the BSP).
```

**Public Members**

void (\***init**)(pi\_device\_t\*, *ak4332\_init\_struct\_t*)  
Init process, see *pi\_ak4332\_init\_std* for more details.

void (\***start**)(pi\_device\_t\*)  
Start process, see *pi\_ak4332\_start\_std* for more details.

void (\***stop**)(pi\_device\_t\*)  
Stop process, see *pi\_ak4332\_stop\_std* for more details.

void (\***set\_volume**)(pi\_device\_t\*, uint8\_t)  
Set Headphone gain, see *pi\_ak4332\_set\_volume\_std* for more details.

int (\***enable**)(void\*)  
AK4332 enable. Set the PDN pin to high logic state.

int (\***disable**)(void\*)  
AK4332 disable. Set the PDN pin to low logic state.

struct **pi\_ak4332\_conf**

#include <ak4332.h> AK4332 hardware configuration structure. Contains all hardware related parameters of the AK4332. The BSP must define at least one static AK4332 configuration then, it must be loaded with pi\_open\_from\_conf function before opening the AK4332 device.

**Public Members**

int **i2c\_itf**  
I2C interface id.

int **i2s\_itf**  
I2S interface id.

*bsp\_ak4332\_pdn\_t* **pdn**  
PDN pin that enable/disable the AK4332.

*pi\_ak4332\_api\_t* \***api**  
AK4332 Dynamic API.

**WIFI****Common API***group* **WIFI**

The WIFI driver provides support for wireless data transfer between the host and an external peripheral such as a smartphone, using WiFi connection.

## TypeDefs

```
typedef struct pi_wifi_api_s pi_wifi_api_t
WIFI specific API.

Structure holding WIFI specific API.
```

## Functions

`int32_t pi_wifi_open(struct pi_device *device)`  
Open a WIFI device.

This function opens and initializes a WIFI device. This function must be called before using device.

### Parameters

- **device** – Pointer to the WIFI device structure.

**Returns 0** If the operation is successful.

**Returns ERRNO** Error code otherwise.

**Returns** `int32_t` Operation status.

`int32_t pi_wifi_close(struct pi_device *device)`  
Close a WIFI device.

This function closes an opened WIFI device. It frees all allocated objects and interfaces used.

### Parameters

- **device** – Pointer to the WIFI device structure.

**Returns 0** If the operation is successful.

**Returns ERRNO** Error code otherwise.

**Returns** `int32_t` Operation status.

`int32_t pi_wifi_ioctl(struct pi_device *device, uint32_t cmd, void *arg)`  
IOctl commands.

This function is used to send special command to WIFI device.

### Parameters

- **device** – Pointer to the WIFI device structure.
- **cmd** – Ioctl command.
- **arg** – Ioctl command arg.

**Returns** Value Value depends on ioctl command.

`int32_t pi_wifi_at_cmd(struct pi_device *device, char *cmd, char *resp)`  
Send AT command to WIFI device.

This function is used to send AT commands to WIFI device.

---

**Note:** The command string should be plain command, i.e. without the “AT” part.

---

**Note:** The command may or may not return a response.

---

#### Parameters

- **device** – Pointer to the WIFI device structure.
- **cmd** – Command string to send.
- **resp** – Buffer to store response.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

```
int8_t pi_wifi_connect_to_ap(struct pi_device *device, const char *ssid, const char *pwd, pi_evt_t *callback)
```

Connect to an access point.

This function should be used to connect to an access point.

#### Parameters

- **device** – Pointer to the WIFI device structure.
- **ssid** – SSID in string format.
- **pwd** – Password in string format.
- **callback** – Task to notify when the module is connected to the AP.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

```
int8_t pi_wifi_disconnect_from_ap(struct pi_device *device)
```

Close a connection to the access point.

This function closes an existing access point connection.

#### Parameters

- **device** – Pointer to the WIFI device structure.

**Returns 0** If operation is successful.

**Returns ERRNO** Error code otherwise.

```
void pi_wifi_data_send(struct pi_device *device, char *buffer, uint32_t size)
```

Send data to WIFI device. Blocking API.

This function is used to send data to WIFI device.

---

**Note:** This function is synchronous, caller is blocked until transfer is finished. The pending asynchronous function is below : [\*pi\\_wifi\\_data\\_send\\_async\(\)\*](#).

---

#### Parameters

- **device** – Pointer to the WIFI structure.
- **buffer** – Buffer to send.
- **size** – Size of data to send.

---

`ssize_t pi_wifi_data_send_async(struct pi_device *device, char *buffer, uint32_t size, pi_evt_t *task)`  
Send data to WIFI device. Blocking API.

This function is used to send data to WIFI device.

---

**Note:** This function is asynchronous. The pending synchronous function is : [\*pi\\_wifi\\_data\\_send\(\)\*](#).

#### Parameters

- **device** – Pointer to the WIFI structure.
- **buffer** – Buffer to send.
- **size** – Size of data to send.
- **task** – Event task used to check end of transfer.

`void pi_wifi_data_get(struct pi_device *device, char *buffer, uint32_t size)`  
Get data from WIFI device. Blocking API.

This function is used to retrieve data from WIFI device.

---

**Note:** This function is synchronous, caller is blocked until transfer is finished. The pending asynchronous function is below : [\*pi\\_wifi\\_data\\_get\\_async\(\)\*](#).

#### Parameters

- **device** – Pointer to the WIFI structure.
- **buffer** – Buffer to store data.
- **size** – Size of data.

`ssize_t pi_wifi_data_get_async(struct pi_device *device, char *buffer, uint32_t size, pi_evt_t *task)`  
Get data from WIFI device. Non blocking API.

This function is used to retrieve data from WIFI device.

---

**Note:** This function is asynchronous. The pending synchronous function is : [\*pi\\_wifi\\_data\\_get\(\)\*](#).

#### Parameters

- **device** – Pointer to the WIFI structure.
- **buffer** – Buffer to store data.
- **size** – Size of data.
- **task** – Event task used to check end of transfer.

`int8_t pi_wifi_create_tcp_server(struct pi_device *device, uint16_t port)`  
Open a TCP server socket on the device.

#### Parameters

- **device** – Device to configure as a TCP server
- **port** – Local port number of the socket

**Returns 0** for success

**Returns** int8\_t Code for success/error operation

```
int8_t pi_wifi_create_tcp_client(struct pi_device *device, char *server_ip, uint16_t server_port,  
                                uint16_t port)
```

Open a TCP client socket on the device.

#### Parameters

- **device** – Device to configure as a TCP client
- **server\_ip** – IP of the server
- **server\_port** – Port number of the server
- **port** – Local port number of the socket

**Returns 0** for success

**Returns** int8\_t Code for success/error operation

```
int8_t pi_wifi_catch_client_connection(struct pi_device *device, char *resp, pi_evt_t *callback)
```

Look for an incoming client connection on the device (pre-configured as a TCP server)

#### Parameters

- **device** – Device pre-configured as a TCP server
- **resp** – [out] Pointer for the response got from the client connection. Useful to know the IP and the port of the client.
- **callback** – [in] Task to notify when a client is connecting.

**Returns 0** for success

**Returns** int8\_t Code for success/error operation

```
int8_t pi_wifi_catch_event(struct pi_device *device, char *resp, pi_evt_t *callback)
```

Look for an incoming event, and catch it.

#### Parameters

- **device** – Device pre-configured as a TCP server or a TCP client
- **resp** – [out] Buffer to store the event
- **callback** – [in] Task to notify when an event appears

**Returns 0** for success

**Returns** int8\_t Code for success/error operation

```
struct pi_wifi_api_s  
#include <wifi.h> WIFI specific API.
```

Structure holding WIFI specific API.

## Public Members

`int32_t (*at_cmd)(struct pi_device *device, char *cmd, char *resp)`  
Function to send AT command.

`int8_t (*connect_to_ap)(struct pi_device *device, const char *ssid, const char *pwd, pi_evt_t *callback)`  
Function to connect WIFI device to an AP.

`int8_t (*disconnect_from_ap)(struct pi_device *device)`  
Function to disconnect WIFI device from a currently associated AP.

## DA16200

### group DA16200

The da16200 driver provides support for data transfer using a WiFi module, here a da16200 WiFi module. This module is interfaced on GAP9\_EVK through UART. SPI isn't supported for now.

### Enums

`enum bsp_da16200_ioctl_cmd_e`  
IOCTL commands for DA16200 module.

*Values:*

`enumerator BSP_DA16200_TCP_SERVER_CONFIGURE`  
Configure device as a tcp server.

`enumerator BSP_DA16200_TCP_CLIENT_CONFIGURE`  
Configure device as a tcp client.

`enumerator BSP_DA16200_UART_CONFIGURE`  
Configure HCI UART.

### Functions

`void pi_wifi_da16200_conf_init(struct pi_device *device, struct pi_da16200_conf *conf)`  
Initialize DA16200 configuration structure.

#### Parameters

- `device` – [out] Pointer to the DA16200 device
- `conf` – [out] Pointer to the DA16200 configuration structure

`void pi_wifi_da16200_client_ip_port_set(struct pi_device *device, const char *client_ip_port)`  
Set struct da16200\_t's client\_ip\_port field.

#### Parameters

- `device` – [in]
- `client_ip_port` – [in]

```
struct pi_da16200_conf
    #include <da16200.h> DA16200 configuration structure.
```

### Public Members

**uint8\_t uart\_if**  
UART interface used to connect WiFi module

**uint32\_t uart\_baudrate**  
UART baudrate

**char uart\_parity\_bits**  
UART parity bits: n (None), e (Even), o (Odd)

**uint8\_t stop\_bits**  
UART stop bits

**uint8\_t use\_ctrl\_flow**  
UART flow control

## 7.3 Builtins

TODO put gap9 builtins

## PYTHON MODULE INDEX

### g

gap9.evk, 392  
gap9.gap9\_v2, 392  
gapylib.chips.gap.flash, 392  
gapylib.chips.gap.gap9\_v2.board\_runner, 395  
gapylib.chips.gap.rom\_v2, 393  
gapylib.flash, 377  
gapylib.fs.hostfs, 391  
gapylib.fs.littlefs, 389  
gapylib.fs.partition, 387  
gapylib.fs.raw, 391  
gapylib.fs.readfs, 388  
gapylib.target, 375  
gapylib.utils, 383  
gv.gvsoc\_control, 333  
gvsoc\_control, 581

### n

nntool.api, 411  
nntool.api.compression, 433  
nntool.api.quantization, 433  
nntool.api.types, 434  
nntool.api.utils, 424  
nntool.api.validation, 435



# INDEX

## Symbols

[anonymous] (C enum), 573	ads1014_comparator_mode.ADS1014_COMPARATOR_MODE_TRADITIONAL (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_CLR (C enumerator), 573	ads1014_comparator_mode.ADS1014_COMPARATOR_MODE_WINDOW (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_EVT_ALLOC (C enumerator), 573	ads1014_comparator_polarity (C enum), 590 ads1014_comparator_polarity.ADS1014_COMPARATOR_POLARITY_AC (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_FREE_EVT (C enumerator), 573	ads1014_comparator_polarity.ADS1014_COMPARATOR_POLARITY_AC (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_FREE_INPUT (C enumerator), 573	ads1014_comparator_status (C enum), 590 ads1014_comparator_status.ADS1014_COMPARATOR_STATUS_ASSERT (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_GET_FIFO_ID (C enumerator), 573	ads1014_comparator_status.ADS1014_COMPARATOR_STATUS_ASSERT (C enumerator), 591
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_GET_LIN_ID (C enumerator), 573	ads1014_comparator_status.ADS1014_COMPARATOR_STATUS_ASSERT (C enumerator), 591
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_SET_DEST (C enumerator), 573	ads1014_comparator_status.ADS1014_COMPARATOR_STATUS_ASSERT (C enumerator), 591
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_SET_EVT (C enumerator), 573	ads1014_comparator_status.ADS1014_COMPARATOR_STATUS_DISABLE (C enumerator), 591
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_SET_INPUT (C enumerator), 573	ads1014_data_rate (C enum), 589 ads1014_data_rate.ADS1014_DATA_RATE_SPS_128 (C enumerator), 590
[anonymous].PI_UDMA_TIMESTAMP_IOCTL_STOP (C enumerator), 573	ads1014_data_rate.ADS1014_DATA_RATE_SPS_1600 (C enumerator), 590
A	ads1014_data_rate.ADS1014_DATA_RATE_SPS_2400 (C enumerator), 590
add_constant() (nntool.api.NNGraph method), 411	ads1014_data_rate.ADS1014_DATA_RATE_SPS_250 (C enumerator), 590
add_dimensions() (nntool.api.NNGraph method), 411	ads1014_data_rate.ADS1014_DATA_RATE_SPS_3300 (C enumerator), 590
add_field() (gapylib.utils.CStruct method), 383	ads1014_data_rate.ADS1014_DATA_RATE_SPS_490 (C enumerator), 590
add_field_array() (gapylib.utils.CStruct method), 383	ads1014_data_rate.ADS1014_DATA_RATE_SPS_920 (C enumerator), 590
add_input() (nntool.api.NNGraph method), 411	ads1014_operating_mode (C enum), 589
add_output() (nntool.api.NNGraph method), 411	ads1014_operating_mode.ADS1014_OPERATING_MODE_CONTINUOUS (C enumerator), 589
add_padding() (gapylib.utils.CStruct method), 383	ads1014_operating_mode.ADS1014_OPERATING_MODE_SINGLE_SHOT (C enumerator), 589
add_struct() (gapylib.flash.FlashSection method), 380	ads1014_pga (C enum), 589
add_struct() (gapylib.utils.CStructParent method), 385	ads1014_pga.ADS1014_PGA_FSR_0V256 (C enumerator), 589
adjust_order() (nntool.api.NNGraph method), 412	ads1014_pga.ADS1014_PGA_FSR_0V512 (C enumerator)
ads1014_comparator_latch (C enum), 590	
ads1014_comparator_latch.ADS1014_COMPARATOR_LATCH_DISABLED (C enumerator), 590	
ads1014_comparator_latch.ADS1014_COMPARATOR_LATCH_ENABLED (C enumerator), 590	
ads1014_comparator_mode (C enum), 590	

*tor), 589*  
*ads1014\_pga.ADS1014\_PGA\_FSR\_1V024 (C enumerator), 589*  
*ads1014\_pga.ADS1014\_PGA\_FSR\_2V048 (C enumerator), 589*  
*ads1014\_pga.ADS1014\_PGA\_FSR\_4V096 (C enumerator), 589*  
*ads1014\_pga.ADS1014\_PGA\_FSR\_6V144 (C enumerator), 589*  
*ak4332\_bit\_t (C enum), 694*  
*ak4332\_bit\_t.AK4332\_RESET (C enumerator), 694*  
*ak4332\_bit\_t.AK4332\_SET (C enumerator), 695*  
*ak4332\_cfg\_register\_t (C enum), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_AIF (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_CMS (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DA1 (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DA2 (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DCD (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DCS (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DFS (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DMM (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_DOV (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_HPVC (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_OMS (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PCSS (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PDMERR (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PFCD1 (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PFCD2 (C enumerator), 694*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PIC (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PM1 (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PM2 (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PM3 (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PM4 (C enumerator), 693*  
*ak4332\_cfg\_register\_t.AK4332\_REG\_PRCD1 (C enumerator), 693*  
*ak4332\_clk\_t (C struct), 706*  
*ak4332\_clk\_t.cm (C var), 707*  
*ak4332\_clk\_t.dac\_clksrc (C var), 707*  
*ak4332\_clk\_t.fs (C var), 707*  
*ak4332\_clk\_t.pll (C var), 707*  
*ak4332\_cm\_t (C enum), 698*  
*ak4332\_cm\_t.CM\_1024\_FS (C enumerator), 698*  
*ak4332\_cm\_t.CM\_128\_FS (C enumerator), 698*  
*ak4332\_cm\_t.CM\_256\_FS (C enumerator), 698*  
*ak4332\_cm\_t.CM\_512\_FS (C enumerator), 698*  
*ak4332\_cpmode\_t (C enum), 695*  
*ak4332\_cpmode\_t.CPLODE\_HALF\_VDD (C enumerator), 696*  
*ak4332\_cpmode\_t.CPMODE\_CLASSG (C enumerator), 695*  
*ak4332\_cpmode\_t.CPMODE\_VDD (C enumerator), 695*  
*ak4332\_dac\_clksrc\_t (C enum), 701*  
*ak4332\_dac\_clksrc\_t.DAC\_CLKSRC\_MCKI (C enumerator), 701*  
*ak4332\_dac\_clksrc\_t.DAC\_CLKSRC\_PLLCLK (C enumerator), 701*  
*ak4332\_dac\_input\_t (C enum), 698*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_AVERAGE\_LEFT\_RIGHT (C enumerator), 699*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_HALF\_LEFT (C enumerator), 699*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_HALF\_RIGHT (C enumerator), 699*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_LEFT (C enumerator), 698*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_LEFT\_AND\_RIGHT (C enumerator), 699*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_MUTE (C enumerator), 698*  
*ak4332\_dac\_input\_t.DAC\_INPUT\_RIGHT (C enumerator), 699*  
*ak4332\_dac\_polarity\_t (C enum), 699*  
*ak4332\_dac\_polarity\_t.DAC\_POLARITY\_INVERTED (C enumerator), 699*  
*ak4332\_dac\_polarity\_t.DAC\_POLARITY\_NORMAL (C enumerator), 699*  
*ak4332\_dac\_t (C struct), 708*  
*ak4332\_dac\_t.filter (C var), 708*  
*ak4332\_dac\_t.input (C var), 708*  
*ak4332\_dac\_t.polarity (C var), 708*  
*ak4332\_dac\_t.volume (C var), 708*  
*ak4332\_datatype\_t (C enum), 699*  
*ak4332\_datatype\_t.AK4332\_DSD (C enumerator), 699*  
*ak4332\_datatype\_t.AK4332\_PCM (C enumerator), 699*  
*ak4332\_datatype\_t.AK4332\_PDM (C enumerator), 699*  
*ak4332\_dsdclk\_pol\_t (C enum), 699*

ak4332\_dsdclk\_pol\_t.AK4332\_DSDCLK\_F\_EDGE (C enumerator), 699  
 ak4332\_dsdclk\_pol\_t.AK4332\_DSDCLK\_R\_EDGE (C enumerator), 699  
 ak4332\_en\_t (C enum), 694  
 ak4332\_en\_t.AK4332\_DISABLE (C enumerator), 694  
 ak4332\_en\_t.AK4332\_ENABLE (C enumerator), 694  
 ak4332\_filter\_t (C enum), 698  
 ak4332\_filter\_t.SHARP\_ROLL\_OFF (C enumerator), 698  
 ak4332\_filter\_t.SHORT\_DELAY\_SHARP\_ROLL\_OFF (C enumerator), 698  
 ak4332\_filter\_t.SHORT\_DELAY\_SLOW\_ROLL\_OFF (C enumerator), 698  
 ak4332\_filter\_t.SLOW\_ROLL\_OFF (C enumerator), 698  
 ak4332\_fs\_t (C enum), 697  
 ak4332\_fs\_t.FS\_11\_025\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_128\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_12\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_16\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_176\_4\_KHZ (C enumerator), 698  
 ak4332\_fs\_t.FS\_192\_KHZ (C enumerator), 698  
 ak4332\_fs\_t.FS\_22\_05\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_24\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_32\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_44\_1\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_48\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_64\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_88\_2\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_8\_KHZ (C enumerator), 697  
 ak4332\_fs\_t.FS\_96\_KHZ (C enumerator), 697  
 ak4332\_hp\_amp\_t (C struct), 708  
 ak4332\_hp\_amp\_t.cpmode (C var), 708  
 ak4332\_hp\_amp\_t.hpohz (C var), 708  
 ak4332\_hp\_amp\_t.hptm (C var), 708  
 ak4332\_hp\_amp\_t.lvdsel (C var), 708  
 ak4332\_hp\_amp\_t.lvdtm (C var), 708  
 ak4332\_hp\_amp\_t.vddtm (C var), 708  
 ak4332\_hp\_amp\_t.volume (C var), 708  
 ak4332\_hpohz\_t (C enum), 696  
 ak4332\_hpohz\_t.HPOHZ\_4 (C enumerator), 697  
 ak4332\_hpohz\_t.HPOHZ\_95 (C enumerator), 697  
 ak4332\_hptm\_t (C enum), 700  
 ak4332\_hptm\_t.HPTM\_1024\_FS (C enumerator), 700  
 ak4332\_hptm\_t.HPTM\_128\_FS (C enumerator), 700  
 ak4332\_hptm\_t.HPTM\_2048\_FS (C enumerator), 700  
 ak4332\_hptm\_t.HPTM\_256\_FS (C enumerator), 700  
 ak4332\_hptm\_t.HPTM\_512\_FS (C enumerator), 700  
 ak4332\_init\_struct\_t (C struct), 709  
 ak4332\_init\_struct\_t.clk (C var), 709  
 ak4332\_init\_struct\_t.dac (C var), 709  
 ak4332\_init\_struct\_t.hp (C var), 709  
 ak4332\_init\_struct\_t.sai (C var), 709  
 ak4332\_lvdsel\_t (C enum), 696  
 ak4332\_lvdsel\_t.LVDSEL\_11 (C enumerator), 696  
 ak4332\_lvdsel\_t.LVDSEL\_16 (C enumerator), 696  
 ak4332\_lvdsel\_t.LVDSEL\_32 (C enumerator), 696  
 ak4332\_lvdsel\_t.LVDSEL\_8 (C enumerator), 696  
 ak4332\_lvdtm\_t (C enum), 695  
 ak4332\_lvdtm\_t.LVDTM\_1024 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_128 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_2048 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_256 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_4096 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_512 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_64 (C enumerator), 695  
 ak4332\_lvdtm\_t.LVDTM\_8192 (C enumerator), 695  
 ak4332\_pdm\_fs\_t (C enum), 702  
 ak4332\_pdm\_fs\_t.AK4332\_PDM\_FULL\_SCALE (C enumerator), 702  
 ak4332\_pdm\_fs\_t.AK4332\_PDM\_NO\_FULL\_SCALE (C enumerator), 702  
 ak4332\_pdm\_mute\_t (C enum), 700  
 ak4332\_pdm\_mute\_t.AK4332\_PDM\_DATA\_MUTE\_DISABLE (C enumerator), 700  
 ak4332\_pdm\_mute\_t.AK4332\_PDM\_DATA\_MUTE\_ENABLE (C enumerator), 700  
 ak4332\_pdm\_options\_t (C struct), 707  
 ak4332\_pdm\_options\_t.data\_mute (C var), 707  
 ak4332\_pdm\_options\_t.dsd\_clk\_pol (C var), 707  
 ak4332\_pdm\_options\_t.pdm\_clk\_pol (C var), 707  
 ak4332\_pdm\_options\_t.pdm\_fullscale (C var), 707  
 ak4332\_pdmclk\_pol\_t (C enum), 699  
 ak4332\_pdmclk\_pol\_t.AK4332\_PDMCLK\_F\_EDGE (C enumerator), 700  
 ak4332\_pdmclk\_pol\_t.AK4332\_PDMCLK\_R\_EDGE (C enumerator), 700  
 ak4332\_pll\_clksrc\_t (C enum), 700  
 ak4332\_pll\_clksrc\_t.PLL\_CLKSRC\_BCLK (C enumerator), 700  
 ak4332\_pll\_clksrc\_t.PLL\_CLKSRC\_MCKI (C enumerator), 700  
 ak4332\_pll\_en\_t (C enum), 695  
 ak4332\_pll\_en\_t.PLL\_DISABLE (C enumerator), 695  
 ak4332\_pll\_en\_t.PLL\_ENABLE (C enumerator), 695  
 ak4332\_pll\_mode\_t (C enum), 700  
 ak4332\_pll\_mode\_t.PLLMD\_HIGH\_F\_MODE (C enumerator), 700  
 ak4332\_pll\_mode\_t.PLLMD\_LOW\_F\_MODE (C enumerator), 701  
 ak4332\_pll\_t (C struct), 706  
 ak4332\_pll\_t.en (C var), 706  
 ak4332\_pll\_t.mdiv (C var), 706  
 ak4332\_pll\_t.mode (C var), 706  
 ak4332\_pll\_t.pld (C var), 706  
 ak4332\_pll\_t.plm (C var), 706  
 ak4332\_pll\_t.src (C var), 706

ak4332\_pm\_t (*C enum*), 694  
 ak4332\_pm\_t.AK4332\_PM\_DOWN (*C enumerator*), 694  
 ak4332\_pm\_t.AK4332\_PM\_UP (*C enumerator*), 694  
 ak4332\_preset\_e (*C enum*), 692  
 ak4332\_preset\_e.AK4332\_PCM\_SLAVE\_PLL\_BCLK\_32WS\_48KHZ 386  
     (*C enumerator*), 692  
 ak4332\_preset\_e.AK4332\_PDM\_DSD\_SLAVE\_PLL\_BCLK\_32WS\_48KHZic 395  
     (*C enumerator*), 692  
 ak4332\_preset\_e.AK4332\_PRESETS\_NUMBER (*C enumerator*), 692  
 ak4332\_sai\_bclk\_out\_t (*C enum*), 701  
 ak4332\_sai\_bclk\_out\_t.AK4332\_BCLK\_32FS (C  
     enumerator), 702  
 ak4332\_sai\_bclk\_out\_t.AK4332\_BCLK\_64FS (C  
     enumerator), 702  
 ak4332\_sai\_format\_t (*C enum*), 701  
 ak4332\_sai\_format\_t.FORMAT\_I2S (*C enumerator*),  
     701  
 ak4332\_sai\_format\_t.FORMAT\_MSB\_JUSTIFIED (*C  
     enumerator*), 701  
 ak4332\_sai\_mode\_t (*C enum*), 701  
 ak4332\_sai\_mode\_t.AK4332\_MASTER (*C enumerator*),  
     701  
 ak4332\_sai\_mode\_t.AK4332\_SLAVE (*C enumerator*),  
     701  
 ak4332\_sai\_t (*C struct*), 707  
 ak4332\_sai\_t.bclk\_master (*C var*), 707  
 ak4332\_sai\_t.data\_type (*C var*), 707  
 ak4332\_sai\_t.format (*C var*), 707  
 ak4332\_sai\_t.mode (*C var*), 707  
 ak4332\_sai\_t.pdm\_options (*C var*), 708  
 ak4332\_sai\_t.word\_size (*C var*), 707  
 ak4332\_vddtm\_t (*C enum*), 696  
 ak4332\_vddtm\_t.VDDTM\_1024 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_131072 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_16384 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_2048 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_262144 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_32768 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_4096 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_65536 (*C enumerator*), 696  
 ak4332\_vddtm\_t.VDDTM\_8192 (*C enumerator*), 696  
 ak4332\_wordsize\_t (*C enum*), 701  
 ak4332\_wordsize\_t.DATALENGTH\_16 (*C enumerator*),  
     701  
 ak4332\_wordsize\_t.DATALENGTH\_24 (*C enumerator*),  
     701  
 ak4332\_wordsize\_t.DATALENGTH\_32 (*C enumerator*),  
     701  
 align\_offset() (*gapylib.flash.FlashSection method*),  
     380  
 align\_offset() (*gapylib.utils.CStructParent method*),  
     385  
 all\_constants (*nntool.api.NNGraph property*), 412  
 all\_expressions (*nntool.api.NNGraph property*), 412  
 alloc\_offset() (*gapylib.flash.FlashSection method*),  
     380  
 alloc\_offset() (*gapylib.utils.CStructParent method*),  
     386  
     append\_args() (*gapylib.chips.gap.gap9\_v2.board\_runner.Runner  
         ic method*), 395  
     append\_args() (*gapylib.target.Target method*), 375  
 at\_options (*nntool.api.types.NNNodeBase property*),  
     434  
 AutoCompress (*class in nntool.api.compression*), 433  
 AveragePoolNode (*class in nntool.api.types*), 434

## B

balance\_filters() (*nntool.api.NNGraph method*),  
     412  
 base (*gapylib.chips.gap.rom\_v2.BinarySegment attribute*), 393  
 Binary (*class in gapylib.chips.gap.rom\_v2*), 393  
 BinarySegment (*class in gapylib.chips.gap.rom\_v2*), 393  
 bsp\_ak4332\_pd़n\_t (*C struct*), 709  
 bsp\_ak4332\_pd़n\_t.context (*C var*), 709  
 bsp\_ak4332\_pd़n\_t.id (*C var*), 709  
 bsp\_da16200\_ioctl\_cmd\_e (*C enum*), 715  
 bsp\_da16200\_ioctl\_cmd\_e.BSP\_DA16200\_TCP\_CLIENT\_CONFIGURE  
     (*C enumerator*), 715  
 bsp\_da16200\_ioctl\_cmd\_e.BSP\_DA16200\_TCP\_SERVER\_CONFIGURE  
     (*C enumerator*), 715  
 bsp\_da16200\_ioctl\_cmd\_e.BSP\_DA16200\_UART\_CONFIGURE  
     (*C enumerator*), 715  
 bsp\_tlv320\_shdnz\_t (*C struct*), 690

## C

clk\_start() (*gv.gvsoc\_control.Testbench\_i2s method*),  
     336  
 clk\_start() (*gvsoc\_control.Testbench\_i2s method*),  
     584  
 clk\_stop() (*gv.gvsoc\_control.Testbench\_i2s method*),  
     336  
 clk\_stop() (*gvsoc\_control.Testbench\_i2s method*), 584  
 close() (*gv.gvsoc\_control.Proxy method*), 333  
 close() (*gv.gvsoc\_control.Testbench\_i2s method*), 336  
 close() (*gv.gvsoc\_control.Testbench\_uart method*), 339  
 close() (*gvsoc\_control.Proxy method*), 581  
 close() (*gvsoc\_control.Testbench\_i2s method*), 584  
 close() (*gvsoc\_control.Testbench\_uart method*), 587  
 collect\_statistics() (*nntool.api.NNGraph method*),  
     412  
 compress\_constant() (*nntool.api.NNGraph method*),  
     413  
 Conv2DNode (*class in nntool.api.types*), 434  
 cos\_sim() (*nntool.api.NNGraph static method*), 413  
 cos\_sims() (*in module nntool.api.utils*), 424  
 CStruct (*class in gapylib.utils*), 383

`CStructArray` (*class in gapylib.utils*), 384  
`CStructField` (*class in gapylib.utils*), 385  
`CStructParent` (*class in gapylib.utils*), 385  
`CStructScalar` (*class in gapylib.utils*), 386

**D**

`data` (*gapylib.chips.gap.rom\_v2.BinarySegment attribute*), 393  
`declare_property()` (*gapylib.flash.FlashSection method*), 380  
`DefaultFlashRomV2` (*class in gapylib.chips.gap.flash*), 392  
`description` (*gapylib.flash.FlashSectionProperty attribute*), 383  
`draw()` (*nntool.api.NNGraph method*), 413  
`dump_image()` (*gapylib.flash.Flash method*), 377  
`dump_layout()` (*gapylib.flash.Flash method*), 377  
`dump_properties()` (*gapylib.flash.FlashSection method*), 380  
`dump_section_description()` (*gapylib.flash.FlashSection method*), 380  
`dump_section_properties()` (*gapylib.flash.Flash method*), 378  
`dump_sections()` (*gapylib.flash.Flash method*), 378  
`dump_table()` (*gapylib.flash.FlashSection method*), 380  
`dump_table()` (*gapylib.utils.CStruct method*), 384  
`dump_table()` (*gapylib.utils.CStructArray method*), 384  
`dump_table()` (*gapylib.utils.CStructParent method*), 386  
`dump_table()` (*gapylib.utils.CStructScalar method*), 386

**E**

`event_add()` (*gv.gvsoc\_control.Proxy method*), 333  
`event_add()` (*gvsoc\_control.Proxy method*), 581  
`event_remove()` (*gv.gvsoc\_control.Proxy method*), 333  
`event_remove()` (*gvsoc\_control.Proxy method*), 581  
`execute()` (*nntool.api.NNGraph method*), 414  
`execute_on_target()` (*nntool.api.NNGraph method*), 414

**F**

`fake()` (*nntool.api.utils.RandomIter class method*), 424  
`fd` (*gapylib.chips.gap.rom\_v2.Binary attribute*), 393  
`FileImporter` (*class in nntool.api.utils*), 424  
`filename` (*nntool.api.NNGraph property*), 415  
`fill_header()` (*gapylib.chips.gap.rom\_v2.Xip method*), 395  
`finalize()` (*gapylib.chips.gap.rom\_v2.RomFlashSection method*), 394  
`finalize()` (*gapylib.flash.FlashSection method*), 380  
`finalize()` (*gapylib.fs.partition.PartitionTableSection method*), 387  
`Flash` (*class in gapylib.flash*), 377

`flash` (*gapylib.chips.gap.rom\_v2.Xip attribute*), 395  
`flash()` (*gapylib.chips.gap.gap9\_v2.board\_runner.Runner method*), 395  
`flash_attributes` (*gapylib.flash.Flash attribute*), 377  
`FlashSection` (*class in gapylib.flash*), 379  
`FlashSectionProperty` (*class in gapylib.flash*), 382  
`func_t` (*C type*), 438  
`fusions()` (*nntool.api.NNGraph method*), 415  
`fxl6408_gpio_dir_e` (*C enum*), 627  
`fxl6408_gpio_dir_e.FXL6408_GPIO_DIR_INPUT` (*C enumerator*), 627  
`fxl6408_gpio_dir_e.FXL6408_GPIO_DIR_OUTPUT` (*C enumerator*), 627  
`fxl6408_gpio_input_trigger_e` (*C enum*), 627  
`fxl6408_gpio_input_trigger_e.FXL6408_GPIO_INPUT_TRIGGER_EDGE_RISING` (*C enumerator*), 628  
`fxl6408_gpio_input_trigger_e.FXL6408_GPIO_INPUT_TRIGGER_FALLING` (*C enumerator*), 627  
`fxl6408_gpio_input_trigger_e.FXL6408_GPIO_INPUT_TRIGGER_RISING` (*C enumerator*), 627  
`fxl6408_gpio_output_state_e` (*C enum*), 627  
`fxl6408_gpio_output_state_e.FXL6408_GPIO_OUTPUT_STATE_DISABLED` (*C enumerator*), 627  
`fxl6408_gpio_output_state_e.FXL6408_GPIO_OUTPUT_STATE_HIGH` (*C enumerator*), 627  
`fxl6408_gpio_output_state_e.FXL6408_GPIO_OUTPUT_STATE_LOW` (*C enumerator*), 627  
`fxl6408_gpio_pull_state_e` (*C enum*), 628  
`fxl6408_gpio_pull_state_e.FXL6408_GPIO_PULL_STATE_DISABLED` (*C enumerator*), 628  
`fxl6408_gpio_pull_state_e.FXL6408_GPIO_PULL_STATE_DOWN` (*C enumerator*), 628  
`fxl6408_gpio_pull_state_e.FXL6408_GPIO_PULL_STATE_UP` (*C enumerator*), 628  
`FXL6408_IS_VALID_OUTPUT_STATE` (*C macro*), 627

**G**

`GAP9._pi_pmu_domain_state` (*C enum*), 538  
`GAP9._pi_pmu_domain_state.PI_PMU_DOMAIN_STATE_DEEP_SLEEP` (*C enumerator*), 538  
`GAP9._pi_pmu_domain_state.PI_PMU_DOMAIN_STATE_DEEP_SLEEP_F` (*C enumerator*), 538  
`GAP9._pi_pmu_domain_state.PI_PMU_DOMAIN_STATE_OFF` (*C enumerator*), 538  
`GAP9._pi_pmu_domain_state.PI_PMU_DOMAIN_STATE_ON` (*C enumerator*), 538  
`gap9.evk`  
  `module`, 392  
`gap9.gap9_v2`  
  `module`, 392  
`GAP9.pi_pmu_boot_state_get` (*C function*), 539  
`GAP9.pi_pmu_domain_e` (*C enum*), 537  
`GAP9.pi_pmu_domain_e.PI_PMU_DOMAIN_CHIP` (*C enumerator*), 538

GAP9.pi\_pmu\_domain\_e.PI\_PMU\_DOMAIN\_CL (*C enumerator*), 537  
 GAP9.pi\_pmu\_domain\_e.PI\_PMU\_DOMAIN\_CSI2 (*C enumerator*), 537  
 GAP9.pi\_pmu\_domain\_e.PI\_PMU\_DOMAIN\_MRAMEL (C enumerator), 538  
 GAP9.pi\_pmu\_domain\_e.PI\_PMU\_DOMAIN\_SFU (C enumerator), 538  
 GAP9.pi\_pmu\_domain\_state\_change (*C function*), 541  
 GAP9.pi\_pmu\_domain\_state\_change\_async (*C function*), 541  
 GAP9.pi\_pmu\_domain\_state\_e (*C type*), 537  
 GAP9.pi\_pmu\_domain\_state\_flags (*C enum*), 538  
 GAP9.pi\_pmu\_domain\_state\_flags.PI\_PMU\_DOMAIN\_SFUEL (C enumerator), 538  
 GAP9.pi\_pmu\_domain\_state\_flags\_e (*C type*), 537  
 GAP9.pi\_pmu\_gpio\_wakeup\_pins (*C function*), 540  
 GAP9.pi\_pmu\_voltage\_domain\_e (*C enum*), 539  
 GAP9.pi\_pmu\_voltage\_domain\_e.PI\_PMU\_VOLTAGE\_DOMAIN\_MCHP (C enumerator), 539  
 GAP9.pi\_pmu\_voltage\_get (*C function*), 540  
 GAP9.pi\_pmu\_voltage\_set (*C function*), 540  
 GAP9.pi\_pmu\_wakeup\_control (*C function*), 539  
 GAP9.pi\_pmu\_wakeup\_e (*C enum*), 538  
 GAP9.pi\_pmu\_wakeup\_e.PI\_PMU\_WAKEUP\_GPIO (C enumerator), 538  
 GAP9.pi\_pmu\_wakeup\_e.PI\_PMU\_WAKEUP\_RTC (C enumerator), 538  
 GAP9.pi\_pmu\_wakeup\_e.PI\_PMU\_WAKEUP\_SPISLAVE (C enumerator), 538  
 GAP9.pi\_pmu\_wakeup\_fll\_settings (*C function*), 540  
 GAP9.pi\_pmu\_wakeup\_reason (*C function*), 540  
 GAP9.pi\_pmu\_wakeup\_sequence (*C function*), 540  
 GAP9.pi\_pmu\_wakeup\_sequence\_e (*C enum*), 538  
 GAP9.pi\_pmu\_wakeup\_sequence\_e.PI\_PMU\_WAKEUP\_SEQUENCE\_SOC (C enumerator), 539  
 GAP9.pi\_pmu\_wakeup\_sequence\_e.PI\_PMU\_WAKEUP\_SEQUENCE\_SOC\_MRAMEL (C enumerator), 539  
 gapylib.chips.gap.flash module, 392  
 gapylib.chips.gap.gap9\_v2.board\_runner module, 395  
 gapylib.chips.gap.rom\_v2 module, 393  
 gapylib.flash module, 377  
 gapylib.fs.hostfs module, 391  
 gapylib.fs.littlefs module, 389  
 gapylib.fs.partition module, 387  
 gapylib.fs.raw module, 391  
 gapylib.NO\_FLAGSadfs module, 388  
 gapylib.target module, 375  
 gapylib.utils module, 381  
 gen\_project() (*nntool.api.NNGraph method*), 416  
 get\_abspath() (*gapylib.target.Target method*), 376  
 get\_args() (*gapylib.target.Target method*), 376  
 get\_current\_offset() (*gapylib.flash.FlashSection method*), 381  
 get\_current\_offset() (*gapylib.utils.CStructParent method*), 386  
 get\_field() (*gapylib.utils.CStruct method*), 384  
 get\_file\_path() (*gapylib.target.Target static method*), 376  
 get\_flash() (*gapylib.flash.FlashSection method*), 381  
 get\_flash\_attribute() (*gapylib.flash.Flash method*), 378  
 get\_fusions() (*nntool.api.NNGraph static method*), 416  
 get\_id() (*gapylib.flash.FlashSection method*), 381  
 get\_image() (*gapylib.flash.FlashSection method*), 381  
 get\_image\_name() (*gapylib.flash.Flash method*), 378  
 get\_image\_name() (*gapylib.flash.FlashSection method*), 381  
 get\_image\_path() (*gapylib.flash.FlashSection method*), 378  
 get\_name() (*gapylib.flash.Flash method*), 378  
 get\_name() (*gapylib.utils.CStruct method*), 384  
 get\_offset() (*gapylib.flash.FlashSection method*), 381  
 get\_offset() (*gapylib.utils.CStruct method*), 384

**get\_offset()** (*gapylib.utils.CStructField method*), 385  
**get\_page\_size()** (*gapylib.chips.gap.rom\_v2.Xip method*), 395  
**get\_partition\_type()** (*gapylib.flash.FlashSection method*), 381  
**get\_partition\_type()** (*gapylib.fs.littlefs.LfsSection method*), 390  
**get\_partition\_type()** (*gapylib.fs.raw.RawSection method*), 392  
**get\_partition\_type()** (*gapylib.fs.readfs.ReadfsSection method*), 389  
**get\_property()** (*gapylib.flash.FlashSection method*), 382  
**get\_section\_by\_name()** (*gapylib.flash.Flash method*), 378  
**get\_sections()** (*gapylib.flash.Flash method*), 378  
**get\_size()** (*gapylib.flash.Flash method*), 379  
**get\_size()** (*gapylib.flash.FlashSection method*), 382  
**get\_size()** (*gapylib.utils.CStruct method*), 384  
**get\_target()** (*gapylib.flash.Flash method*), 379  
**get\_target()** (*in module gapylib.target*), 377  
**get\_working\_dir()** (*gapylib.target.Target method*), 376  
**GlobalAveragePoolNode** (*class in nnntool.api.types*), 434  
**GlobalMaxPoolNode** (*class in nnntool.api.types*), 434  
**GlobalMinPoolNode** (*class in nnntool.api.types*), 434  
**GlobalPoolingNodeBase** (*class in nnntool.api.types*), 434  
**GlobalSumPoolNode** (*class in nnntool.api.types*), 434  
**GRUNode** (*class in nnntool.api.types*), 435  
**gv.gvsoc\_control module**, 333  
**gv\_pcer\_conf** (*C function*), 579  
**gv\_pcer\_dump** (*C function*), 580  
**gv\_pcer\_dump\_end** (*C function*), 580  
**gv\_pcer\_dump\_start** (*C function*), 580  
**gv\_pcer\_read** (*C function*), 580  
**gv\_pcer\_reset** (*C function*), 579  
**gv\_pcer\_start** (*C function*), 579  
**gv\_pcer\_stop** (*C function*), 579  
**gv\_stop** (*C function*), 578  
**gv\_trace\_disable** (*C function*), 579  
**gv\_trace\_enable** (*C function*), 579  
**gv\_vcd\_conf\_t** (*C struct*), 581  
**gv\_vcd\_configure** (*C function*), 580  
**gv\_vcd\_dump\_trace** (*C function*), 581  
**gv\_vcd\_dump\_trace\_string** (*C function*), 581  
**gv\_vcd\_open\_trace** (*C function*), 580  
**gv\_vcd\_release\_trace** (*C function*), 581  
**gvsoc\_control module**, 581

## H

**handle\_command()** (*gapylib.target.Target method*), 376  
**has\_dsp** (*nnntool.api.NNGraph property*), 416  
**has\_expressions** (*nnntool.api.NNGraph property*), 416  
**has\_node\_type()** (*nnntool.api.NNGraph method*), 417  
**has\_quantized\_parameters** (*nnntool.api.NNGraph property*), 417  
**has\_resizer** (*nnntool.api.NNGraph property*), 417  
**has\_rnn()** (*nnntool.api.NNGraph method*), 417  
**has\_ssd\_postprocess** (*nnntool.api.NNGraph property*), 417  
**hist\_compare\_tensors()** (*nnntool.api.NNGraph static method*), 417  
**HostfsSection** (*class in gapylib.fs.hostfs*), 391  
**hyperflash\_conf** (*C struct*), 615

## I

**i2s\_get()** (*gv.gvsoc\_control.Testbench method*), 336  
**i2s\_get()** (*gvsoc\_control.Testbench method*), 584  
**id** (*gapylib.flash.FlashSection attribute*), 379  
**image\_name** (*gapylib.flash.Flash attribute*), 377  
**import\_data()** (*in module nnntool.api.utils*), 424  
**input\_nodes()** (*nnntool.api.NNGraph method*), 417  
**inputs\_and\_constants()** (*nnntool.api.NNGraph method*), 417  
**insert\_dsp\_preprocessing()** (*nnntool.api.NNGraph method*), 418  
**is\_empty()** (*gapylib.chips.gap.rom\_v2.RomFlashSection method*), 394  
**is\_empty()** (*gapylib.flash.Flash method*), 379  
**is\_empty()** (*gapylib.flash.FlashSection method*), 382  
**is\_empty()** (*gapylib.fs.littlefs.LfsSection method*), 390  
**is\_empty()** (*gapylib.fs.partition.PartitionTableSection method*), 387  
**is\_empty()** (*gapylib.fs.readfs.ReadfsSection method*), 389  
**is\_xip\_segment()** (*gapylib.chips.gap.rom\_v2.Xip method*), 395

## L

**LfsHeader** (*class in gapylib.fs.littlefs*), 389  
**LfsSection** (*class in gapylib.fs.littlefs*), 390  
**LinearNode** (*class in nnntool.api.types*), 434  
**load\_graph()** (*nnntool.api.NNGraph static method*), 418  
**LSTMNode** (*class in nnntool.api.types*), 435

## M

**margin** (*nnntool.api.validation.ValidationResultBase property*), 435  
**MaxPoolNode** (*class in nnntool.api.types*), 434  
**mem\_read()** (*gv.gvsoc\_control.Router method*), 334  
**mem\_read()** (*gvsoc\_control.Router method*), 582  
**mem\_read\_int()** (*gv.gvsoc\_control.Router method*), 335

`mem_read_int()` (`gvsoc_control.Router` method), 583  
`mem_write()` (`gv.gvsoc_control.Router` method), 335  
`mem_write()` (`gvsoc_control.Router` method), 583  
`mem_write_int()` (`gv.gvsoc_control.Router` method), 335  
`mem_write_int()` (`gvsoc_control.Router` method), 583  
`model` (`nntool.api.NNGraph` property), 419  
`model_settings()` (in module `nntool.api.utils`), 425  
`module`  
  `gap9.evk`, 392  
  `gap9.gap9_v2`, 392  
  `gapylib.chips.gap.flash`, 392  
  `gapylib.chips.gap.gap9_v2.board_runner`, 395  
  `gapylib.chips.gap.rom_v2`, 393  
  `gapylib.flash`, 377  
  `gapylib.fs.hostfs`, 391  
  `gapylib.fs.littlefs`, 389  
  `gapylib.fs.partition`, 387  
  `gapylib.fs.raw`, 391  
  `gapylib.fs.readfs`, 388  
  `gapylib.target`, 375  
  `gapylib.utils`, 383  
  `gv.gvsoc_control`, 333  
  `gvsoc_control`, 581  
`nntool.api`, 411  
`nntool.api.compression`, 433  
`nntool.api.quantization`, 433  
`nntool.api.types`, 434  
`nntool.api.utils`, 424  
`nntool.api.validation`, 435  
`MultMulBiasScaleQType` (class in `nntool.api.quantization`), 433  
`name` (`gapylib.chips.gap.flash.DefaultFlashRomV2` attribute), 393  
`name` (`gapylib.chips.gap.rom_v2.RomEmptyHeader` attribute), 393  
`name` (`gapylib.chips.gap.rom_v2.RomFlashSection` attribute), 393  
`name` (`gapylib.chips.gap.rom_v2.RomHeader` attribute), 394  
`name` (`gapylib.chips.gap.rom_v2.RomSegment` attribute), 394  
`name` (`gapylib.chips.gap.rom_v2.RomSegmentHeader` attribute), 394  
`name` (`gapylib.flash.Flash` attribute), 377  
`name` (`gapylib.flash.FlashSection` attribute), 379  
`name` (`gapylib.flash.FlashSectionProperty` attribute), 382  
`name` (`gapylib.fs.hostfs.HostfsSection` attribute), 391  
`name` (`gapylib.fs.littlefs.LfsHeader` attribute), 389  
`name` (`gapylib.fs.littlefs.LfsSection` attribute), 390  
`name` (`gapylib.fs.partition.PartitionTableHeader` attribute), 387  
`name` (`gapylib.fs.partition.PartitionTableSection` attribute), 387  
`name` (`gapylib.fs.partition.PartitionTableSectionHeader` attribute), 388  
`name` (`gapylib.fs.raw.RawHeader` attribute), 391  
`name` (`gapylib.fs.raw.RawSection` attribute), 391  
`name` (`gapylib.fs.readfs.ReadfsFile` attribute), 388  
`name` (`gapylib.fs.readfs.ReadfsFileHeader` attribute), 388  
`name` (`gapylib.fs.readfs.ReadfsHeader` attribute), 388  
`name` (`gapylib.fs.readfs.ReadfsSection` attribute), 389  
`name` (`gapylib.utils.CStruct` attribute), 383  
`name` (`gapylib.utils.CStructField` attribute), 385  
`name` (`gapylib.utils.CStructParent` attribute), 385  
`name` (`nntool.api.NNGraph` property), 419  
`name_len` (`gapylib.fs.readfs.ReadfsFileHeader` attribute), 388  
`needs_adjust()` (`nntool.api.NNGraph` method), 419  
`NNGraph` (class in `nntool.api`), 411  
`NNNodeBase` (class in `nntool.api.types`), 434  
`nntool.api`  
  `module`, 411  
`nntool.api.compression`  
  `module`, 433  
`nntool.api.quantization`  
  `module`, 433  
`nntool.api.types`  
  `module`, 434  
`nntool.api.utils`  
  `module`, 424  
`nntool.api.validation`  
  `module`, 435  
`nodes()` (`nntool.api.NNGraph` method), 419  
`nodes_by_step_idx` (`nntool.api.NNGraph` property), 420  
`nodes_by_step_idx_with_fusions`  
  (`nntool.api.NNGraph` property), 420  
`nodes_iterator()` (`nntool.api.NNGraph` method), 420  
`num_constants` (`nntool.api.NNGraph` property), 420  
`num_inputs` (`nntool.api.NNGraph` property), 420  
`num_outputs` (`nntool.api.NNGraph` property), 420

## O

`offset` (`gapylib.utils.CStructField` attribute), 385  
`open()` (`gv.gvsoc_control.Testbench_i2s` method), 336  
`open()` (`gv.gvsoc_control.Testbench_uart` method), 339  
`open()` (`gvsoc_control.Testbench_i2s` method), 584  
`open()` (`gvsoc_control.Testbench_uart` method), 587  
`options` (`gapylib.target.Target` attribute), 375  
`output_nodes()` (`nntool.api.NNGraph` method), 420

## P

`pack()` (`gapylib.utils.CStruct` method), 384

pack() (*gapylib.utils.CStructParent method*), 386  
 parent (*gapylib.chips.gap.rom\_v2.RomEmptyHeader attribute*), 393  
 parent (*gapylib.chips.gap.rom\_v2.RomFlashSection attribute*), 393  
 parent (*gapylib.chips.gap.rom\_v2.RomHeader attribute*), 394  
 parent (*gapylib.chips.gap.rom\_v2.RomSegment attribute*), 394  
 parent (*gapylib.chips.gap.rom\_v2.RomSegmentHeader attribute*), 394  
 parent (*gapylib.flash.FlashSection attribute*), 379  
 parent (*gapylib.fs.hostfs.HostfsSection attribute*), 391  
 parent (*gapylib.fs.littlefs.LfsHeader attribute*), 389  
 parent (*gapylib.fs.littlefs.LfsSection attribute*), 390  
 parent (*gapylib.fs.partition.PartitionTableHeader attribute*), 387  
 parent (*gapylib.fs.partition.PartitionTableSection attribute*), 387  
 parent (*gapylib.fs.partition.PartitionTableSectionHeader attribute*), 388  
 parent (*gapylib.fs.raw.RawHeader attribute*), 391  
 parent (*gapylib.fs.raw.RawSection attribute*), 391  
 parent (*gapylib.fs.readfs.ReadfsFile attribute*), 388  
 parent (*gapylib.fs.readfs.ReadfsFileHeader attribute*), 388  
 parent (*gapylib.fs.readfs.ReadfsHeader attribute*), 389  
 parent (*gapylib.fs.readfs.ReadfsSection attribute*), 389  
 parent (*gapylib.utils.CStruct attribute*), 383  
 parent (*gapylib.utils.CStructParent attribute*), 385  
 parse\_args() (*gapylib.chips.gap.gap9\_v2.board\_runner.Rpi\_apis method*), 395  
 parse\_args() (*gapylib.target.Target method*), 376  
 PartitionTableHeader (*class in gapylib.fs.partition*), 387  
 PartitionTableSection (*class in gapylib.fs.partition*), 387  
 PartitionTableSectionHeader (*class in gapylib.fs.partition*), 387  
 pi\_ads1014\_close (*C function*), 591  
 pi\_ads1014\_conf (*C struct*), 592  
 pi\_ads1014\_conf.comparator\_latch (*C var*), 592  
 pi\_ads1014\_conf.comparator\_mode (*C var*), 592  
 pi\_ads1014\_conf.comparator\_polarity (*C var*), 592  
 pi\_ads1014\_conf.comparator\_status (*C var*), 592  
 pi\_ads1014\_conf.data\_rate (*C var*), 592  
 pi\_ads1014\_conf.i2c\_addr (*C var*), 592  
 pi\_ads1014\_conf.i2c\_itf (*C var*), 592  
 pi\_ads1014\_conf.operating\_mode (*C var*), 592  
 pi\_ads1014\_conf.pga (*C var*), 592  
 pi\_ads1014\_conf\_init (*C function*), 591  
 pi\_ads1014\_open (*C function*), 591  
 pi\_ads1014\_read (*C function*), 591  
 pi\_ads1014\_set\_comparator\_thresholds (*C function*), 591  
 pi\_aes\_close (*C function*), 450  
 pi\_aes\_conf (*C struct*), 452  
 pi\_aes\_conf.itf (*C var*), 453  
 pi\_aes\_conf.iv (*C var*), 453  
 pi\_aes\_conf.key (*C var*), 453  
 pi\_aes\_conf.key\_len (*C var*), 453  
 pi\_aes\_conf.mode (*C var*), 453  
 pi\_aes\_conf\_init (*C function*), 450  
 pi\_aes\_conf\_t (*C type*), 449  
 pi\_aes\_ctr\_set (*C function*), 452  
 pi\_aes\_decrypt (*C function*), 451  
 pi\_aes\_decrypt\_async (*C function*), 452  
 pi\_aes\_encrypt (*C function*), 451  
 pi\_aes\_encrypt\_async (*C function*), 451  
 pi\_aes\_ioctl (*C function*), 450  
 pi\_aes\_ioctl\_e (*C enum*), 450  
 pi\_aes\_ioctl\_e.PI\_AES\_IOCTL\_STOP\_FIFO\_MODE (*C enumerator*), 450  
 pi\_aes\_key\_len\_e (*C enum*), 449  
 pi\_aes\_key\_len\_e.PI\_AES\_KEY\_128 (*C enumerator*), 450  
 pi\_aes\_key\_len\_e.PI\_AES\_KEY\_256 (*C enumerator*), 450  
 pi\_aes\_mode\_e (*C enum*), 449  
 pi\_aes\_mode\_e.PI\_AES\_MODE\_CBC (*C enumerator*), 449  
 pi\_aes\_mode\_e.PI\_AES\_MODE\_CTR (*C enumerator*), 449  
 pi\_aes\_mode\_e.PI\_AES\_MODE\_ECB (*C enumerator*), 449  
 pi\_aes\_open (*C function*), 450  
 pi\_ak4332\_api (*C struct*), 709  
 pi\_ak4332\_api.disable (*C var*), 710  
 pi\_ak4332\_api.enable (*C var*), 710  
 pi\_ak4332\_api.init (*C var*), 710  
 pi\_ak4332\_api.set\_volume (*C var*), 710  
 pi\_ak4332\_api.start (*C var*), 710  
 pi\_ak4332\_api.stop (*C var*), 710  
 pi\_ak4332\_api\_t (*C type*), 692  
 pi\_ak4332\_close (*C function*), 706  
 pi\_ak4332\_conf (*C struct*), 710  
 pi\_ak4332\_conf.api (*C var*), 710  
 pi\_ak4332\_conf.i2c\_itf (*C var*), 710  
 pi\_ak4332\_conf.i2s\_itf (*C var*), 710  
 pi\_ak4332\_conf.pdn (*C var*), 710  
 pi\_ak4332\_conf\_t (*C type*), 692  
 pi\_ak4332\_disable (*C function*), 705  
 pi\_ak4332\_enable (*C function*), 705  
 pi\_ak4332\_init (*C function*), 704  
 pi\_ak4332\_init\_std (*C function*), 702  
 pi\_ak4332\_is\_pdm\_fullscale (*C function*), 703  
 pi\_ak4332\_load\_init\_preset (*C function*), 702

pi\_ak4332\_open (*C function*), 705  
 pi\_ak4332\_set\_volume (*C function*), 705  
 pi\_ak4332\_set\_volume\_std (*C function*), 703  
 pi\_ak4332\_start (*C function*), 704  
 pi\_ak4332\_start\_std (*C function*), 702  
 pi\_ak4332\_stop (*C function*), 704  
 pi\_ak4332\_stop\_std (*C function*), 703  
 pi\_alloc\_fail (*C function*), 444  
 PI\_AT\_RESP\_ARRAY\_LENGTH (*C macro*), 596  
 pi\_audio\_buffer\_data (*C function*), 266  
 pi\_audio\_buffer\_init (*C function*), 265  
 pi\_audio\_close (*C function*), 264  
 pi\_audio\_conf (*C struct*), 267  
 pi\_audio\_conf.fs (*C var*), 268  
 pi\_audio\_conf\_init (*C function*), 264  
 pi\_audio\_conf\_t (*C type*), 256  
 pi\_audio\_configure\_node (*C function*), 266  
 pi\_audio\_graph\_close (*C function*), 264  
 pi\_audio\_graph\_get\_input\_by\_name (*C function*),  
     266  
 pi\_audio\_graph\_get\_output\_by\_name (*C function*),  
     266  
 pi\_audio\_graph\_open (*C function*), 264  
 pi\_audio\_graph\_reconfigure\_async (*C function*),  
     267  
 pi\_audio\_graph\_start (*C function*), 265  
 pi\_audio\_graph\_stop (*C function*), 265  
 pi\_audio\_input\_enqueue (*C function*), 265  
 pi\_audio\_open (*C function*), 264  
 pi\_audio\_output\_enqueue (*C function*), 266  
 pi\_audio\_set\_volume (*C function*), 267  
 pi\_ble\_api\_s (*C struct*), 596  
 pi\_ble\_api\_s.at\_cmd (*C var*), 596  
 pi\_ble\_api\_s.peer\_connect (*C var*), 596  
 pi\_ble\_api\_s.peer\_disconnect (*C var*), 596  
 pi\_ble\_api\_t (*C type*), 593  
 pi\_ble\_at\_cmd (*C function*), 593  
 pi\_ble\_catch\_peer\_event\_async (*C function*), 595  
 pi\_ble\_close (*C function*), 593  
 pi\_ble\_data\_get (*C function*), 595  
 pi\_ble\_data\_get\_async (*C function*), 595  
 pi\_ble\_data\_send (*C function*), 594  
 pi\_ble\_data\_send\_async (*C function*), 594  
 pi\_ble\_ioctl (*C function*), 593  
 pi\_ble\_ioctl\_cmd\_e (*C enum*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_CLIENT\_CONFIGURE  
     (*C enumerator*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_DATA\_MODE\_ENTER  
     (*C enumerator*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_DATA\_MODE\_EXIT  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_MODEL\_INFO  
     (*C enumerator*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_SERVER\_CONFIGURE  
     (*C enumerator*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_UART\_CONFIGURE  
     (*C enumerator*), 596  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B112\_WAIT\_FOR\_EVENT  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_CATCH\_PEER\_EVENT  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_CLIENT\_CONFIGURE  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_DATA\_MODE\_ENTER  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_DATA\_MODE\_EXIT  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_MODEL\_INFO  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_SERVER\_CONFIGURE  
     (*C enumerator*), 597  
 pi\_ble\_ioctl\_cmd\_e.PI\_NINA\_B312\_UART\_CONFIGURE  
     (*C enumerator*), 597  
 pi\_ble\_nina\_b312\_conf\_init (*C function*), 597  
 pi\_ble\_open (*C function*), 593  
 pi\_ble\_peer\_connect (*C function*), 594  
 pi\_ble\_peer\_disconnect (*C function*), 594  
 pi\_callback\_init (*C function*), 443  
 pi\_camera\_capture (*C function*), 600  
 pi\_camera\_capture\_async (*C function*), 601  
 pi\_camera\_close (*C function*), 601  
 pi\_camera\_cmd\_e (*C enum*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_AEG\_INIT  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_CONTINUE\_MODE  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_OFF  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_ON  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_POWERDOWN\_MODE  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_PWM\_CLK  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_SNAPSHOT  
     (*C enumerator*), 599  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_START  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_STOP  
     (*C enumerator*), 598  
 pi\_camera\_cmd\_e.PI\_CAMERA\_CMD\_TRIGGER\_MODE  
     (*C enumerator*), 598  
 pi\_camera\_color\_mode\_e (*C enum*), 599  
 pi\_camera\_color\_mode\_e.PI\_CAMERA\_GRAY8  
     (*C enumerator*), 599  
 pi\_camera\_color\_mode\_e.PI\_CAMERA\_RGB565  
     (*C enumerator*), 600

pi\_camera\_color\_mode\_e.PI\_CAMERA\_RGB888 (C enumerator), 600  
 pi\_camera\_color\_mode\_e.PI\_CAMERA\_YUV (C enumerator), 600  
 pi\_camera\_control (C function), 600  
 pi\_camera\_format\_e (C enum), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_HD1080 (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_HD720 (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_QQVGA (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_QVGA (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_QWXGA (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_VGA (C enumerator), 599  
 pi\_camera\_format\_e.PI\_CAMERA\_WXGA (C enumerator), 599  
 pi\_camera\_open (C function), 600  
 pi\_camera\_opts\_e (C enum), 599  
 pi\_camera\_opts\_e.PI\_CAMERA\_NO\_OPT (C enumerator), 599  
 pi\_camera\_opts\_e.PI\_CAMERA\_OPT\_NO\_REG\_INIT (C enumerator), 599  
 pi\_camera\_reg\_get (C function), 601  
 pi\_camera\_reg\_set (C function), 601  
 pi\_camera\_slicing\_conf (C struct), 602  
 pi\_camera\_slicing\_conf\_t (C struct), 602  
 pi\_camera\_slicing\_conf\_t.h (C var), 602  
 pi\_camera\_slicing\_conf\_t.slice\_en (C var), 602  
 pi\_camera\_slicing\_conf\_t.w (C var), 602  
 pi\_camera\_slicing\_conf\_t.x (C var), 602  
 pi\_camera\_slicing\_conf\_t.y (C var), 602  
 pi\_cl\_cluster\_nb\_cores (C function), 458  
 pi\_cl\_dma\_cmd (C function), 462  
 pi\_cl\_dma\_cmd\_2d (C function), 463  
 pi\_cl\_dma\_cmd\_t (C type), 462  
 pi\_cl\_dma\_cmd\_wait (C function), 463  
 pi\_cl\_dma\_copy\_2d\_t (C type), 462  
 pi\_cl\_dma\_copy\_t (C type), 462  
 pi\_cl\_dma\_dir\_e (C enum), 462  
 pi\_cl\_dma\_dir\_e.PI\_CL\_DMA\_DIR\_EXT2LOC (C enumerator), 462  
 pi\_cl\_dma\_dir\_e.PI\_CL\_DMA\_DIR\_LOC2EXT (C enumerator), 462  
 pi\_cl\_dma\_flush (C function), 463  
 pi\_cl\_dma\_memcpy (C function), 463  
 pi\_cl\_dma\_memcpy\_2d (C function), 463  
 pi\_cl\_dma\_wait (C function), 464  
 pi\_cl\_fs\_copy (C function), 625  
 pi\_cl\_fs\_copy\_2d (C function), 625  
 pi\_cl\_fs\_direct\_read (C function), 624  
 pi\_cl\_fs\_read (C function), 623  
 pi\_cl\_fs\_req\_t (C type), 615  
 pi\_cl\_fs\_seek (C function), 624  
 pi\_cl\_fs\_seek\_read (C function), 625  
 pi\_cl\_fs\_wait (C function), 626  
 pi\_cl\_fs\_write (C function), 624  
 pi\_cl\_hyper\_copy (C function), 491  
 pi\_cl\_hyper\_copy\_2d (C function), 491  
 pi\_cl\_hyper\_read (C function), 489  
 pi\_cl\_hyper\_read\_2d (C function), 490  
 pi\_cl\_hyper\_read\_wait (C function), 490  
 pi\_cl\_hyper\_req\_t (C type), 484  
 pi\_cl\_hyper\_write (C function), 490  
 pi\_cl\_hyper\_write\_2d (C function), 491  
 pi\_cl\_hyper\_write\_wait (C function), 491  
 pi\_cl\_ll\_available\_get (C function), 446  
 pi\_cl\_ll\_free (C function), 446  
 pi\_cl\_ll\_malloc (C function), 446  
 pi\_cl\_ll\_malloc\_align (C function), 447  
 pi\_cl\_ll\_malloc\_dump (C function), 447  
 pi\_cl\_ll\_scratch\_alloc (C function), 457  
 pi\_cl\_ll\_scratch\_free (C function), 457  
 pi\_cl\_octospi\_copy (C function), 531  
 pi\_cl\_octospi\_copy\_2d (C function), 532  
 pi\_cl\_octospi\_read (C function), 529  
 pi\_cl\_octospi\_read\_2d (C function), 530  
 pi\_cl\_octospi\_read\_wait (C function), 530  
 pi\_cl\_octospi\_req\_t (C type), 523  
 pi\_cl\_octospi\_write (C function), 530  
 pi\_cl\_octospi\_write\_2d (C function), 531  
 pi\_cl\_octospi\_write\_wait (C function), 531  
 pi\_cl\_ram\_alloc (C function), 636  
 pi\_cl\_ram\_alloc\_req\_t (C type), 630  
 pi\_cl\_ram\_alloc\_wait (C function), 637  
 pi\_cl\_ram\_copy (C function), 638  
 pi\_cl\_ram\_copy\_2d (C function), 639  
 pi\_cl\_ram\_copy\_wait (C function), 640  
 pi\_cl\_ram\_free (C function), 636  
 pi\_cl\_ram\_free\_req\_t (C type), 630  
 pi\_cl\_ram\_free\_wait (C function), 637  
 pi\_cl\_ram\_read (C function), 637  
 pi\_cl\_ram\_read\_2d (C function), 638  
 pi\_cl\_ram\_read\_wait (C function), 639  
 pi\_cl\_ram\_req\_t (C type), 630  
 pi\_cl\_ram\_write (C function), 637  
 pi\_cl\_ram\_write\_2d (C function), 638  
 pi\_cl\_ram\_write\_wait (C function), 640  
 pi\_cl\_send\_callback\_to\_fc (C function), 454  
 pi\_cl\_send\_task\_to\_fc (C function), 454  
 pi\_cl\_task\_yield (C function), 457  
 pi\_cl\_team\_barrier (C function), 461  
 pi\_cl\_team\_barrier\_alloc (C function), 460  
 pi\_cl\_team\_barrier\_free (C function), 460  
 pi\_cl\_team\_barrier\_id (C function), 460

pi\_cl\_team\_barrier\_nb\_available (*C function*), 460  
pi\_cl\_team\_barrier\_set (*C function*), 460  
pi\_cl\_team\_barrier\_wait (*C function*), 460  
pi\_cl\_team\_critical\_enter (*C function*), 461  
pi\_cl\_team\_critical\_exit (*C function*), 461  
pi\_cl\_team\_fork (*C function*), 458  
pi\_cl\_team\_fork\_task (*C function*), 459  
pi\_cl\_team\_nb\_cores (*C function*), 458  
pi\_cl\_team\_prepare\_fork (*C function*), 459  
pi\_cl\_team\_preset\_fork (*C function*), 459  
pi\_cl\_uart\_read (*C function*), 563  
pi\_cl\_uart\_read\_byte (*C function*), 563  
pi\_cl\_uart\_read\_wait (*C function*), 563  
pi\_cl\_uart\_req\_t (*C type*), 553  
pi\_cl\_uart\_write (*C function*), 562  
pi\_cl\_uart\_write\_byte (*C function*), 562  
pi\_cl\_uart\_write\_wait (*C function*), 563  
pi\_cluster\_close (*C function*), 454  
pi\_cluster\_conf (*C struct*), 458  
pi\_cluster\_conf.cc\_stack\_size (*C var*), 458  
pi\_cluster\_conf.device\_type (*C var*), 458  
pi\_cluster\_conf.icache\_conf (*C var*), 458  
pi\_cluster\_conf.id (*C var*), 458  
pi\_cluster\_conf.scratch\_size (*C var*), 458  
pi\_cluster\_conf\_init (*C function*), 454  
pi\_cluster\_enqueue\_task (*C function*), 456  
pi\_cluster\_enqueue\_task\_async (*C function*), 456  
pi\_cluster\_flags\_e (*C enum*), 453  
pi\_cluster\_flags\_e.PI\_CLUSTER\_FLAGS\_FORK\_BASED  
    (*C enumerator*), 453  
pi\_cluster\_flags\_e.PI\_CLUSTER\_FLAGS\_TASK\_BASED  
    (*C enumerator*), 453  
pi\_cluster\_open (*C function*), 454  
pi\_cluster\_send\_task (*C function*), 455  
pi\_cluster\_send\_task\_async (*C function*), 456  
pi\_cluster\_task (*C function*), 454  
pi\_cluster\_task\_priority (*C function*), 455  
pi\_cluster\_task\_stacks (*C function*), 455  
pi\_cpi\_capture (*C function*), 465  
pi\_cpi\_capture\_async (*C function*), 465  
pi\_cpi\_close (*C function*), 465  
pi\_cpi\_conf (*C struct*), 467  
pi\_cpi\_conf.datasize (*C var*), 468  
pi\_cpi\_conf.device (*C var*), 468  
pi\_cpi\_conf.itf (*C var*), 468  
pi\_cpi\_conf\_init (*C function*), 465  
pi\_cpi\_control\_start (*C function*), 466  
pi\_cpi\_control\_stop (*C function*), 466  
pi\_cpi\_format\_e (*C enum*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_BYPASS\_BIGEND  
    (*C enumerator*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_BYPASS\_LITEND  
    (*C enumerator*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_RGB444 (*C enum*  
    *enumerator*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_RGB555 (*C enum*  
    *enumerator*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_RGB565 (*C enum*  
    *enumerator*), 464  
pi\_cpi\_format\_e.PI\_CPI\_FORMAT\_RGB888 (*C enum*  
    *enumerator*), 464  
pi\_cpi\_open (*C function*), 465  
pi\_cpi\_set\_filter (*C function*), 466  
pi\_cpi\_set\_format (*C function*), 466  
pi\_cpi\_set\_frame\_drop (*C function*), 466  
pi\_cpi\_set\_rgb\_sequence (*C function*), 467  
pi\_cpi\_set\_rowlen (*C function*), 466  
pi\_cpi\_set\_slice (*C function*), 467  
pi\_cpi\_set\_sync\_polarity (*C function*), 467  
pi\_crc32\_compute (*C function*), 607  
pi\_csi2\_capture (*C function*), 469  
pi\_csi2\_capture\_async (*C function*), 469  
pi\_csi2\_close (*C function*), 469  
pi\_csi2\_conf (*C struct*), 470  
pi\_csi2\_conf.datasize (*C var*), 471  
pi\_csi2\_conf.device (*C var*), 471  
pi\_csi2\_conf.itf (*C var*), 471  
pi\_csi2\_conf\_init (*C function*), 468  
pi\_csi2\_control\_start (*C function*), 469  
pi\_csi2\_control\_stop (*C function*), 469  
pi\_csi2\_format\_e (*C enum*), 468  
pi\_csi2\_format\_e.PI\_CSI2\_FORMAT\_RAW10 (*C enum*  
    *enumerator*), 468  
pi\_csi2\_format\_e.PI\_CSI2\_FORMAT\_RAW8 (*C enum*  
    *enumerator*), 468  
pi\_csi2\_open (*C function*), 468  
pi\_csi2\_set\_apb\_clk\_div (*C function*), 470  
pi\_csi2\_set\_lane (*C function*), 470  
pi\_csi2\_set\_pixel\_clk\_div (*C function*), 470  
pi\_csi2\_set\_virtual\_channel (*C function*), 470  
pi\_da16200\_conf (*C struct*), 715  
pi\_da16200\_conf.stop\_bits (*C var*), 716  
pi\_da16200\_conf.uart\_baudrate (*C var*), 716  
pi\_da16200\_conf.uart\_itf (*C var*), 716  
pi\_da16200\_conf.uart\_parity\_bits (*C var*), 716  
pi\_da16200\_conf.use\_ctrl\_flow (*C var*), 716  
pi\_display\_ioctl (*C function*), 608  
pi\_display\_ioctl\_cmd\_e (*C enum*), 607  
pi\_display\_ioctl\_cmd\_e.PI\_DISPLAY\_IOCTL\_CUSTOM  
    (*C enumerator*), 607  
pi\_display\_open (*C function*), 607  
pi\_display\_write (*C function*), 607  
pi\_display\_write\_async (*C function*), 608  
pi\_dmacpy\_close (*C function*), 472  
pi\_dmacpy\_conf (*C struct*), 473  
pi\_dmacpy\_conf.id (*C var*), 473  
pi\_dmacpy\_conf\_init (*C function*), 471

pi\_dmacpy\_copy (*C function*), 472  
 pi\_dmacpy\_copy\_async (*C function*), 472  
 pi\_dmacpy\_dir\_e (*C enum*), 471  
 pi\_dmacpy\_dir\_e.PI\_DMACPY\_FC\_L1\_L2 (*C enumerator*), 471  
 pi\_dmacpy\_dir\_e.PI\_DMACPY\_L2\_FC\_L1 (*C enumerator*), 471  
 pi\_dmacpy\_dir\_e.PI\_DMACPY\_L2\_L2 (*C enumerator*), 471  
 pi\_dmacpy\_open (*C function*), 471  
 pi\_efuse\_ioctl (*C function*), 474  
 pi\_efuse\_ioctl\_e (*C enum*), 473  
 pi\_efuse\_ioctl\_e.PI\_EFUSE\_IOCTL\_CLOSE (*C enumerator*), 474  
 pi\_efuse\_ioctl\_e.PI\_EFUSE\_IOCTL\_PROGRAM\_START (*C enumerator*), 473  
 pi\_efuse\_ioctl\_e.PI\_EFUSE\_IOCTL\_READ\_START (*C enumerator*), 473  
 pi\_efuse\_program (*C function*), 474  
 pi\_efuse\_value\_get (*C function*), 474  
 pi\_err\_t (*C enum*), 448  
 pi\_err\_t.PI\_ERR\_ALREADY\_EXISTS (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_I2C\_NACK (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_INVALID\_APP (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_INVALID\_ARG (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_INVALID\_CRC (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_INVALID\_MAGIC\_CODE (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_INVALID\_SIZE (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_INVALID\_STATE (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_INVALID\_VERSION (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_L2\_NO\_MEM (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_NO\_MEM (*C enumerator*), 449  
 pi\_err\_t.PI\_ERR\_NOT\_FOUND (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_NOT\_SUPPORTED (*C enumerator*), 448  
 pi\_err\_t.PI\_ERR\_TIMEOUT (*C enumerator*), 448  
 pi\_err\_t.PI\_FAIL (*C enumerator*), 448  
 pi\_err\_t.PI\_OK (*C enumerator*), 448  
 pi\_evt\_callback\_irq\_init (*C function*), 441  
 pi\_evt\_callback\_no\_irq\_init (*C function*), 441  
 pi\_evt\_cancel\_delayed\_us (*C function*), 443  
 pi\_evt\_push (*C function*), 442  
 pi\_evt\_push\_delayed\_us (*C function*), 442  
 pi\_evt\_push\_from\_irq (*C function*), 442  
 pi\_evt\_sig\_init (*C function*), 441  
 pi\_evt\_status\_get (*C function*), 443  
 pi\_evt\_status\_set (*C function*), 443  
 pi\_evt\_timeout\_set (*C function*), 443  
 pi\_evt\_wait (*C function*), 442  
 pi\_evt\_wait\_on (*C macro*), 440  
 pi\_fc\_l1\_available\_get (*C function*), 446  
 pi\_fc\_l1\_free (*C function*), 447  
 pi\_fc\_l1\_malloc (*C function*), 447  
 pi\_fc\_l1\_malloc\_align (*C function*), 447  
 pi\_fc\_l1\_malloc\_dump (*C function*), 448  
 pi\_ffc\_close (*C function*), 476  
 pi\_ffc\_conf (*C struct*), 477  
 pi\_ffc\_conf.fixed\_precision (*C var*), 477  
 pi\_ffc\_conf.fixed\_scale (*C var*), 477  
 pi\_ffc\_conf.fixed\_type (*C var*), 477  
 pi\_ffc\_conf.float\_type (*C var*), 477  
 pi\_ffc\_conf.io\_mode (*C var*), 477  
 pi\_ffc\_conf.itf (*C var*), 477  
 pi\_ffc\_conf.mode (*C var*), 477  
 pi\_ffc\_conf\_init (*C function*), 476  
 pi\_ffc\_conf\_t (*C type*), 475  
 pi\_ffc\_convert (*C function*), 477  
 pi\_ffc\_convert\_async (*C function*), 477  
 pi\_ffc\_fixed\_type\_e (*C enum*), 475  
 pi\_ffc\_fixed\_type\_e.PI\_FFC\_FIXED\_16 (*C enumerator*), 475  
 pi\_ffc\_fixed\_type\_e.PI\_FFC\_FIXED\_24 (*C enumerator*), 475  
 pi\_ffc\_fixed\_type\_e.PI\_FFC\_FIXED\_32 (*C enumerator*), 475  
 pi\_ffc\_fixed\_type\_e.PI\_FFC\_FIXED\_8 (*C enumerator*), 475  
 pi\_ffc\_float\_type\_e (*C enum*), 475  
 pi\_ffc\_float\_type\_e.PI\_FFC\_FLOAT\_BFP16 (*C enumerator*), 475  
 pi\_ffc\_float\_type\_e.PI\_FFC\_FLOAT\_FP16 (*C enumerator*), 475  
 pi\_ffc\_float\_type\_e.PI\_FFC\_FLOAT\_FP32 (*C enumerator*), 475  
 pi\_ffc\_io\_mode\_e (*C enum*), 475  
 pi\_ffc\_io\_mode\_e.PI\_FFC\_MEMORY\_IN\_MEMORY\_OUT (*C enumerator*), 475  
 pi\_ffc\_io\_mode\_e.PI\_FFC\_MEMORY\_IN\_STREAM\_OUT (*C enumerator*), 475  
 pi\_ffc\_io\_mode\_e.PI\_FFC\_STREAM\_IN\_MEMORY\_OUT (*C enumerator*), 475  
 pi\_ffc\_io\_mode\_e.PI\_FFC\_STREAM\_IN\_STREAM\_OUT (*C enumerator*), 475  
 pi\_ffc\_ioctl (*C function*), 476  
 pi\_ffc\_ioctl\_e (*C enum*), 475  
 pi\_ffc\_ioctl\_e.PI\_FFC\_IOCTL\_CONTINUOUS\_ENABLE (*C enumerator*), 476  
 pi\_ffc\_ioctl\_e.PI\_FFC\_IOCTL\_SET\_IO\_MODE (*C enumerator*), 475  
 pi\_ffc\_mode\_e (*C enum*), 475  
 pi\_ffc\_mode\_e.PI\_FFC\_FIXED\_TO\_FLOAT (*C enumerator*), 475  
 pi\_ffc\_mode\_e.PI\_FFC\_FLOAT\_TO\_FIXED (*C enumerator*), 475

pi\_ffc\_open (*C function*), 476  
 pi\_flash\_close (*C function*), 609  
 pi\_flash\_conf (*C struct*), 613  
 pi\_flash\_conf.api (*C var*), 614  
 pi\_flash\_erase (*C function*), 612  
 pi\_flash\_erase\_async (*C function*), 613  
 pi\_flash\_erase\_chip (*C function*), 611  
 pi\_flash\_erase\_chip\_async (*C function*), 613  
 pi\_flash\_erase\_sector (*C function*), 611  
 pi\_flash\_erase\_sector\_async (*C function*), 613  
 pi\_flash\_info (*C struct*), 614  
 pi\_flash\_info.flash\_start (*C var*), 614  
 pi\_flash\_info.sector\_size (*C var*), 614  
 pi\_flash\_ioctl (*C function*), 609  
 pi\_flash\_ioctl\_e (*C enum*), 609  
 pi\_flash\_ioctl\_e.PI\_FLASH\_IOCTL\_AES\_ENABLE  
     (*C enumerator*), 609  
 pi\_flash\_ioctl\_e.PI\_FLASH\_IOCTL\_INFO (*C enumerator*), 609  
 pi\_flash\_ioctl\_e.PI\_FLASH\_IOCTL\_SET\_BAUDRATE  
     (*C enumerator*), 609  
 pi\_flash\_ioctl\_e.PI\_FLASH\_IOCTL\_SET\_MBA (*C enumerator*), 609  
 pi\_flash\_open (*C function*), 609  
 pi\_flash\_program (*C function*), 611  
 pi\_flash\_program\_async (*C function*), 612  
 pi\_flash\_read (*C function*), 611  
 pi\_flash\_read\_async (*C function*), 612  
 pi\_flash\_reg\_get (*C function*), 610  
 pi\_flash\_reg\_get\_async (*C function*), 610  
 pi\_flash\_reg\_set (*C function*), 610  
 pi\_flash\_reg\_set\_async (*C function*), 610  
 pi\_free (*C function*), 445  
 pi\_fs\_close (*C function*), 619  
 pi\_fs\_conf (*C struct*), 626  
 pi\_fs\_conf.api (*C var*), 626  
 pi\_fs\_conf.auto\_format (*C var*), 626  
 pi\_fs\_conf.flash (*C var*), 626  
 pi\_fs\_conf.partition\_name (*C var*), 626  
 pi\_fs\_conf.type (*C var*), 626  
 pi\_fs\_conf\_init (*C function*), 617  
 pi\_fs\_copy (*C function*), 620  
 pi\_fs\_copy\_2d (*C function*), 621  
 pi\_fs\_copy\_2d\_async (*C function*), 623  
 pi\_fs\_copy\_async (*C function*), 623  
 pi\_fs\_dir\_close (*C function*), 618  
 pi\_fs\_dir\_open (*C function*), 618  
 pi\_fs\_dir\_read (*C function*), 618  
 pi\_fs\_dir\_t (*C type*), 615  
 pi\_fs\_direct\_read (*C function*), 619  
 pi\_fs\_direct\_read\_async (*C function*), 622  
 pi\_fs\_error\_e (*C enum*), 616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_BADF (*C enumerator*), 617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_CORRUPT (*C enumerator*),  
     616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_EXIST (*C enumerator*),  
     616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_FBIG (*C enumerator*), 617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_INVAL (*C enumerator*),  
     617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_IO (*C enumerator*), 616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_ISDIR (*C enumerator*),  
     616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NAMETOOLONG (*C enumerator*), 617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOATTR (*C enumerator*),  
     617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOENT (*C enumerator*),  
     616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOMEM (*C enumerator*),  
     617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOSPC (*C enumerator*),  
     617  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOTDIR (*C enumerator*),  
     616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_NOTEEMPTY (*C enumerator*), 616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_OK (*C enumerator*), 616  
 pi\_fs\_error\_e.PI\_FS\_ERR\_UNSUPPORTED (*C enumerator*), 617  
 pi\_fs\_error\_e.PI\_FS\_MOUNT\_FLASH\_ERROR (*C enumerator*), 617  
 pi\_fs\_error\_e.PI\_FS\_MOUNT\_MEM\_ERROR (*C enumerator*), 617  
 pi\_fs\_file\_t (*C type*), 615  
 pi\_fs\_file\_type\_e (*C enum*), 616  
 pi\_fs\_file\_type\_e.PI\_FS\_TYPE\_DIR (*C enumerator*), 616  
 pi\_fs\_file\_type\_e.PI\_FS\_TYPE\_REG (*C enumerator*), 616  
 pi\_fs\_flags\_e (*C enum*), 616  
 pi\_fs\_flags\_e.PI\_FS\_FLAGS\_APPEND (*C enumerator*), 616  
 pi\_fs\_flags\_e.PI\_FS\_FLAGS\_READ (*C enumerator*),  
     616  
 pi\_fs\_flags\_e.PI\_FS\_FLAGS\_WRITE (*C enumerator*),  
     616  
 pi\_fs\_info\_s (*C struct*), 626  
 pi\_fs\_info\_t (*C type*), 615  
 pi\_fs\_ls (*C function*), 618  
 pi\_fs\_mkdir (*C function*), 617  
 pi\_fs\_mount (*C function*), 617  
 pi\_fs\_open (*C function*), 618  
 pi\_fs\_read (*C function*), 619  
 pi\_fs\_read\_async (*C function*), 621  
 pi\_fs\_remove (*C function*), 618  
 pi\_fs\_seek (*C function*), 620

pi\_fs\_seek\_read (*C function*), 620  
 pi\_fs\_seek\_read\_async (*C function*), 621  
 pi\_fs\_type\_e (*C enum*), 616  
 pi\_fs\_type\_e.PI\_FS\_HOST (*C enumerator*), 616  
 pi\_fs\_type\_e.PI\_FS\_LFS (*C enumerator*), 616  
 pi\_fs\_type\_e.PI\_FS\_READ\_ONLY (*C enumerator*), 616  
 pi\_fs\_unmount (*C function*), 617  
 pi\_fs\_write (*C function*), 619  
 pi\_fs\_write\_async (*C function*), 622  
 pi\_fuser\_reg (*C struct*), 474  
 pi\_fuser\_reg.id (*C var*), 474  
 pi\_fuser\_reg.val (*C var*), 474  
 pi\_fuser\_reg\_t (*C type*), 473  
 pi\_fx16408\_close (*C function*), 628  
 pi\_fx16408\_conf (*C struct*), 629  
 pi\_fx16408\_conf.i2c\_itf (*C var*), 629  
 pi\_fx16408\_conf.interrupt\_pin (*C var*), 629  
 pi\_fx16408\_conf\_init (*C function*), 628  
 pi\_fx16408\_gpio\_conf\_init (*C function*), 628  
 pi\_fx16408\_gpio\_conf\_t (*C struct*), 629  
 pi\_fx16408\_gpio\_conf\_t.direction (*C var*), 630  
 pi\_fx16408\_gpio\_conf\_t.id (*C var*), 630  
 pi\_fx16408\_gpio\_conf\_t.input\_trigger (*C var*),  
     630  
 pi\_fx16408\_gpio\_conf\_t.irq\_task (*C var*), 630  
 pi\_fx16408\_gpio\_conf\_t.output\_state (*C var*),  
     630  
 pi\_fx16408\_gpio\_conf\_t.pull\_state (*C var*), 630  
 pi\_fx16408\_gpio\_set (*C function*), 628  
 pi\_fx16408\_input\_status\_get (*C function*), 629  
 pi\_fx16408\_interrupt\_status\_get (*C function*),  
     629  
 pi\_fx16408\_open (*C function*), 628  
 pi\_gpio\_callback\_add (*C function*), 482  
 pi\_gpio\_callback\_init (*C function*), 482  
 pi\_gpio\_callback\_remove (*C function*), 483  
 pi\_gpio\_callback\_s (*C struct*), 483  
 pi\_gpio\_callback\_s.args (*C var*), 484  
 pi\_gpio\_callback\_s.handler (*C var*), 484  
 pi\_gpio\_callback\_s.next (*C var*), 484  
 pi\_gpio\_callback\_s.pin\_mask (*C var*), 484  
 pi\_gpio\_callback\_s.prev (*C var*), 484  
 pi\_gpio\_callback\_t (*C type*), 478  
 pi\_gpio\_close (*C function*), 479  
 pi\_gpio\_conf (*C struct*), 483  
 pi\_gpio\_conf.device (*C var*), 483  
 pi\_gpio\_conf.port (*C var*), 483  
 pi\_gpio\_conf\_init (*C function*), 479  
 pi\_gpio\_flags\_e (*C enum*), 478  
 pi\_gpio\_flags\_e.PI\_GPIO\_DRIVE\_STRENGTH\_HIGH  
     (*C enumerator*), 478  
 pi\_gpio\_flags\_e.PI\_GPIO\_DRIVE\_STRENGTH\_LOW  
     (*C enumerator*), 478  
 pi\_gpio\_flags\_e.PI\_GPIO\_INPUT (*C enumerator*),  
     478  
 pi\_gpio\_flags\_e.PI\_GPIO\_OUTPUT (*C enumerator*),  
     478  
 pi\_gpio\_flags\_e.PI\_GPIO\_PULL\_DISABLE (*C enumerator*), 478  
 pi\_gpio\_flags\_e.PI\_GPIO\_PULL\_ENABLE (*C enumerator*), 478  
 pi\_gpio\_notif\_e (*C enum*), 478  
 pi\_gpio\_notif\_e.PI\_GPIO\_NOTIF\_EDGE (*C enumerator*), 478  
 pi\_gpio\_notif\_e.PI\_GPIO\_NOTIF\_FALL (*C enumerator*), 478  
 pi\_gpio\_notif\_e.PI\_GPIO\_NOTIF\_NONE (*C enumerator*), 478  
 pi\_gpio\_notif\_e.PI\_GPIO\_NOTIF\_RISE (*C enumerator*), 478  
 pi\_gpio\_open (*C function*), 479  
 pi\_gpio\_pin\_configure (*C function*), 479  
 pi\_gpio\_pin\_notif\_clear (*C function*), 481  
 pi\_gpio\_pin\_notif\_configure (*C function*), 481  
 pi\_gpio\_pin\_notif\_get (*C function*), 481  
 pi\_gpio\_pin\_read (*C function*), 480  
 pi\_gpio\_pin\_task\_add (*C function*), 481  
 pi\_gpio\_pin\_task\_remove (*C function*), 482  
 pi\_gpio\_pin\_toggle (*C function*), 479  
 pi\_gpio\_pin\_write (*C function*), 480  
 pi\_himax\_conf (*C struct*), 603  
 pi\_himax\_conf.camera (*C var*), 603  
 pi\_himax\_conf.cpi\_itf (*C var*), 603  
 pi\_himax\_conf.format (*C var*), 603  
 pi\_himax\_conf.i2c\_itf (*C var*), 603  
 pi\_himax\_conf.roi (*C var*), 603  
 pi\_himax\_conf.skip\_pads\_config (*C var*), 603  
 pi\_himax\_conf\_init (*C function*), 602  
 pi\_hyper\_close (*C function*), 485  
 pi\_hyper\_conf (*C struct*), 492  
 pi\_hyper\_conf.baudrate (*C var*), 493  
 pi\_hyper\_conf.cs (*C var*), 493  
 pi\_hyper\_conf.device (*C var*), 493  
 pi\_hyper\_conf.id (*C var*), 493  
 pi\_hyper\_conf.type (*C var*), 493  
 pi\_hyper\_conf\_init (*C function*), 485  
 pi\_hyper\_conf\_t (*C type*), 484  
 pi\_hyper\_copy\_2d\_async (*C function*), 489  
 pi\_hyper\_copy\_async (*C function*), 488  
 pi\_hyper\_ioctl (*C function*), 485  
 pi\_hyper\_ioctl\_cmd (*C enum*), 485  
 pi\_hyper\_ioctl\_cmd.PI\_HYPER\_IOCTL\_SET\_LATENCY  
     (*C enumerator*), 485  
 pi\_hyper\_offset (*C function*), 492  
 pi\_hyper\_open (*C function*), 485  
 pi\_hyper\_read (*C function*), 485  
 pi\_hyper\_read\_2d (*C function*), 487

pi\_hyper\_read\_2d\_async (*C function*), 487  
pi\_hyper\_read\_async (*C function*), 486  
pi\_hyper\_type\_e (*C enum*), 484  
pi\_hyper\_type\_e.PI\_HYPER\_TYPE\_FLASH (*C enumerator*), 484  
pi\_hyper\_type\_e.PI\_HYPER\_TYPE\_RAM (*C enumerator*), 484  
pi\_hyper\_write (*C function*), 486  
pi\_hyper\_write\_2d (*C function*), 488  
pi\_hyper\_write\_2d\_async (*C function*), 488  
pi\_hyper\_write\_async (*C function*), 486  
pi\_hyper\_xip\_lock (*C function*), 492  
pi\_hyper\_xip\_unlock (*C function*), 492  
pi\_hyperflash\_conf (*C struct*), 614  
pi\_hyperflash\_conf.baudrate (*C var*), 615  
pi\_hyperflash\_conf.flash (*C var*), 615  
pi\_hyperflash\_conf.hyper\_cs (*C var*), 615  
pi\_hyperflash\_conf.hyper\_itf (*C var*), 615  
pi\_hyperflash\_conf.skip\_pads\_config (*C var*), 615  
pi\_hyperflash\_conf\_init (*C function*), 614  
pi\_hyperflash\_deep\_sleep\_enter (*C function*), 614  
pi\_hyperflash\_deep\_sleep\_exit (*C function*), 614  
pi\_hyperram\_conf (*C struct*), 641  
pi\_hyperram\_conf.baudrate (*C var*), 641  
pi\_hyperram\_conf.hyper\_cs (*C var*), 641  
pi\_hyperram\_conf.hyper\_itf (*C var*), 641  
pi\_hyperram\_conf.ram (*C var*), 641  
pi\_hyperram\_conf.ram\_size (*C var*), 641  
pi\_hyperram\_conf.ram\_start (*C var*), 641  
pi\_hyperram\_conf.reserve\_addr\_0 (*C var*), 642  
pi\_hyperram\_conf.skip\_pads\_config (*C var*), 641  
pi\_hyperram\_conf.xip\_en (*C var*), 641  
pi\_hyperram\_conf\_init (*C function*), 641  
pi\_i2c\_callback\_t (*C type*), 500  
pi\_i2c\_close (*C function*), 495  
pi\_i2c\_conf (*C struct*), 499  
pi\_i2c\_conf.cs (*C var*), 499  
pi\_i2c\_conf.itf (*C var*), 499  
pi\_i2c\_conf.max\_baudrate (*C var*), 499  
pi\_i2c\_conf.ts\_ch (*C var*), 499  
pi\_i2c\_conf.ts\_evt\_id (*C var*), 500  
pi\_i2c\_conf\_init (*C function*), 495  
pi\_i2c\_conf\_set\_slave\_addr (*C function*), 495  
pi\_i2c\_conf\_set\_wait\_cycles (*C function*), 495  
pi\_i2c\_conf\_t (*C type*), 493  
pi\_i2c\_detect (*C function*), 499  
pi\_i2c\_get\_request\_status (*C function*), 499  
pi\_i2c\_ioctl (*C function*), 496  
pi\_i2c\_ioctl\_e (*C enum*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_CTRL\_SET\_MAX\_BAUDRATE (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_ABORT\_RX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_ABORT\_TX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_ATTACH\_TIMEOUT\_RX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_ATTACH\_TIMEOUT\_TX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_DETACH\_TIMEOUT\_RX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_DETACH\_TIMEOUT\_TX (*C enumerator*), 494  
pi\_i2c\_ioctl\_e.PI\_I2C\_IOCTL\_EN\_TIMESTAMP (*C enumerator*), 495  
pi\_i2c\_open (*C function*), 495  
pi\_i2c\_read (*C function*), 496  
pi\_i2c\_read\_async (*C function*), 497  
pi\_i2c\_read\_timeout (*C function*), 497  
pi\_i2c\_slave\_args (*C struct*), 502  
pi\_i2c\_slave\_args\_t (*C type*), 500  
pi\_i2c\_slave\_close (*C function*), 501  
pi\_i2c\_slave\_conf (*C struct*), 502  
pi\_i2c\_slave\_conf.addr0 (*C var*), 503  
pi\_i2c\_slave\_conf.addr0\_10\_bit (*C var*), 502  
pi\_i2c\_slave\_conf.addr1 (*C var*), 503  
pi\_i2c\_slave\_conf.addr1\_10\_bit (*C var*), 502  
pi\_i2c\_slave\_conf.eof0 (*C var*), 502  
pi\_i2c\_slave\_conf.eof1 (*C var*), 502  
pi\_i2c\_slave\_conf.itf (*C var*), 502  
pi\_i2c\_slave\_conf.mask0 (*C var*), 503  
pi\_i2c\_slave\_conf.mask1 (*C var*), 503  
pi\_i2c\_slave\_conf.max\_baudrate (*C var*), 503  
pi\_i2c\_slave\_conf.rx\_callback (*C var*), 503  
pi\_i2c\_slave\_conf.sof0 (*C var*), 502  
pi\_i2c\_slave\_conf.sof1 (*C var*), 502  
pi\_i2c\_slave\_conf.tx\_callback (*C var*), 503  
pi\_i2c\_slave\_conf\_init (*C function*), 500  
pi\_i2c\_slave\_conf\_set\_addr0 (*C function*), 500  
pi\_i2c\_slave\_conf\_set\_addr1 (*C function*), 501  
pi\_i2c\_slave\_conf\_t (*C type*), 500  
pi\_i2c\_slave\_ioctl (*C function*), 502  
pi\_i2c\_slave\_ioctl\_e (*C enum*), 500  
pi\_i2c\_slave\_ioctl\_e.PI\_I2C\_SLAVE\_CTRL\_SET\_MAX\_BAUDRATE (*C enumerator*), 500  
pi\_i2c\_slave\_open (*C function*), 501  
pi\_i2c\_slave\_set\_rx (*C function*), 501  
pi\_i2c\_slave\_set\_tx (*C function*), 501  
pi\_i2c\_slave\_stop\_rx (*C function*), 502  
pi\_i2c\_slave\_stop\_tx (*C function*), 502  
pi\_i2c\_slave\_unlock (*C function*), 501  
pi\_i2c\_write (*C function*), 496  
pi\_i2c\_write\_async (*C function*), 497  
pi\_i2c\_write\_dual (*C function*), 496  
pi\_i2c\_write\_dual\_async (*C function*), 497  
pi\_i2c\_write\_read (*C function*), 496  
pi\_i2c\_write\_read\_async (*C function*), 497

pi\_i2c\_write\_timeout (*C function*), 498  
 pi\_i2c\_xfer\_flags\_e (*C enum*), 493  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_NO\_RESTART  
     (*C enumerator*), 494  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_NO\_START (*C  
     enumerator*), 494  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_NO\_STOP (*C  
     enumerator*), 493  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_RESTART (*C  
     enumerator*), 494  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_START (*C  
     enumerator*), 493  
 pi\_i2c\_xfer\_flags\_e.PI\_I2C\_XFER\_STOP (*C  
     enumerator*), 493  
 PI\_I2S\_CH\_FMT\_DATA\_ALIGN\_LEFT (*C macro*), 504  
 PI\_I2S\_CH\_FMT\_DATA\_ALIGN\_RIGHT (*C macro*), 504  
 PI\_I2S\_CH\_FMT\_DATA\_ORDER\_LSB (*C macro*), 504  
 PI\_I2S\_CH\_FMT\_DATA\_ORDER\_MSB (*C macro*), 504  
 PI\_I2S\_CH\_FMT\_DATA\_SIGN\_EXTEND (*C macro*), 504  
 PI\_I2S\_CH\_FMT\_DATA\_SIGN\_NO\_EXTEND (*C macro*),  
     504  
 pi\_i2s\_channel\_conf (*C struct*), 518  
 pi\_i2s\_channel\_conf.block\_size (*C var*), 519  
 pi\_i2s\_channel\_conf.format (*C var*), 519  
 pi\_i2s\_channel\_conf.mem\_slab (*C var*), 519  
 pi\_i2s\_channel\_conf.mem\_word\_size (*C var*), 519  
 pi\_i2s\_channel\_conf.options (*C var*), 519  
 pi\_i2s\_channel\_conf.pingpong\_buffers (*C var*),  
     519  
 pi\_i2s\_channel\_conf.slot\_enable (*C var*), 519  
 pi\_i2s\_channel\_conf.stream\_id (*C var*), 519  
 pi\_i2s\_channel\_conf.ts\_evt\_id (*C var*), 519  
 pi\_i2s\_channel\_conf.word\_size (*C var*), 519  
 pi\_i2s\_channel\_get (*C function*), 511  
 pi\_i2s\_channel\_init (*C function*), 510  
 pi\_i2s\_channel\_set (*C function*), 510  
 pi\_i2s\_channel\_read (*C function*), 513  
 pi\_i2s\_channel\_read\_async (*C function*), 513  
 pi\_i2s\_channel\_timestamp\_set (*C function*), 511  
 pi\_i2s\_channel\_write (*C function*), 515  
 pi\_i2s\_channel\_write\_async (*C function*), 515  
 pi\_i2s\_close (*C function*), 508  
 pi\_i2s\_conf (*C struct*), 517  
 pi\_i2s\_conf.block\_size (*C var*), 517  
 pi\_i2s\_conf.channels (*C var*), 518  
 pi\_i2s\_conf.format (*C var*), 517  
 pi\_i2s\_conf.frame\_clk\_freq (*C var*), 517  
 pi\_i2s\_conf.itf (*C var*), 518  
 pi\_i2s\_conf.mem\_slab (*C var*), 517  
 pi\_i2s\_conf.mem\_word\_size (*C var*), 518  
 pi\_i2s\_conf.options (*C var*), 518  
 pi\_i2s\_conf.pdm\_diff (*C var*), 518  
 pi\_i2s\_conf.pdm\_polarity (*C var*), 518  
 pi\_i2s\_conf.pingpong\_buffers (*C var*), 517  
 pi\_i2s\_conf.ref\_clk\_freq (*C var*), 518  
 pi\_i2s\_conf.ts\_evt\_id (*C var*), 518  
 pi\_i2s\_conf.word\_size (*C var*), 517  
 pi\_i2s\_ws\_delay (*C var*), 518  
 pi\_i2s\_ws\_type (*C var*), 518  
 pi\_i2s\_conf\_init (*C function*), 508  
 PI\_I2S\_FMT\_DATA\_FORMAT\_I2S (*C macro*), 503  
 PI\_I2S\_FMT\_DATA\_FORMAT\_PDM (*C macro*), 503  
 pi\_i2s\_fmt\_t (*C type*), 507  
 pi\_i2s\_frame\_channel\_conf\_set (*C function*), 511  
 pi\_i2s\_frame\_read (*C function*), 513  
 pi\_i2s\_frame\_read\_async (*C function*), 514  
 pi\_i2s\_frame\_write (*C function*), 516  
 pi\_i2s\_frame\_write\_async (*C function*), 516  
 pi\_i2s\_ioctl (*C function*), 508  
 pi\_i2s\_ioctl\_cmd\_e (*C enum*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_CLOCK\_DISABLE  
     (*C enumerator*), 508  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_CLOCK\_ENABLE  
     (*C enumerator*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_CONF\_GET (*C  
     enumerator*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_CONF\_SET (*C  
     enumerator*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_EN\_TIMESTAMP  
     (*C enumerator*), 508  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_START (*C  
     enumerator*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_STOP (*C  
     enumerator*), 507  
 pi\_i2s\_ioctl\_cmd\_e.PI\_I2S\_IOCTL\_SYNC (*C  
     enumerator*), 507  
 pi\_i2s\_open (*C function*), 508  
 PI\_I2S\_OPT\_CLK\_POLARITY\_FALLING\_EDGE  
     (*C  
         macro*), 506  
 PI\_I2S\_OPT\_CLK\_POLARITY\_RISING\_EDGE (*C macro*),  
     506  
 PI\_I2S\_OPT\_DISABLED (*C macro*), 505  
 PI\_I2S\_OPT\_ENABLED (*C macro*), 505  
 PI\_I2S\_OPT\_EXT\_CLK (*C macro*), 505  
 PI\_I2S\_OPT\_EXT\_CLK\_INT\_ROUTED (*C macro*), 506  
 PI\_I2S\_OPT\_EXT\_WS (*C macro*), 506  
 PI\_I2S\_OPT\_EXT\_WS\_INT\_ROUTED (*C macro*), 506  
 PI\_I2S\_OPT\_FULL\_DUPLEX (*C macro*), 504  
 PI\_I2S\_OPT\_INT\_CLK (*C macro*), 505  
 PI\_I2S\_OPT\_INT\_WS (*C macro*), 506  
 PI\_I2S\_OPT\_IS\_RX (*C macro*), 505  
 PI\_I2S\_OPT\_IS\_TX (*C macro*), 505  
 PI\_I2S\_OPT\_LOOPBACK (*C macro*), 505  
 PI\_I2S\_OPT\_MEM\_SLAB (*C macro*), 504  
 PI\_I2S\_OPT\_PINGPONG (*C macro*), 504  
 PI\_I2S\_OPT\_REF\_CLK\_FAST (*C macro*), 506  
 pi\_i2s\_opt\_t (*C type*), 507  
 PI\_I2S\_OPT\_TDM (*C macro*), 505

PI\_I2S\_OPT\_WS\_POLARITY\_FALLING\_EDGE (*C macro*), 506  
PI\_I2S\_OPT\_WS\_POLARITY\_RISING\_EDGE (*C macro*), 506  
pi\_i2s\_read (*C function*), 509  
pi\_i2s\_read\_async (*C function*), 509  
pi\_i2s\_read\_status (*C function*), 514  
pi\_i2s\_setup (*C function*), 508  
pi\_i2s\_slots\_enable (*C function*), 512  
pi\_i2s\_slots\_stop (*C function*), 512  
pi\_i2s\_slots\_stop\_async (*C function*), 512  
pi\_i2s\_write (*C function*), 509  
pi\_i2s\_write\_async (*C function*), 510  
pi\_i2s\_write\_status (*C function*), 517  
pi\_i3c\_close (*C function*), 521  
pi\_i3c\_conf (*C struct*), 522  
pi\_i3c\_conf.cs (*C var*), 523  
pi\_i3c\_conf.itf (*C var*), 523  
pi\_i3c\_conf.max\_baudrate (*C var*), 523  
pi\_i3c\_conf\_init (*C function*), 520  
pi\_i3c\_conf\_set\_slave\_addr (*C function*), 520  
pi\_i3c\_conf\_t (*C type*), 520  
pi\_i3c\_get\_request\_status (*C function*), 522  
pi\_i3c\_ioctl (*C function*), 521  
pi\_i3c\_ioctl\_e (*C enum*), 520  
pi\_i3c\_ioctl\_e.PI\_I3C\_CTRL\_SET\_MAX\_BAUDRATE  
    (*C enumerator*), 520  
pi\_i3c\_open (*C function*), 520  
pi\_i3c\_read (*C function*), 521  
pi\_i3c\_read\_async (*C function*), 522  
pi\_i3c\_write (*C function*), 521  
pi\_i3c\_write\_async (*C function*), 522  
PI\_INLINE\_XIP0 (*C macro*), 576  
pi\_l2\_available\_get (*C function*), 446  
pi\_l2\_free (*C function*), 445  
pi\_l2\_malloc (*C function*), 445  
pi\_l2\_malloc\_align (*C function*), 445  
pi\_l2\_malloc\_dump (*C function*), 446  
pi\_malloc (*C function*), 444  
pi\_malloc\_align (*C function*), 445  
pi\_malloc\_dump (*C function*), 445  
pi\_malloc\_init (*C function*), 444  
pi\_mt9v034\_conf (*C struct*), 604  
pi\_mt9v034\_conf.camera (*C var*), 604  
pi\_mt9v034\_conf.column\_flip (*C var*), 604  
pi\_mt9v034\_conf.cpi\_itf (*C var*), 604  
pi\_mt9v034\_conf.format (*C var*), 604  
pi\_mt9v034\_conf.i2c\_itf (*C var*), 604  
pi\_mt9v034\_conf.power\_gpio (*C var*), 604  
pi\_mt9v034\_conf.row\_flip (*C var*), 604  
pi\_mt9v034\_conf.skip\_pads\_config (*C var*), 604  
pi\_mt9v034\_conf.trigger\_gpio (*C var*), 604  
pi\_mt9v034\_conf\_init (*C function*), 604  
pi\_nina\_b312\_conf (*C struct*), 597  
pi\_nina\_b312\_conf.baudrate (*C var*), 598  
pi\_nina\_b312\_conf.local\_name (*C var*), 598  
pi\_nina\_b312\_conf.uart\_itf (*C var*), 598  
pi\_octospi\_close (*C function*), 526  
pi\_octospi\_cmd\_e (*C enum*), 525  
pi\_octospi\_cmd\_e.PI\_OCTOSPI\_CMD\_ADDR EVEN  
    (*C enumerator*), 525  
pi\_octospi\_cmd\_e.PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_EN  
    (*C enumerator*), 525  
pi\_octospi\_cmd\_e.PI\_OCTOSPI\_CMD\_AUTO\_RW\_BIT\_SET\_READ  
    (*C enumerator*), 525  
pi\_octospi\_conf (*C struct*), 533  
pi\_octospi\_conf.baudrate (*C var*), 534  
pi\_octospi\_conf.cs (*C var*), 533  
pi\_octospi\_conf.device (*C var*), 533  
pi\_octospi\_conf.id (*C var*), 533  
pi\_octospi\_conf.mba (*C var*), 534  
pi\_octospi\_conf.type (*C var*), 533  
pi\_octospi\_conf.xip\_en (*C var*), 533  
pi\_octospi\_conf\_init (*C function*), 526  
pi\_octospi\_conf\_t (*C type*), 523  
pi\_octospi\_flags\_e (*C enum*), 523  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_DTR  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_0  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_1  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_2  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_3  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_SIZE\_4  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_ADDR\_STR  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_CMD\_DTR  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_0  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_1  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_CMD\_SIZE\_2  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_CMD\_STR  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_DATA\_DTR  
    (*C enumerator*), 524  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_DATA\_DTR\_LSB  
    (*C enumerator*), 525  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_DATA\_DTR\_MSB  
    (*C enumerator*), 525  
pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_DATA\_STR  
    (*C enumerator*), 524

pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_LINE\_OCTO (C enumerator), 524  
 pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_LINE\_QUAD (C enumerator), 524  
 pi\_octospi\_flags\_e.PI\_OCTOSPI\_FLAG\_LINE\_SINGLE (C enumerator), 524  
 pi\_octospi\_ioctl (C function), 526  
 pi\_octospi\_ioctl\_cmd (C enum), 525  
 pi\_octospi\_ioctl\_cmd.PI\_OCTOSPI\_IOCTL\_SET\_OP (C enumerator), 525  
 pi\_octospi\_ioctl\_cmd.PI\_OCTOSPI\_IOCTL\_SET\_XIP (C enumerator), 525  
 pi\_octospi\_op\_conf (C struct), 533  
 pi\_octospi\_op\_conf.cmd (C var), 533  
 pi\_octospi\_op\_conf.flags (C var), 533  
 pi\_octospi\_op\_conf.latency (C var), 533  
 pi\_octospi\_op\_conf\_t (C type), 523  
 pi\_octospi\_open (C function), 526  
 pi\_octospi\_read (C function), 526  
 pi\_octospi\_read\_2d (C function), 528  
 pi\_octospi\_read\_2d\_async (C function), 528  
 pi\_octospi\_read\_async (C function), 527  
 pi\_octospi\_type\_e (C enum), 523  
 pi\_octospi\_type\_e.PI\_OCTOSPI\_TYPE\_FLASH (C enumerator), 523  
 pi\_octospi\_type\_e.PI\_OCTOSPI\_TYPE\_RAM (C enumerator), 523  
 pi\_octospi\_write (C function), 527  
 pi\_octospi\_write\_2d (C function), 529  
 pi\_octospi\_write\_2d\_async (C function), 529  
 pi\_octospi\_write\_async (C function), 527  
 pi\_octospi\_xip\_lock (C function), 532  
 pi\_octospi\_xip\_unlock (C function), 533  
 pi\_os\_close (C function), 438  
 pi\_os\_open (C function), 438  
 pi\_ov5647\_conf (C struct), 605  
 pi\_ov5647\_conf.camera (C var), 605  
 pi\_ov5647\_conf.csi2\_itf (C var), 605  
 pi\_ov5647\_conf.format (C var), 605  
 pi\_ov5647\_conf.i2c\_itf (C var), 605  
 pi\_ov5647\_conf.roi (C var), 605  
 pi\_ov5647\_conf.vc (C var), 605  
 pi\_ov5647\_conf\_init (C function), 605  
 pi\_ov7670\_conf (C struct), 603  
 pi\_ov7670\_conf.camera (C var), 603  
 pi\_ov7670\_conf.cpi\_itf (C var), 603  
 pi\_ov7670\_conf.format (C var), 603  
 pi\_ov7670\_conf.i2c\_itf (C var), 603  
 pi\_ov7670\_conf.skip\_pads\_config (C var), 603  
 pi\_ov7670\_conf\_init (C function), 602  
 pi\_ov9281\_conf (C struct), 606  
 pi\_ov9281\_conf.camera (C var), 606  
 pi\_ov9281\_conf.csi2\_itf (C var), 606  
 pi\_ov9281\_conf.format (C var), 606  
 pi\_ov9281\_conf.i2c\_itf (C var), 606  
 pi\_ov9281\_conf.roi (C var), 606  
 pi\_ov9281\_conf.vc (C var), 606  
 pi\_ov9281\_conf\_init (C function), 606  
 pi\_pad\_flags\_e (C enum), 534  
 pi\_pad\_flags\_e.PI\_PAD\_DS\_HIGH (C enumerator), 535  
 pi\_pad\_flags\_e.PI\_PAD\_DS\_LOW (C enumerator), 534  
 pi\_pad\_flags\_e.PI\_PAD\_PULL\_DISABLE (C enumerator), 534  
 pi\_pad\_flags\_e.PI\_PAD\_PULL\_ENABLE (C enumerator), 534  
 pi\_pad\_func\_e (C enum), 534  
 pi\_pad\_func\_e.PI\_PAD\_FUNC0 (C enumerator), 534  
 pi\_pad\_func\_e.PI\_PAD\_FUNC1 (C enumerator), 534  
 pi\_pad\_func\_e.PI\_PAD\_FUNC2 (C enumerator), 534  
 pi\_pad\_func\_e.PI\_PAD\_FUNC3 (C enumerator), 534  
 pi\_pad\_get\_function (C function), 535  
 pi\_pad\_init (C function), 535  
 pi\_pad\_set\_configuration (C function), 535  
 pi\_pad\_set\_function (C function), 535  
 pi\_pad\_sleepcfg\_flags\_e (C enum), 535  
 pi\_pad\_sleepcfg\_flags\_e.PI\_PAD\_SLEEPCFG\_ACTIVE\_HIGH (C enumerator), 535  
 pi\_pad\_sleepcfg\_flags\_e.PI\_PAD\_SLEEPCFG\_ACTIVE\_LOW (C enumerator), 535  
 pi\_pad\_sleepcfg\_flags\_e.PI\_PAD\_SLEEPCFG\_INPUT (C enumerator), 535  
 pi\_pad\_sleepcfg\_flags\_e.PI\_PAD\_SLEEPCFG\_OUTPUT (C enumerator), 535  
 pi\_pad\_sleepcfg\_set (C function), 536  
 pi\_perf\_conf (C function), 536  
 pi\_perf\_read (C function), 537  
 pi\_perf\_reset (C function), 536  
 pi\_perf\_start (C function), 537  
 pi\_perf\_stop (C function), 537  
 pi\_ram\_alloc (C function), 631  
 pi\_ram\_close (C function), 631  
 pi\_ram\_conf (C struct), 640  
 pi\_ram\_conf.api (C var), 641  
 pi\_ram\_copy (C function), 632  
 pi\_ram\_copy\_2d (C function), 633  
 pi\_ram\_copy\_2d\_async (C function), 636  
 pi\_ram\_copy\_async (C function), 634  
 pi\_ram\_free (C function), 632  
 pi\_ram\_info (C struct), 641  
 pi\_ram\_ioctl\_e (C enum), 631  
 pi\_ram\_ioctl\_e.PI\_RAM\_IOCTL\_AES\_ENABLE (C enumerator), 631  
 pi\_ram\_ioctl\_e.PI\_RAM\_IOCTL\_INFO (C enumerator), 631  
 pi\_ram\_ioctl\_e.PI\_RAM\_IOCTL\_SET\_MBA (C enumerator), 631  
 pi\_ram\_open (C function), 631

pi\_ram\_read (*C function*), 632  
pi\_ram\_read\_2d (*C function*), 633  
pi\_ram\_read\_2d\_async (*C function*), 635  
pi\_ram\_read\_async (*C function*), 634  
pi\_ram\_write (*C function*), 632  
pi\_ram\_write\_2d (*C function*), 633  
pi\_ram\_write\_2d\_async (*C function*), 635  
pi\_ram\_write\_async (*C function*), 634  
pi\_RTC\_alarm\_get (*C function*), 544  
pi\_RTC\_alarm\_repeat\_e (*C enum*), 541  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_DAY  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_HOUR  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_MIN  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_MON  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_NONE  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_SEC  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_repeat\_e.PI\_RTC\_ALARM\_RPT\_YEAR  
    (*C enumerator*), 542  
pi\_RTC\_alarm\_set (*C function*), 544  
pi\_RTC\_close (*C function*), 543  
pi\_RTC\_conf (*C struct*), 545  
pi\_RTC\_conf.alarm (*C var*), 545  
pi\_RTC\_conf.clk\_div (*C var*), 545  
pi\_RTC\_conf.counter (*C var*), 545  
pi\_RTC\_conf.mode (*C var*), 545  
pi\_RTC\_conf.rtc\_id (*C var*), 545  
pi\_RTC\_conf.time (*C var*), 545  
pi\_RTC\_conf\_init (*C function*), 543  
pi\_RTC\_datetime\_get (*C function*), 544  
pi\_RTC\_datetime\_set (*C function*), 543  
pi\_RTC\_ioctl (*C function*), 545  
pi\_RTC\_ioctl\_cmd\_e (*C enum*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_ALARM\_ATTACH\_TASK  
    (*C enumerator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_ALARM\_START (*C enu-  
merator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_ALARM\_STOP (*C enu-  
merator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_CALENDAR\_START (*C  
enumerator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_CALENDAR\_STOP (*C  
enumerator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_TIMER\_ATTACH\_TASK  
    (*C enumerator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_TIMER\_START (*C enu-  
merator*), 542  
pi\_RTC\_ioctl\_cmd\_e.PI\_RTC\_TIMER\_STOP (*C enu-  
merator*), 542  
pi\_RTC\_mode\_e (*C enum*), 541  
pi\_RTC\_mode\_e.PI\_RTC\_MODE\_ALARM (*C enumerator*),  
    541  
pi\_RTC\_mode\_e.PI\_RTC\_MODE\_CALENDAR (*C enum-  
erator*), 541  
pi\_RTC\_mode\_e.PI\_RTC\_MODE\_CALIBRATION (*C enu-  
merator*), 541  
pi\_RTC\_mode\_e.PI\_RTC\_MODE\_TIMER (*C enumerator*),  
    541  
pi\_RTC\_open (*C function*), 543  
pi\_RTC\_timer\_get (*C function*), 544  
pi\_RTC\_timer\_set (*C function*), 544  
pi\_sfu\_asrc\_ratio\_get (*C function*), 263  
pi\_sfu\_asrc\_ratio\_set (*C function*), 263  
pi\_sfu\_audio\_clock\_set (*C function*), 264  
pi\_sfu\_buffer\_data (*C function*), 262  
pi\_sfu\_buffer\_init (*C function*), 261  
pi\_sfu\_close (*C function*), 256  
pi\_sfu\_conf (*C struct*), 267  
pi\_sfu\_conf\_init (*C function*), 256  
pi\_sfu\_conf\_t (*C type*), 256  
pi\_sfu\_disable (*C function*), 256  
pi\_sfu\_dump\_l2\_graph (*C function*), 260  
pi\_sfu\_dump\_l2\_graph\_patch (*C function*), 261  
pi\_sfu\_enable (*C function*), 256  
pi\_sfu\_enqueue (*C function*), 261  
pi\_sfu\_get\_mem\_port (*C function*), 261  
pi\_sfu\_get\_stream\_id (*C function*), 261  
pi\_sfu\_graph\_apply\_clone\_patch (*C function*), 258  
pi\_sfu\_graph\_apply\_dynamic\_clone\_patch  
    (*C  
function*), 258  
pi\_sfu\_graph\_bind\_i2s (*C function*), 260  
pi\_sfu\_graph\_bind\_pdm (*C function*), 260  
pi\_sfu\_graph\_clone (*C function*), 258  
pi\_sfu\_graph\_close (*C function*), 258  
pi\_sfu\_graph\_dynamic\_clone (*C function*), 258  
pi\_sfu\_graph\_end\_wait (*C function*), 259  
pi\_sfu\_graph\_end\_wait\_async (*C function*), 260  
pi\_sfu\_graph\_get\_gfu\_filter\_refs (*C function*),  
    263  
pi\_sfu\_graph\_load (*C function*), 259  
pi\_sfu\_graph\_load\_async (*C function*), 259  
pi\_sfu\_graph\_open (*C function*), 257  
pi\_sfu\_graph\_open\_in\_group (*C function*), 257  
pi\_sfu\_graph\_reconfigure (*C function*), 262  
pi\_sfu\_graph\_reconfigure\_async (*C function*), 263  
pi\_sfu\_graph\_save (*C function*), 262  
pi\_sfu\_graph\_save\_async (*C function*), 262  
pi\_sfu\_graph\_set\_filter\_coeffs (*C function*), 263  
pi\_sfu\_graph\_unload (*C function*), 259  
pi\_sfu\_open (*C function*), 256  
pi\_sfu\_reset (*C function*), 256  
pi\_sfu\_status\_asrc\_lock\_get (*C function*), 263  
pi\_sfu\_switch (*C function*), 256

pi\_spi\_close (*C function*), 549  
 pi\_spi\_conf (*C struct*), 551  
 pi\_spi\_conf.big\_endian (*C var*), 552  
 pi\_spi\_conf.cs (*C var*), 552  
 pi\_spi\_conf.dummy\_clk\_cycle (*C var*), 552  
 pi\_spi\_conf.dummy\_clk\_cycle\_mode (*C var*), 552  
 pi\_spi\_conf.is\_slave (*C var*), 552  
 pi\_spi\_conf.itf (*C var*), 552  
 pi\_spi\_conf.max\_baudrate (*C var*), 552  
 pi\_spi\_conf.max\_rcv\_chunk\_size (*C var*), 552  
 pi\_spi\_conf.max\_snd\_chunk\_size (*C var*), 552  
 pi\_spi\_conf.phase (*C var*), 552  
 pi\_spi\_conf.polarity (*C var*), 552  
 pi\_spi\_conf.ts\_ch (*C var*), 552  
 pi\_spi\_conf.ts\_evt\_id (*C var*), 552  
 pi\_spi\_conf.wordsize (*C var*), 552  
 pi\_spi\_conf\_init (*C function*), 548  
 pi\_spi\_conf\_t (*C struct*), 552  
 PI\_SPI\_DUMMY\_CLK\_CYCLE\_DEFAULT (*C macro*), 546  
 PI\_SPI\_DUMMY\_CLK\_CYCLE\_MAX (*C macro*), 546  
 PI\_SPI\_DUMMY\_CLK\_CYCLE\_MIN (*C macro*), 546  
 PI\_SPI\_DUMMY\_CLK\_CYCLE\_MODE\_DEFAULT (*C macro*), 546  
 pi\_spi\_dummy\_clk\_cycle\_mode\_e (*C enum*), 546  
 pi\_spi\_dummy\_clk\_cycle\_mode\_e.PI\_SPI\_DUMMY\_CLK\_CYCLE\_MODE\_0 (*C enumerator*), 547  
 pi\_spi\_dummy\_clk\_cycle\_mode\_e.PI\_SPI\_DUMMY\_CLK\_CYCLE\_MODE\_1 (*C enumerator*), 547  
 pi\_spi\_flags\_e (*C enum*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_COPY\_EXT2LOC (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_COPY\_LOC2EXT (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_CS\_AUTO (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_CS\_KEEP (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_CS\_NONE (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_LINES\_OCTAL (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_LINES\_QUAD (*C enumerator*), 548  
 pi\_spi\_flags\_e.PI\_SPI\_LINES\_SINGLE (*C enumerator*), 548  
 pi\_spi\_ioctl (*C function*), 549  
 pi\_spi\_ioctl\_e (*C enum*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_BIG\_ENDIAN (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_CPHA0 (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_CPHA1 (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_CPOL0 (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_CPOL1 (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_LITTLE\_ENDIAN (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_SET\_MAX\_BAUDRATE (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_SET\_TIMESTAMP (*C enumerator*), 548  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_WORDSIZE\_32 (*C enumerator*), 547  
 pi\_spi\_ioctl\_e.PI\_SPI\_CTRL\_WORDSIZE\_8 (*C enumerator*), 547  
 PI\_SPI\_IS\_VALID\_DUMMY\_CLK\_CYCLE (*C macro*), 546  
 PI\_SPI\_IS\_VALID\_DUMMY\_CLK\_CYCLE\_MODE (*C macro*), 546  
 pi\_spi\_open (*C function*), 548  
 pi\_spi\_phase\_e (*C enum*), 547  
 pi\_spi\_phase\_e.PI\_SPI\_PHASE\_0 (*C enumerator*), 547  
 pi\_spi\_phase\_e.PI\_SPI\_PHASE\_1 (*C enumerator*), 547  
 pi\_spi\_polarity\_e (*C enum*), 546  
 pi\_SPI\_POLARITY\_0 (*C enumerator*), 546  
 pi\_SPI\_POLARITY\_1 (*C enumerator*), 546  
 pi\_spi\_receive (*C function*), 549  
 pi\_spi\_receive\_async (*C function*), 550  
 pi\_spi\_send (*C function*), 549  
 pi\_spi\_send\_async (*C function*), 550  
 pi\_spi\_transfer (*C function*), 550  
 pi\_spi\_transfer\_async (*C function*), 551  
 pi\_spi\_wordsize\_e (*C enum*), 546  
 pi\_spi\_wordsize\_e.PI\_SPI\_WORDSIZE\_16 (*C enumerator*), 546  
 pi\_spi\_wordsize\_e.PI\_SPI\_WORDSIZE\_32 (*C enumerator*), 546  
 pi\_spi\_wordsize\_e.PI\_SPI\_WORDSIZE\_8 (*C enumerator*), 546  
 pi\_spiram\_conf (*C struct*), 640  
 pi\_spiram\_conf.baudrate (*C var*), 640  
 pi\_spiram\_conf.ram (*C var*), 640  
 pi\_spiram\_conf.ram\_size (*C var*), 640  
 pi\_spiram\_conf.ram\_start (*C var*), 640  
 pi\_spiram\_conf.skip\_pads\_config (*C var*), 640  
 pi\_spiram\_conf.spi\_cs (*C var*), 640  
 pi\_spiram\_conf.spi\_itf (*C var*), 640  
 pi\_spiram\_conf\_init (*C function*), 631  
 pi\_task\_block (*C macro*), 440  
 pi\_task\_callback (*C macro*), 440  
 pi\_task\_cancel\_delayed\_us (*C macro*), 440  
 pi\_task\_irq\_callback (*C macro*), 440

pi\_task\_push (*C macro*), 440  
pi\_task\_push\_delayed\_us (*C macro*), 440  
pi\_task\_push\_irq\_safe (*C macro*), 440  
pi\_task\_status\_get (*C macro*), 440  
pi\_task\_status\_set (*C macro*), 440  
pi\_task\_timeout\_set (*C macro*), 440  
pi\_task\_wait\_on (*C macro*), 440  
pi\_thread\_create (*C function*), 438  
pi\_thread\_delete (*C function*), 439  
pi\_thread\_get\_name (*C function*), 439  
pi\_thread\_resume (*C function*), 439  
pi\_thread\_suspend (*C function*), 439  
pi\_timestamp\_cnt\_gpio\_trig\_type\_e (*C enum*),  
    573  
pi\_timestamp\_cnt\_gpio\_trig\_type\_e.PI\_TIMESTAMP  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_gpio\_trig\_type\_e.PI\_TIMESTAMP  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_gpio\_trig\_type\_e.PI\_TIMESTAMP  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_src\_e (*C enum*), 573  
pi\_timestamp\_cnt\_src\_e.PI\_TIMESTAMP\_CNT\_GPIO  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_src\_e.PI\_TIMESTAMP\_CNT\_PWM  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_src\_e.PI\_TIMESTAMP\_CNT\_REF\_CLK  
    (*C enumerator*), 573  
pi\_timestamp\_cnt\_src\_e.PI\_TIMESTAMP\_CNT\_SOC\_CLK  
    (*C enumerator*), 573  
pi\_timestamp\_conf (*C struct*), 574  
pi\_timestamp\_conf.cnt\_src (*C var*), 574  
pi\_timestamp\_conf.cnt\_src\_id (*C var*), 574  
pi\_timestamp\_conf.cnt\_trig\_gpio (*C var*), 574  
pi\_timestamp\_conf.cnt\_trig\_type (*C var*), 574  
pi\_timestamp\_conf.itf (*C var*), 574  
pi\_timestamp\_conf.prescaler (*C var*), 574  
pi\_timestamp\_conf\_init (*C function*), 574  
pi\_timestamp\_event\_t (*C struct*), 575  
pi\_timestamp\_input\_t (*C struct*), 574  
pi\_timestamp\_input\_t.input\_sel (*C var*), 574  
pi\_timestamp\_input\_t.input\_type (*C var*), 574  
pi\_timestamp\_input\_t.ts\_input\_id (*C var*), 574  
pi\_tlv320\_build\_default\_analog\_ch (*C function*),  
    672  
pi\_tlv320\_build\_default\_asi (*C function*), 672  
pi\_tlv320\_close (*C function*), 671  
pi\_tlv320\_conf (*C struct*), 690  
pi\_tlv320\_conf.addr0\_sclk (*C var*), 690  
pi\_tlv320\_conf.addr1\_miso (*C var*), 690  
pi\_tlv320\_conf.i2c\_itf (*C var*), 690  
pi\_tlv320\_conf.i2c\_max\_baudrate (*C var*), 690  
pi\_tlv320\_conf.shdnz (*C var*), 690  
pi\_tlv320\_conf\_init (*C function*), 671  
pi\_tlv320\_get\_asi (*C function*), 672  
pi\_tlv320\_open (*C function*), 671  
pi\_tlv320\_power (*C function*), 675  
pi\_tlv320\_read\_analog\_ch (*C function*), 674  
pi\_tlv320\_read\_config\_register (*C function*), 671  
pi\_tlv320\_reset (*C function*), 672  
pi\_tlv320\_set\_asi (*C function*), 672  
pi\_tlv320\_set\_bias\_cfg (*C function*), 674  
pi\_tlv320\_set\_channel\_input (*C function*), 674  
pi\_tlv320\_set\_channel\_output (*C function*), 675  
pi\_tlv320\_set\_clk (*C function*), 676  
pi\_tlv320\_set\_master\_clk (*C function*), 676  
pi\_tlv320\_set\_shutdown\_mode (*C function*), 676  
pi\_tlv320\_set\_slave\_clk\_bclk (*C function*), 675  
pi\_tlv320\_set\_slave\_clk\_mclk (*C function*), 675  
pi\_tlv320\_set\_slave\_clk\_pll (*C function*), 675  
pi\_tlv320\_set\_sleep\_mode (*C function*), 672  
pi\_tlv320\_set\_txedge\_api\_t (*C struct*), 689  
pi\_tlv320\_shdnz\_api\_t.reset (*C var*), 690  
pi\_tlv320\_shdnz\_api\_t.set (*C var*), 690  
pi\_tlv320\_write\_analog\_ch (*C function*), 673  
pi\_tlv320\_write\_config\_register (*C function*),  
    671  
pi\_uart\_abort (*C function*), 565  
pi\_uart\_close (*C function*), 556  
pi\_uart\_conf (*C struct*), 566  
pi\_uart\_conf.baudrate\_bps (*C var*), 566  
pi\_uart\_conf.enable\_rx (*C var*), 566  
pi\_uart\_conf.enable\_tx (*C var*), 566  
pi\_uart\_conf.is\_usart (*C var*), 566  
pi\_uart\_conf.parity\_mode (*C var*), 566  
pi\_uart\_conf.stop\_bit\_count (*C var*), 566  
pi\_uart\_conf.uart\_id (*C var*), 566  
pi\_uart\_conf.usart\_phase (*C var*), 567  
pi\_uart\_conf.usart\_polarity (*C var*), 566  
pi\_uart\_conf.use\_ctrl\_flow (*C var*), 566  
pi\_uart\_conf.use\_fast\_clk (*C var*), 567  
pi\_uart\_conf.word\_size (*C var*), 566  
pi\_uart\_conf\_init (*C function*), 556  
pi\_uart\_get\_bytes\_left (*C function*), 565  
pi\_uart\_get\_curr\_addr (*C function*), 566  
pi\_uart\_ioctl (*C function*), 557  
pi\_uart\_ioctl\_cmd (*C enum*), 554  
pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ABORT\_RX (*C  
enumerator*), 554  
pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ABORT\_TX (*C  
enumerator*), 554  
pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ATTACH\_TIMEOUT\_RX  
    (*C enumerator*), 555  
pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ATTACH\_TIMEOUT\_TX  
    (*C enumerator*), 555  
pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_CONF\_SETUP  
    (*C enumerator*), 554

pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_DETACH\_TIMEOUT\_RX (*C enumerator*), 554  
     (*C enumerator*), 555  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_DETACH\_TIMEOUT\_TX (*C enumerator*), 554  
     (*C enumerator*), 555  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_DISABLE\_FLOW\_CONTROL (*C enumerator*), 555  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ENABLE\_FLOW\_CONTROL (*C enumerator*), 554  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ENABLE\_RX (*C enumerator*), 554  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_ENABLE\_TX (*C enumerator*), 554  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_FLUSH (*C enumerator*), 555  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_RESUME\_RX (*C enumerator*), 556  
 pi\_uart\_ioctl\_cmd.PI\_UART\_IOCTL\_RESUME\_TX (*C enumerator*), 555  
 pi\_uart\_open (*C function*), 556  
 pi\_uartparity\_mode (*C enum*), 553  
 pi\_uartparity\_mode.PI\_UART\_PARITY\_DISABLE  
     (*C enumerator*), 553  
 pi\_uartparity\_mode.PI\_UART\_PARITY\_ENABLE  
     (*C enumerator*), 553  
 pi\_uart\_pause (*C function*), 563  
 pi\_uart\_read (*C function*), 560  
 pi\_uart\_read\_async (*C function*), 559  
 pi\_uart\_read\_byte (*C function*), 560  
 pi\_uart\_read\_timeout (*C function*), 561  
 pi\_uart\_read\_timeout\_async (*C function*), 560  
 pi\_uart\_resume (*C function*), 564  
 pi\_uart\_resume\_async (*C function*), 564  
 pi\_uart\_resume\_timeout (*C function*), 565  
 pi\_uart\_resume\_timeout\_async (*C function*), 564  
 pi\_uart\_stop\_bits (*C enum*), 553  
 pi\_uart\_stop\_bits.PI\_UART\_STOP\_BITS\_ONE  
     (*C enumerator*), 553  
 pi\_uart\_stop\_bits.PI\_UART\_STOP\_BITS\_TWO  
     (*C enumerator*), 553  
 pi\_uart\_timeout\_flags (*C enum*), 556  
 pi\_uart\_timeout\_flags.PI\_UART\_TIMEOUT\_FLAG\_HW  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP16  
     (*C enumerator*), 556  
 pi\_uart\_timeout\_flags.PI\_UART\_TIMEOUT\_FLAG\_SW  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP24  
     (*C enumerator*), 556  
 pi\_uart\_timeout\_flags.PI\_UART\_TIMEOUT\_FLAG\_SW  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP32  
     (*C enumerator*), 556  
 pi\_uart\_timeout\_get\_bytes\_left (*C macro*), 553  
 pi\_uarttimeout\_get\_curr\_addr (*C macro*), 553  
 pi\_uart\_word\_size (*C enum*), 553  
 pi\_uart\_word\_size.PI\_UART\_WORD\_SIZE\_5\_BITS  
     (*C enumerator*), 553  
 pi\_uart\_word\_size.PI\_UART\_WORD\_SIZE\_6\_BITS  
     (*C enumerator*), 554  
 pi\_uart\_word\_size.PI\_UART\_WORD\_SIZE\_7\_BITS  
     (*C enumerator*), 554  
     pi\_uart\_word\_size.PI\_UART\_WORD\_SIZE\_8\_BITS  
     (*C enumerator*), 554  
     pi\_uart\_write (*C function*), 557  
     pi\_uart\_write\_byte (*C function*), 558  
     pi\_uart\_write\_byte\_async (*C function*), 557  
     pi\_uart\_write\_timeout (*C function*), 559  
     pi\_uart\_write\_timeout\_async (*C function*), 558  
     pi\_udma\_datamove\_close (*C function*), 568  
     pi\_udma\_datamove\_conf\_init (*C function*), 567  
     pi\_udma\_datamove\_conf\_s (*C struct*), 568  
     pi\_udma\_datamove\_conf\_s.device\_id (*C var*), 569  
     pi\_udma\_datamove\_conf\_s.dst\_trf\_cfg  
         (*C var*), 569  
     pi\_udma\_datamove\_conf\_s.src\_trf\_cfg  
         (*C var*), 569  
     pi\_udma\_datamove\_conf\_t (*C type*), 567  
     pi\_udma\_datamove\_copy (*C function*), 568  
     pi\_udma\_datamove\_copy\_async (*C function*), 568  
     pi\_udma\_datamove\_open (*C function*), 567  
     pi\_udma\_datamove\_transf\_cfg\_s (*C struct*), 568  
     pi\_udma\_datamove\_transf\_cfg\_t (*C type*), 567  
     pi\_udma\_datamove\_transf\_type\_e (*C enum*), 567  
     pi\_udma\_datamove\_transf\_type\_e.PI\_UDMA\_DATAMOVE\_TRF\_2D  
         (*C enumerator*), 567  
     pi\_udma\_datamove\_transf\_type\_e.PI\_UDMA\_DATAMOVE\_TRF\_LINEAR  
         (*C enumerator*), 567  
     pi\_udma\_fifo\_close (*C function*), 571  
     pi\_udma\_fifo\_conf (*C struct*), 571  
     pi\_udma\_fifo\_conf.device (*C var*), 571  
     pi\_udma\_fifo\_conf.size (*C var*), 571  
     pi\_udma\_fifo\_conf\_init (*C function*), 570  
     pi\_udma\_fifo\_conf\_t (*C type*), 569  
     pi\_udma\_fifo\_ioctl (*C function*), 571  
     pi\_udma\_fifo\_ioctl\_e (*C enum*), 569  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_GET\_ID  
         (*C enumerator*), 569  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_GET\_LEVEL  
         (*C enumerator*), 569  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP16  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP24  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP32  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_POP8  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_PUSH16  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_PUSH24  
         (*C enumerator*), 570  
     pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_PUSH32  
         (*C enumerator*), 570

pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_PUSH8 (C function), 711  
*enumerator*, 570  
 pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_SET\_EVENT\_CB (C function), 570  
*enumerator*, 570  
 pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_SET\_EVENT\_THRESHOLD 577  
*enumerator*, 569  
 pi\_udma\_fifo\_ioctl\_e.PI\_UDMA\_FIFO\_SET\_SIZE (C function), 569  
*enumerator*, 569  
 pi\_udma\_fifo\_open (C function), 570  
 pi\_udma\_timeout\_alloc (C function), 572  
 pi\_udma\_timeout\_free (C function), 572  
 pi\_udma\_timeout\_ioctl (C function), 572  
 pi\_udma\_timeout\_ioctl\_cmd (C enum), 572  
 pi\_udma\_timeout\_ioctl\_cmd.PI\_UDMA\_TIMEOUT\_IOCTL 577  
*enumerator*, 572  
 pi\_udma\_timeout\_ioctl\_cmd.PI\_UDMA\_TIMEOUT\_IOCTL 577  
*enumerator*, 572  
 pi\_udma\_timeout\_mode\_e (C enum), 572  
 pi\_udma\_timeout\_mode\_e.PI\_UDMA\_TIMEOUT\_MODE\_RX 577  
*enumerator*, 572  
 pi\_udma\_timeout\_mode\_e.PI\_UDMA\_TIMEOUT\_MODE\_SW\_TRIGGER 576  
*enumerator*, 572  
 pi\_udma\_timeout\_mode\_e.PI\_UDMA\_TIMEOUT\_MODE\_TRANSFER 576  
*enumerator*, 572  
 pi\_udma\_timestamp\_close (C function), 574  
 pi\_udma\_timestamp\_ioctl (C function), 574  
 pi\_udma\_timestamp\_open (C function), 574  
 pi\_user\_thread\_create (C function), 439  
 pi\_watchdog\_start (C function), 575  
 pi\_watchdog\_stop (C function), 575  
 pi\_watchdog\_timer\_rearm (C function), 575  
 pi\_watchdog\_timer\_set (C function), 575  
 pi\_wifi\_api\_s (C struct), 714  
 pi\_wifi\_api\_s.at\_cmd (C var), 715  
 pi\_wifi\_api\_s.connect\_to\_ap (C var), 715  
 pi\_wifi\_api\_s.disconnect\_from\_ap (C var), 715  
 pi\_wifi\_api\_t (C type), 711  
 pi\_wifi\_at\_cmd (C function), 711  
 pi\_wifi\_catch\_client\_connection (C function), 714  
 pi\_wifi\_catch\_event (C function), 714  
 pi\_wifi\_close (C function), 711  
 pi\_wifi\_connect\_to\_ap (C function), 712  
 pi\_wifi\_create\_tcp\_client (C function), 714  
 pi\_wifi\_create\_tcp\_server (C function), 713  
 pi\_wifi\_da16200\_client\_ip\_port\_set (C function), 715  
 pi\_wifi\_da16200\_conf\_init (C function), 715  
 pi\_wifi\_data\_get (C function), 713  
 pi\_wifi\_data\_get\_async (C function), 713  
 pi\_wifi\_data\_send (C function), 712  
 pi\_wifi\_data\_send\_async (C function), 712  
 pi\_wifi\_disconnect\_from\_ap (C function), 712  
 pi\_wifi\_ioctl (C function), 711  
 pi\_wifi\_open (C function), 711  
 pi\_xip\_conf\_s (C struct), 577  
 pi\_xip\_conf\_s.cl\_irq\_handler (C var), 577  
 pi\_xip\_conf\_s.data\_exception\_handler (C var), 577  
 pi\_xip\_conf\_s.instr\_exception\_handler (C var), 577  
 pi\_xip\_conf\_s.page\_mask (C var), 577  
 pi\_xip\_conf\_s.page\_size (C var), 577  
 pi\_xip\_conf\_s.per\_id (C var), 577  
 pi\_xip\_conf\_s.ro (C var), 577  
 pi\_xip\_conf\_s.tlb\_en (C var), 577  
 pi\_xip\_conf\_t (C type), 576  
 pi\_xip\_data\_err\_s (C struct), 577  
 pi\_xip\_data\_err\_s.error\_addr (C var), 578  
 pi\_xip\_data\_err\_s.violation (C var), 578  
 pi\_xip\_data\_err\_t (C type), 576  
 pi\_xip\_dcache\_page\_alloc (C function), 576  
 pi\_xip\_dcache\_page\_free (C function), 577  
 pi\_xip\_free\_page\_mask\_get (C function), 576  
 pi\_xip\_unmount (C function), 576  
 plot\_mem\_usage() (nntool.api.NNNGraph method), 420  
 pmsis\_exit (C function), 438  
 pmsis\_kickoff (C function), 438  
 PoolingNodeBase (class in nntool.api.types), 434  
 precision\_key() (nntool.api.quantization.QType static method), 433  
 print\_progress() (in module nntool.api.compression), 433  
 Proxy (class in gv.gvsoc\_control), 333  
 Proxy (class in gvsoc\_control), 581

## Q

QRec (class in nntool.api.quantization), 433  
 QSet (in module nntool.api.quantization), 433  
 qshow() (nntool.api.NNNGraph method), 421  
 qsnr() (in module nntool.api.utils), 431  
 qsnrs() (in module nntool.api.utils), 431  
 qsnrs() (nntool.api.NNNGraph static method), 421  
 QType (class in nntool.api.quantization), 433  
 quantization (nntool.api.NNNGraph property), 421  
 quantization\_options() (in module nntool.api.utils), 431  
 quantize() (nntool.api.NNNGraph method), 421  
 quit() (gv.gvsoc\_control.Proxy method), 334  
 quit() (gvsoc\_control.Proxy method), 582

## R

RandomIter (class in nntool.api.utils), 424  
 RawHeader (class in gapylib.fs.raw), 391  
 RawSection (class in gapylib.fs.raw), 391

**S**

**section\_id** (*gapylib.chips.gap.rom\_v2.RomFlashSection attribute*), 393  
**section\_id** (*gapylib.fs.hostfs.HostfsSection attribute*), 391  
**section\_id** (*gapylib.fs.littlefs.LfsSection attribute*), 390  
**section\_id** (*gapylib.fs.partition.PartitionTableSection attribute*), 387  
**section\_id** (*gapylib.fs.raw.RawSection attribute*), 391  
**section\_id** (*gapylib.fs.readfs.ReadfsSection attribute*), 389  
**set()** (*gapylib.utils.CStructField method*), 385  
**set\_content()** (*gapylib.chips.gap.rom\_v2.RomFlashSection method*), 394  
**set\_content()** (*gapylib.flash.Flash method*), 379  
**set\_content()** (*gapylib.flash.FlashSection method*), 382  
**set\_content()** (*gapylib.fs.hostfs.HostfsSection method*), 391  
**set\_content()** (*gapylib.fs.littlefs.LfsSection method*), 390  
**set\_content()** (*gapylib.fs.partition.PartitionTableSection method*), 387  
**set\_content()** (*gapylib.fs.raw.RawSection method*), 392  
**set\_content()** (*gapylib.fs.readfs.ReadfsSection method*), 389  
**set\_field()** (*gapylib.utils.CStruct method*), 384  
**set\_node\_option()** (*nntool.api.NNGraph method*), 422  
**set\_offset()** (*gapylib.flash.FlashSection method*), 382  
**set\_properties()** (*gapylib.flash.Flash method*), 379  
**set\_target\_dirs()** (*gapylib.target.Target method*), 376  
**set\_working\_dir()** (*gapylib.target.Target method*), 377  
**settings** (*nntool.api.NNGraph property*), 422  
**show()** (*nntool.api.NNGraph method*), 422  
**size** (*gapylib.chips.gap.flash.DefaultFlashRomV2 attribute*), 393  
**size** (*gapylib.flash.Flash attribute*), 377  
**size** (*gapylib.fs.readfs.ReadfsFile attribute*), 388  
**size** (*gapylib.utils.CStructField attribute*), 385  
**slot\_close()** (*gv.gvsoc\_control.Testbench\_i2s method*), 337  
**slot\_close()** (*gvsoc\_control.Testbench\_i2s method*), 585  
**slot\_open()** (*gv.gvsoc\_control.Testbench\_i2s method*), 337  
**slot\_open()** (*gvsoc\_control.Testbench\_i2s method*), 585  
**slot\_rx\_file\_reader()** (*gv.gvsoc\_control.Testbench\_i2s method*), 337

slot_rx_file_reader() ( <i>gvsoc_control.Testbench_i2s method</i> ), 585	TLV320_ASI_CFG0_ASI_FORMAT_TDM ( <i>C macro</i> ), 645
slot_stop() ( <i>gv.gvsoc_control.Testbench_i2s method</i> ), 338	TLV320_ASI_CFG0_ASI_WLEN_16BITS ( <i>C macro</i> ), 646
slot_stop() ( <i>gvsoc_control.Testbench_i2s method</i> ), 586	TLV320_ASI_CFG0_ASI_WLEN_20BITS ( <i>C macro</i> ), 646
slot_tx_file_dumper()	TLV320_ASI_CFG0_ASI_WLEN_24BITS ( <i>C macro</i> ), 646
( <i>gv.gvsoc_control.Testbench_i2s method</i> ), 338	TLV320_ASI_CFG0_ASI_WLEN_32BITS ( <i>C macro</i> ), 646
slot_tx_file_dumper() ( <i>gvsoc_control.Testbench_i2s method</i> ), 586	TLV320_ASI_CFG0_BCLK_POL_INVERTED ( <i>C macro</i> ), 646
stop() ( <i>gv.gvsoc_control.Proxy method</i> ), 334	TLV320_ASI_CFG0_BCLK_POL_NORMAL ( <i>C macro</i> ), 646
stop() ( <i>gvsoc_control.Proxy method</i> ), 582	TLV320_ASI_CFG0_FSYNC_POL_INVERTED ( <i>C macro</i> ), 646
<b>T</b>	
Target ( <i>class in gap9.evk</i> ), 392	TLV320_ASI_CFG0_FSYNC_POL_NORMAL ( <i>C macro</i> ), 646
Target ( <i>class in gap9.gap9_v2</i> ), 392	tlv320_asi_cfg0_t ( <i>C union</i> ), 681
Target ( <i>class in gapylib.target</i> ), 375	tlv320_asi_cfg0_t.asi_format ( <i>C var</i> ), 681
target ( <i>gapylib.chips.gap.flash.DefaultFlashRomV2 attribute</i> ), 392	tlv320_asi_cfg0_t.asi_wlen ( <i>C var</i> ), 681
target ( <i>gapylib.chips.gap.gap9_v2.board_runner.Runner attribute</i> ), 395	tlv320_asi_cfg0_t.bclk_pol ( <i>C var</i> ), 681
target ( <i>gapylib.flash.Flash attribute</i> ), 377	tlv320_asi_cfg0_t.fsync_pol ( <i>C var</i> ), 681
Testbench ( <i>class in gv.gvsoc_control</i> ), 335	tlv320_asi_cfg0_t.raw ( <i>C var</i> ), 681
Testbench ( <i>class in gvsoc_control</i> ), 583	tlv320_asi_cfg0_t.tx_edge ( <i>C var</i> ), 681
Testbench_i2s ( <i>class in gv.gvsoc_control</i> ), 336	tlv320_asi_cfg0_t.tx_fill ( <i>C var</i> ), 681
Testbench_i2s ( <i>class in gvsoc_control</i> ), 584	tlv320_asi_cfg0_t.[anonymous] ( <i>C var</i> ), 681
Testbench_uart ( <i>class in gv.gvsoc_control</i> ), 338	TLV320_ASI_CFG0_TX_EDGE_INVERTED ( <i>C macro</i> ), 646
Testbench_uart ( <i>class in gvsoc_control</i> ), 586	TLV320_ASI_CFG0_TX_EDGE_NORMAL ( <i>C macro</i> ), 646
TLV320_ADDR0_SLAVE_ADDR_BITSHIFT ( <i>C macro</i> ), 642	TLV320_ASI_CFG0_TX_FILL_HI_Z ( <i>C macro</i> ), 646
TLV320_ADDR1_SLAVE_ADDR_BITSHIFT ( <i>C macro</i> ), 642	TLV320_ASI_CFG0_TX_FILL_ZERO ( <i>C macro</i> ), 646
tlv320_afc_fscale_t ( <i>C enum</i> ), 670	tlv320_asi_cfg1_t ( <i>C union</i> ), 682
tlv320_afc_fscale_t.TLV320_ADC_FSCALE_1_375	tlv320_asi_cfg1_t.raw ( <i>C var</i> ), 682
( <i>C enumerator</i> ), 671	tlv320_asi_cfg1_t.tx_keeper ( <i>C var</i> ), 682
tlv320_afc_fscale_t.TLV320_ADC_FSCALE_VREF_2_5	tlv320_asi_cfg1_t.tx_lsb ( <i>C var</i> ), 682
( <i>C enumerator</i> ), 670	tlv320_asi_cfg1_t.tx_offset ( <i>C var</i> ), 682
tlv320_afc_fscale_t.TLV320_ADC_FSCALE_VREF_2_75	tlv320_asi_cfg1_t.[anonymous] ( <i>C var</i> ), 682
( <i>C enumerator</i> ), 670	TLV320_ASI_CFG1_TX_KEEPER_DISABLED ( <i>C macro</i> ), 647
tlv320_analog_ch_t ( <i>C struct</i> ), 678	TLV320_ASI_CFG1_TX_KEEPER_ENABLED ( <i>C macro</i> ), 648
tlv320_analog_ch_t.gain ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_KEEPER_ENABLED_1_5_CYCLES ( <i>C macro</i> ), 648
tlv320_analog_ch_t.gain_calib ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_KEEPER_ENABLED_1_CYCLE ( <i>C macro</i> ), 648
tlv320_analog_ch_t.id ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_LSB_FULL_CYCLE ( <i>C macro</i> ), 647
tlv320_analog_ch_t.input ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_LSB_HALF_CYCLE ( <i>C macro</i> ), 647
tlv320_analog_ch_t.output ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_OFFSET_DEFAULT ( <i>C macro</i> ), 648
tlv320_analog_ch_t.phase_calib ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_OFFSET_DISABLED ( <i>C macro</i> ), 648
tlv320_analog_ch_t.volume ( <i>C var</i> ), 678	TLV320_ASI_CFG1_TX_OFFSET_MAX ( <i>C macro</i> ), 648
TLV320_ANALOG_CHANNEL_NB ( <i>C macro</i> ), 644	TLV320_ASI_CFG1_TX_OFFSET_MIN ( <i>C macro</i> ), 648
tlv320_asi_bclk_pol_t ( <i>C enum</i> ), 668	TLV320_ASI_CFG1_TX_OFFSET_SIZE ( <i>C macro</i> ), 648
tlv320_asi_bclk_pol_t.TLV320_BCLKPOL_INVERTED	tlv320_asi_ch_t ( <i>C union</i> ), 682
( <i>C enumerator</i> ), 668	tlv320_asi_ch_t.output ( <i>C var</i> ), 682
tlv320_asi_bclk_pol_t.TLV320_BCLKPOL_NORMAL	tlv320_asi_ch_t.raw ( <i>C var</i> ), 682
( <i>C enumerator</i> ), 668	tlv320_asi_ch_t.reserved ( <i>C var</i> ), 682
TLV320_ASI_CFG0_ASI_FORMAT_I2S ( <i>C macro</i> ), 645	tlv320_asi_ch_t.slot ( <i>C var</i> ), 682
TLV320_ASI_CFG0_ASI_FORMAT_LJ ( <i>C macro</i> ), 646	

tlv320\_asi\_ch\_t. [anonymous] (C var), 682  
 TLV320\_ASI\_CHx\_OUTPUT\_PRIMARY (C macro), 649  
 TLV320\_ASI\_CHx\_OUTPUT\_SECONDARY (C macro), 649  
 tlv320\_asi\_format\_t (C enum), 667  
 tlv320\_asi\_format\_t.TLV320\_I2S (C enumerator),  
     667  
 tlv320\_asi\_format\_t.TLV320\_LJ (C enumerator),  
     668  
 tlv320\_asi\_format\_t.TLV320\_TDM (C enumerator),  
     667  
 tlv320\_asi\_fsync\_pol\_t (C enum), 668  
 tlv320\_asi\_fsync\_pol\_t.TLV320\_FSYNCPOL\_INVERTED  
     (C enumerator), 668  
 tlv320\_asi\_fsync\_pol\_t.TLV320\_FSYNCPOL\_NORMAL  
     (C enumerator), 668  
 TLV320\_ASI\_OUT\_CH\_EN\_ENABLED (C macro), 661  
 tlv320\_asi\_out\_ch\_en\_t (C union), 688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch1\_en (C var),  
     689  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch2\_en (C var),  
     689  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch3\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch4\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch5\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch6\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch7\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.asi\_out\_ch8\_en (C var),  
     688  
 tlv320\_asi\_out\_ch\_en\_t.raw (C var), 689  
 tlv320\_asi\_out\_ch\_en\_t. [anonymous] (C var), 689  
 TLV320\_ASI\_OUT\_CH\_EN\_TRISTATE (C macro), 661  
 tlv320\_asi\_t (C struct), 676  
 tlv320\_asi\_t.bclk\_pol (C var), 676  
 tlv320\_asi\_t.format (C var), 676  
 tlv320\_asi\_t.fsync\_pol (C var), 676  
 tlv320\_asi\_t.tx\_edge (C var), 677  
 tlv320\_asi\_t.tx\_fill (C var), 677  
 tlv320\_asi\_t.tx\_keeper (C var), 677  
 tlv320\_asi\_t.tx\_lsb (C var), 677  
 tlv320\_asi\_t.tx\_offset (C var), 677  
 tlv320\_asi\_t.wlen (C var), 676  
 tlv320\_asi\_tx\_edge\_t (C enum), 668  
 tlv320\_asi\_tx\_edge\_t.TLV320\_TXEDGE\_INVERTED  
     (C enumerator), 668  
 tlv320\_asi\_tx\_edge\_t.TLV320\_TXEDGE\_NORMAL (C  
     enumerator), 668  
 tlv320\_asi\_tx\_fill\_t (C enum), 669  
 tlv320\_asi\_tx\_fill\_t.TLV320\_TXFILL\_HI\_Z (C  
     enumerator), 669  
 tlv320\_asi\_tx\_fill\_t.TLV320\_TXFILL\_ZERO (C  
     enumerator), 669  
 tlv320\_asi\_wlen\_t (C enum), 668  
 tlv320\_asi\_wlen\_t.TLV320\_WLEN\_16 (C enumera-  
     tor), 668  
 tlv320\_asi\_wlen\_t.TLV320\_WLEN\_20 (C enumera-  
     tor), 668  
 tlv320\_asi\_wlen\_t.TLV320\_WLEN\_24 (C enumera-  
     tor), 668  
 tlv320\_asi\_wlen\_t.TLV320\_WLEN\_32 (C enumera-  
     tor), 668  
 tlv320\_bias\_cfg\_t (C union), 685  
 tlv320\_bias\_cfg\_t.adc\_fscale (C var), 685  
 tlv320\_bias\_cfg\_t.mbias\_val (C var), 685  
 tlv320\_bias\_cfg\_t.raw (C var), 685  
 tlv320\_bias\_cfg\_t.reserved1 (C var), 685  
 tlv320\_bias\_cfg\_t.reserved2 (C var), 685  
 tlv320\_bias\_cfg\_t. [anonymous] (C var), 685  
 TLV320\_BIASCFG\_ADCFSCALE\_VREF\_1\_375 (C macro),  
     657  
 TLV320\_BIASCFG\_ADCFSCALE\_VREF\_2\_5 (C macro),  
     657  
 TLV320\_BIASCFG\_ADCFSCALE\_VREF\_2\_75 (C macro),  
     657  
 TLV320\_BIASCFG\_BIAS\_1\_096\_VREF (C macro), 657  
 TLV320\_BIASCFG\_BIAS\_AVDD (C macro), 657  
 TLV320\_BIASCFG\_BIAS\_VREF (C macro), 657  
 tlv320\_ch\_cfg0\_t (C union), 685  
 tlv320\_ch\_cfg0\_t.ch\_dc (C var), 686  
 tlv320\_ch\_cfg0\_t.ch\_dreen (C var), 686  
 tlv320\_ch\_cfg0\_t.ch\_imp (C var), 686  
 tlv320\_ch\_cfg0\_t.ch\_insrc (C var), 686  
 tlv320\_ch\_cfg0\_t.ch\_intyp (C var), 686  
 tlv320\_ch\_cfg0\_t.raw (C var), 686  
 tlv320\_ch\_cfg0\_t.reserved (C var), 686  
 tlv320\_ch\_cfg0\_t. [anonymous] (C var), 686  
 tlv320\_ch\_cfg1\_t (C union), 686  
 tlv320\_ch\_cfg1\_t.ch\_gain (C var), 686  
 tlv320\_ch\_cfg1\_t.raw (C var), 686  
 tlv320\_ch\_cfg1\_t.reserved (C var), 686  
 tlv320\_ch\_cfg1\_t. [anonymous] (C var), 686  
 tlv320\_ch\_cfg2\_t (C union), 686  
 tlv320\_ch\_cfg2\_t.ch\_dvol (C var), 687  
 tlv320\_ch\_cfg2\_t.raw (C var), 687  
 tlv320\_ch\_cfg2\_t. [anonymous] (C var), 687  
 tlv320\_ch\_cfg3\_t (C union), 687  
 tlv320\_ch\_cfg3\_t.ch\_gcal (C var), 687  
 tlv320\_ch\_cfg3\_t.raw (C var), 687  
 tlv320\_ch\_cfg3\_t.reserved (C var), 687  
 tlv320\_ch\_cfg3\_t. [anonymous] (C var), 687  
 tlv320\_ch\_cfg4\_t (C union), 687  
 tlv320\_ch\_cfg4\_t.ch\_pcal (C var), 687  
 tlv320\_ch\_cfg4\_t.raw (C var), 687  
 tlv320\_ch\_cfg4\_t. [anonymous] (C var), 687

tlv320_ch_input_agcdre_t (C enum), 670	TLV320_CHx_CFG0_DRE_AGC_DISABLED (C macro), 659
tlv320_ch_input_agcdre_t.TLV320_AGCDRE_DISABLED (C enumerator), 670	TLV320_CHx_CFG0_DRE_AGC_ENABLE (C macro), 659
tlv320_ch_input_agcdre_t.TLV320_AGCDRE_ENABLED (C enumerator), 670	TLV320_CHx_CFG0_INSRC_ANALOG_DIFF (C macro), 658
tlv320_ch_input_agcdre_t.TLV320_INPUT_AC_COUPLED (C enum), 670	TLV320_CHx_CFG0_INSRC_ANALOG_SINGLE (C macro), 658
tlv320_ch_input_agcdre_t.TLV320_INPUT_AC_COUPLED (C enumerator), 670	TLV320_CHx_CFG0_INSRC_DIGITAL (C macro), 658
tlv320_ch_input_agcdre_t.TLV320_INPUT_DC_COUPLED (C enum), 670	TLV320_CHx_CFG0_INTYP_LINE (C macro), 658
tlv320_ch_input_agcdre_t.TLV320_INPUT_DC_COUPLED (C enumerator), 670	TLV320_CHx_CFG0_INTYP_MICRO (C macro), 658
tlv320_ch_input_t (C struct), 677	TLV320_CHx_CFG0_Z_10_KOHMS (C macro), 658
tlv320_ch_input_t.agcdre (C var), 677	TLV320_CHx_CFG0_Z_20_KOHMS (C macro), 658
tlv320_ch_input_t.coupling (C var), 677	TLV320_CHx_CFG0_Z_2_5_KOHMS (C macro), 658
tlv320_ch_input_t.src (C var), 677	TLV320_CHx_GAIN_CALIB_CODE_0_DB (C macro), 659
tlv320_ch_input_t.type (C var), 677	TLV320_CHx_GAIN_CALIB_DEFAULT (C macro), 659
tlv320_ch_input_t.z (C var), 677	TLV320_CHx_GAIN_CALIB_MAX (C macro), 659
tlv320_ch_input_z_t (C enum), 670	TLV320_CHx_GAIN_CALIB_MIN (C macro), 659
tlv320_ch_input_z_t.TLV320_INPUT_Z_10_KOHM (C enumerator), 670	TLV320_CHx_GAIN_CALIB_NB (C macro), 659
tlv320_ch_input_z_t.TLV320_INPUT_Z_10_KOHM (C enumerator), 670	TLV320_CHx_GAIN_CALIB_STEP (C macro), 659
tlv320_ch_input_z_t.TLV320_INPUT_Z_2_5_KOHM (C enumerator), 670	TLV320_CHx_GAIN_DEFAULT (C macro), 659
tlv320_ch_input_z_t.TLV320_INPUT_Z_2_5_KOHM (C enumerator), 670	TLV320_CHx_GAIN_MAX (C macro), 659
tlv320_ch_input_z_t.TLV320_INPUT_Z_20_KOHM (C enumerator), 670	TLV320_CHx_GAIN_MIN (C macro), 659
tlv320_ch_insrc_t (C enum), 669	TLV320_CHx_PHASE_CALIB_DEFAULT (C macro), 660
tlv320_ch_insrc_t.TLV320_INSRC_ANALOG_DIFF (C enumerator), 670	TLV320_CHx_PHASE_CALIB_MAX (C macro), 660
tlv320_ch_insrc_t.TLV320_INSRC_ANALOG_SINGLE (C enumerator), 670	TLV320_CHx_PHASE_CALIB_MIN (C macro), 660
tlv320_ch_insrc_t.TLV320_INSRC_DIGITAL (C enumerator), 670	TLV320_CHx_VOLUME_CODE_0_DB (C macro), 659
tlv320_ch_insrc_t.TLV320_INSRC_DIGITAL (C enum), 669	TLV320_CHx_VOLUME_DEFAULT (C macro), 659
tlv320_ch_intyp_t (C enum), 669	TLV320_CHx_VOLUME_MAX (C macro), 659
tlv320_ch_intyp_t.TLV320_INTYP_LINE (C enumerator), 669	TLV320_CHx_VOLUME_MIN (C macro), 659
tlv320_ch_intyp_t.TLV320_INTYP_MICRO (C enumerator), 669	TLV320_CLK_CFG_T (C struct), 680
tlv320_ch_output_line_t (C enum), 670	tlv320_clk_cfg_t.fs (C var), 680
tlv320_ch_output_line_t.TLV320_OUTPUT_PRIMARY (C enumerator), 670	tlv320_clk_cfg_t.mclk (C var), 680
tlv320_ch_output_line_t.TLV320_OUTPUT_SECONDARY (C enumerator), 670	tlv320_clk_cfg_t.mode (C var), 680
tlv320_ch_output_t (C struct), 677	tlv320_clk_mode_t (C struct), 678
tlv320_ch_output_t.line (C var), 678	tlv320_clk_mode_t.auto_clk (C var), 678
tlv320_ch_output_t.slot (C var), 678	tlv320_clk_mode_t.auto_pll (C var), 678
TLV320_CHANNEL_1 (C macro), 643	tlv320_clk_mode_t.clk_src (C var), 679
TLV320_CHANNEL_2 (C macro), 643	tlv320_clk_mode_t.master (C var), 678
TLV320_CHANNEL_3 (C macro), 643	tlv320_clk_src_t (C union), 683
TLV320_CHANNEL_4 (C macro), 644	tlv320_clk_src_t.dis_pll_slv_clk_src (C var), 684
TLV320_CHANNEL_5 (C macro), 644	tlv320_clk_src_t.mclk_freq_sel_mode (C var), 684
TLV320_CHANNEL_6 (C macro), 644	tlv320_clk_src_t.mclk_ratio_sel (C var), 684
TLV320_CHANNEL_7 (C macro), 644	tlv320_clk_src_t.raw (C var), 684
TLV320_CHANNEL_8 (C macro), 644	tlv320_clk_src_t.reserved (C var), 684
TLV320_CHANNEL_NB (C macro), 644	tlv320_clk_src_t.[anonymous] (C var), 684
TLV320_CHx_CFG0_AC_COUPLED (C macro), 658	TLV320_CLKSRC_AUDIOROOT_BCLK (C macro), 653
TLV320_CHx_CFG0_DC_COUPLED (C macro), 658	TLV320_CLKSRC_AUDIOROOT_MCLK (C macro), 653
	TLV320_CLKSRC_MCLK_FS_RATIO_1024 (C macro), 654
	TLV320_CLKSRC_MCLK_FS_RATIO_1536 (C macro), 654
	TLV320_CLKSRC_MCLK_FS_RATIO_2304 (C macro), 654
	TLV320_CLKSRC_MCLK_FS_RATIO_256 (C macro), 654

TLV320_CLKSRC_MCLK_FS_RATIO_384 ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH2_CFG3 (C enumerator), 665
TLV320_CLKSRC_MCLK_FS_RATIO_512 ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH2_CFG4 (C enumerator), 665
TLV320_CLKSRC_MCLK_FS_RATIO_64 ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH3_CFG0 (C enumerator), 665
TLV320_CLKSRC_MCLK_FS_RATIO_768 ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH3_CFG1 (C enumerator), 665
TLV320_CLKSRC_MCLK_MODE_FREQ_SEL ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH3_CFG2 (C enumerator), 666
TLV320_CLKSRC_MCLK_MODE_FSYNC_RATIO ( <i>C macro</i> ), 654	tlv320_config_register_t.TLV320_CH3_CFG3 (C enumerator), 666
tlv320_config_register_t ( <i>C enum</i> ), 663	tlv320_config_register_t.TLV320_CH3_CFG4 (C enumerator), 666
tlv320_config_register_t.TLV320阿根_CFG0 (C enumerator), 667	tlv320_config_register_t.TLV320_CH4_CFG0 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CFG0 (C enumerator), 663	tlv320_config_register_t.TLV320_CH4_CFG1 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CFG1 (C enumerator), 663	tlv320_config_register_t.TLV320_CH4_CFG2 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CFG2 (C enumerator), 663	tlv320_config_register_t.TLV320_CH4_CFG3 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH1 (C enumerator), 663	tlv320_config_register_t.TLV320_CH4_CFG4 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH2 (C enumerator), 663	tlv320_config_register_t.TLV320_CH5_CFG0 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH3 (C enumerator), 663	tlv320_config_register_t.TLV320_CH5_CFG1 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH4 (C enumerator), 663	tlv320_config_register_t.TLV320_CH5_CFG2 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH5 (C enumerator), 663	tlv320_config_register_t.TLV320_CH5_CFG3 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH6 (C enumerator), 663	tlv320_config_register_t.TLV320_CH5_CFG4 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH7 (C enumerator), 664	tlv320_config_register_t.TLV320_CH6_CFG0 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_CH8 (C enumerator), 664	tlv320_config_register_t.TLV320_CH6_CFG1 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_OUT_CH_EN tlv320_config_register_t.TLV320_CH6_CFG3 (C enumerator), 667	tlv320_config_register_t.TLV320_CH6_CFG2 (C enumerator), 666
tlv320_config_register_t.TLV320_AS1_STS (C enumerator), 664	tlv320_config_register_t.TLV320_CH6_CFG3 (C enumerator), 666
tlv320_config_register_t.TLV320_BIAS_CFG (C enumerator), 665	tlv320_config_register_t.TLV320_CH7_CFG0 (C enumerator), 666
tlv320_config_register_t.TLV320_CH1_CFG0 (C enumerator), 665	tlv320_config_register_t.TLV320_CH7_CFG1 (C enumerator), 666
tlv320_config_register_t.TLV320_CH1_CFG1 (C enumerator), 665	tlv320_config_register_t.TLV320_CH7_CFG2 (C enumerator), 666
tlv320_config_register_t.TLV320_CH1_CFG2 (C enumerator), 665	tlv320_config_register_t.TLV320_CH8_CFG0 (C enumerator), 667
tlv320_config_register_t.TLV320_CH1_CFG3 (C enumerator), 665	tlv320_config_register_t.TLV320_CH8_CFG1 (C enumerator), 667
tlv320_config_register_t.TLV320_CH1_CFG4 (C enumerator), 665	tlv320_config_register_t.TLV320_CH8_CFG2 (C enumerator), 667
tlv320_config_register_t.TLV320_CH2_CFG0 (C enumerator), 665	tlv320_config_register_t.TLV320_CLK_SRC (C enumerator), 664
tlv320_config_register_t.TLV320_CH2_CFG1 (C enumerator), 665	tlv320_config_register_t.TLV320_DEV_STS0 (C enumerator), 667
tlv320_config_register_t.TLV320_CH2_CFG2 (C enumerator), 665	tlv320_config_register_t.TLV320_DEV_STS1 (C enumerator), 667

tlv320_config_register_t.TLV320_DRE_CFG0 (C enumerator), 667	tlv320_fs_t (C struct), 679 tlv320_fs_t.bclk_ratio (C var), 679
tlv320_config_register_t.TLV320_DSP_CFG0 (C enumerator), 667	tlv320_fs_t.gate (C var), 679 tlv320_fs_t.mode (C var), 679
tlv320_config_register_t.TLV320_DSP_CFG1 (C enumerator), 667	tlv320_fs_t.rate (C var), 679 TLV320_GENERIC_ERROR (C macro), 643
tlv320_config_register_t.TLV320_GPI_CFG0 (C enumerator), 664	tlv320_gpi_cfg_t (C union), 684 tlv320_gpi_cfg_t.cfg_x (C var), 685
tlv320_config_register_t.TLV320_GPI_CFG1 (C enumerator), 664	tlv320_gpi_cfg_t.cfg_y (C var), 685 tlv320_gpi_cfg_t.raw (C var), 685
tlv320_config_register_t.TLV320_GPI_MON (C enumerator), 665	tlv320_gpi_cfg_t.reserved1 (C var), 685 tlv320_gpi_cfg_t.reserved2 (C var), 685
tlv320_config_register_t.TLV320_GPIO_CFG0 (C enumerator), 664	tlv320_gpi_cfg_t.[anonymous] (C var), 685 TLV320_GPIx_CFG_DISABLED (C macro), 656
tlv320_config_register_t.TLV320_GPIO_MON (C enumerator), 664	TLV320_GPIx_CFG_GPI (C macro), 656 TLV320_GPIx_CFG_MCLK (C macro), 656
tlv320_config_register_t.TLV320_GPO_CFG0 (C enumerator), 664	TLV320_GPIx_CFG_PDMIN1 (C macro), 657 TLV320_GPIx_CFG_PDMIN2 (C macro), 657
tlv320_config_register_t.TLV320_GPO_CFG1 (C enumerator), 664	TLV320_GPIx_CFG_PDMIN3 (C macro), 657 TLV320_GPIx_CFG_PDMIN4 (C macro), 657
tlv320_config_register_t.TLV320_GPO_CFG2 (C enumerator), 664	TLV320_GPIx_CFG_SDIN (C macro), 657 tlv320_gpo_cfg_t (C union), 684
tlv320_config_register_t.TLV320_GPO_CFG3 (C enumerator), 664	tlv320_gpo_cfg_t.cfg (C var), 684 tlv320_gpo_cfg_t.drv (C var), 684
tlv320_config_register_t.TLV320_GPO_VAL (C enumerator), 664	tlv320_gpo_cfg_t.raw (C var), 684 tlv320_gpo_cfg_t.reserved (C var), 684
tlv320_config_register_t.TLV320_I2C_CKSUM (C enumerator), 667	tlv320_gpo_cfg_t.[anonymous] (C var), 684 TLV320_GPOx_CFG_ASI (C macro), 655
tlv320_config_register_t.TLV320_IN_CH_EN (C enumerator), 667	TLV320_GPOx_CFG_DIO (C macro), 655 TLV320_GPOx_CFG_DISABLED (C macro), 655
tlv320_config_register_t.TLV320_INT_CFG (C enumerator), 665	TLV320_GPOx_CFG_GPO (C macro), 655 TLV320_GPOx_CFG_PDMCLK (C macro), 655
tlv320_config_register_t.TLV320_INT_LTCH0 (C enumerator), 665	TLV320_GPOx_DRV_ACTIVE_LOW_ACTIVE_HIGH (C macro), 656
tlv320_config_register_t.TLV320_INT_MASK0 (C enumerator), 665	TLV320_GPOx_DRV_ACTIVE_LOW_HI_Z (C macro), 656 TLV320_GPOx_DRV_ACTIVE_LOW_WEAK_HIGH (C macro), 656
tlv320_config_register_t.TLV320_MST_CFG0 (C enumerator), 664	TLV320_GPOx_DRV_HI_Z (C macro), 655
tlv320_config_register_t.TLV320_MST_CFG1 (C enumerator), 664	TLV320_GPOx_DRV_HI_Z_ACTIVE_HIGH (C macro), 656 TLV320_GPOx_DRV_WEAK_LOW_ACTIVE_HIGH (C macro), 656
tlv320_config_register_t.TLV320_PAGE_CFG (C enumerator), 663	tlv320_hw_mode_t (C enum), 662
tlv320_config_register_t.TLV320_PDMCLK_CFG (C enumerator), 664	tlv320_hw_mode_t.TLV320_ACTIVE (C enumerator), 663
tlv320_config_register_t.TLV320_PDMIN_CFG (C enumerator), 664	tlv320_hw_mode_t.TLV320_SHUTDOWN (C enumerator), 663
tlv320_config_register_t.TLV320_PWR_CFG (C enumerator), 667	TLV320_I2C_MAX_BAUDRATE (C macro), 643 TLV320_I2C_MSG_SIZE (C macro), 643
tlv320_config_register_t.TLV320_SHDN_CFG (C enumerator), 663	TLV320_I2C_READ_RX_SIZE (C macro), 643 TLV320_I2C_READ_TX_SIZE (C macro), 643
tlv320_config_register_t.TLV320_SLEEP_CFG (C enumerator), 663	TLV320_I2C_WRITE_FLAGS (C macro), 643 TLV320_IN_CH_EN_DISABLED (C macro), 661
tlv320_config_register_t.TLV320_SW_RESET (C enumerator), 663	TLV320_IN_CH_EN_ENABLED (C macro), 661 tlv320_in_ch_en_t (C union), 687

tlv320\_in\_ch\_en\_t.in\_ch1\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch2\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch3\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch4\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch5\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch6\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch7\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.in\_ch8\_en (*C var*), 688  
 tlv320\_in\_ch\_en\_t.raw (*C var*), 688  
 tlv320\_in\_ch\_en\_t.[anonymous] (*C var*), 688  
 TLV320\_IS\_INCLUDED (*C macro*), 642  
 TLV320\_IS\_VALID\_ADC\_FSCALE (*C macro*), 658  
 TLV320\_IS\_VALID\_ANALOG\_CH (*C macro*), 661  
 TLV320\_IS\_VALID\_ANALOG\_CHANNEL\_ID (*C macro*),  
     644  
 TLV320\_IS\_VALID\_AXI\_BCLK\_POL (*C macro*), 647  
 TLV320\_IS\_VALID\_AXI\_CHx\_OUTPUT (*C macro*), 649  
 TLV320\_IS\_VALID\_AXI\_FORMAT (*C macro*), 646  
 TLV320\_IS\_VALID\_AXI\_FSYNC\_POL (*C macro*), 647  
 TLV320\_IS\_VALID\_AXI\_MASTER\_SLAVE\_CFG  
     (*C macro*), 650  
 TLV320\_IS\_VALID\_AXI\_TX\_EDGE (*C macro*), 647  
 TLV320\_IS\_VALID\_AXI\_TX\_FILL (*C macro*), 647  
 TLV320\_IS\_VALID\_AXI\_TX\_KEEPER (*C macro*), 648  
 TLV320\_IS\_VALID\_AXI\_TX\_LSB\_CYCLE (*C macro*), 648  
 TLV320\_IS\_VALID\_AXI\_WLEN (*C macro*), 647  
 TLV320\_IS\_VALID\_AUDIOROOT\_CLK\_SRC (*C macro*),  
     654  
 TLV320\_IS\_VALID\_AUTO\_CLK (*C macro*), 650  
 TLV320\_IS\_VALID\_BCLKFSYNCGATE (*C macro*), 651  
 TLV320\_IS\_VALID\_CH\_GAIN (*C macro*), 660  
 TLV320\_IS\_VALID\_CH\_GAIN\_CALIB (*C macro*), 660  
 TLV320\_IS\_VALID\_CH\_PHASE\_CALIB (*C macro*), 660  
 TLV320\_IS\_VALID\_CH\_VOLUME (*C macro*), 660  
 TLV320\_IS\_VALID\_CHANNEL\_ID (*C macro*), 644  
 TLV320\_IS\_VALID\_CLK\_CFG (*C macro*), 655  
 TLV320\_IS\_VALID\_CLK\_FS\_CFG (*C macro*), 655  
 TLV320\_IS\_VALID\_CLK\_MCLK\_CFG (*C macro*), 655  
 TLV320\_IS\_VALID\_CLK\_MODE\_CFG (*C macro*), 655  
 TLV320\_IS\_VALID\_CONFIG\_REG (*C macro*), 644  
 TLV320\_IS\_VALID\_COUPLING (*C macro*), 661  
 TLV320\_IS\_VALID\_DRE\_AGC (*C macro*), 661  
 TLV320\_IS\_VALID\_FSBCLK\_RATIO (*C macro*), 653  
 TLV320\_IS\_VALID\_FSMODE (*C macro*), 651  
 TLV320\_IS\_VALID\_FSRATE (*C macro*), 653  
 TLV320\_IS\_VALID\_GPI\_CFG (*C macro*), 657  
 TLV320\_IS\_VALID\_GPO\_CFG (*C macro*), 656  
 TLV320\_IS\_VALID\_GPO\_DRV (*C macro*), 656  
 TLV320\_IS\_VALID\_HW\_MODE (*C macro*), 644  
 TLV320\_IS\_VALID\_INPUT\_Z (*C macro*), 661  
 TLV320\_IS\_VALID\_INSRC (*C macro*), 660  
 TLV320\_IS\_VALID\_INTYP (*C macro*), 660  
 TLV320\_IS\_VALID\_MCLK (*C macro*), 651  
 TLV320\_IS\_VALID\_MCLK\_FREQ\_MODE (*C macro*), 654  
 TLV320\_IS\_VALID\_MCLK\_RATIO\_SEL (*C macro*), 654  
 TLV320\_IS\_VALID\_MICBIAS (*C macro*), 658  
 TLV320\_IS\_VALID\_PLLAUTOMODE (*C macro*), 650  
 TLV320\_IS\_VALID\_SLEEP\_MODE (*C macro*), 645  
 TLV320\_IS\_VALID\_STATE (*C macro*), 644  
 TLV320\_IS\_VALID\_TDM\_SLOT\_ID (*C macro*), 649  
 TLV320\_IS\_VALID\_TX\_OFFSET (*C macro*), 648  
 tlv320\_mclk\_t (*C struct*), 679  
 tlv320\_mclk\_t.fs\_ratio (*C var*), 679  
 tlv320\_mclk\_t.mode (*C var*), 679  
 tlv320\_mclk\_t.select (*C var*), 680  
 tlv320\_micbias\_t (*C enum*), 670  
 tlv320\_micbias\_t.TLV320\_MICBIAS\_VREF (*C enum*  
     enumerator), 670  
 tlv320\_micbias\_t.TLV320\_MICBIAS\_VREF\_1\_096  
     (*C enumerator*), 670  
 tlv320\_micbias\_t.TLV320\_MICBIAS\_VREF\_AVDD (*C*  
     enumerator), 670  
 tlv320\_mst\_cfg0\_t (*C union*), 682  
 tlv320\_mst\_cfg0\_t.auto\_clk\_cfg (*C var*), 683  
 tlv320\_mst\_cfg0\_t.auto\_mode\_pll\_dis (*C var*),  
     683  
 tlv320\_mst\_cfg0\_t.bclk\_fsync\_gate (*C var*), 683  
 tlv320\_mst\_cfg0\_t.fs\_mode (*C var*), 683  
 tlv320\_mst\_cfg0\_t.mclk\_freq\_sel (*C var*), 683  
 tlv320\_mst\_cfg0\_t.mst\_slv\_cfg (*C var*), 683  
 tlv320\_mst\_cfg0\_t.raw (*C var*), 683  
 tlv320\_mst\_cfg0\_t.[anonymous] (*C var*), 683  
 tlv320\_mst\_cfg1\_t (*C union*), 683  
 tlv320\_mst\_cfg1\_t.fs\_bclk\_ratio (*C var*), 683  
 tlv320\_mst\_cfg1\_t.fs\_rate (*C var*), 683  
 tlv320\_mst\_cfg1\_t.raw (*C var*), 683  
 tlv320\_mst\_cfg1\_t.[anonymous] (*C var*), 683  
 TLV320\_MSTCFG0\_AUTOCLK\_DISABLE (*C macro*), 649  
 TLV320\_MSTCFG0\_AUTOCLK\_ENABLE (*C macro*), 649  
 TLV320\_MSTCFG0\_BCLKFSYNCGATE\_DISABLE  
     (*C macro*), 649  
 TLV320\_MSTCFG0\_BCLKFSYNCGATE\_ENABLE (*C macro*),  
     650  
 TLV320\_MSTCFG0\_FSMODE\_44\_1KHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_FSMODE\_48KHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MASTER (*C macro*), 649  
 TLV320\_MSTCFG0\_MCLK\_12\_288\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_12\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_13\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_16\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_19\_2\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_19\_68\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_24\_576\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_MCLK\_24\_MHZ (*C macro*), 650  
 TLV320\_MSTCFG0\_PLLAUTOMODE\_DISABLE (*C macro*),  
     649  
 TLV320\_MSTCFG0\_PLLAUTOMODE\_ENABLE (*C macro*),  
     649

TLV320_MSTCFG0_SLAVE (C macro), 649	tlv320_pwr_cfg_t.dyn_ch_pupd_en (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_1024 (C macro), 652	tlv320_pwr_cfg_t.dyn_maxch_sel (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_128 (C macro), 652	tlv320_pwr_cfg_t.micbias_pdz (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_16 (C macro), 652	tlv320_pwr_cfg_t.pll_pdz (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_192 (C macro), 652	tlv320_pwr_cfg_t.raw (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_2048 (C macro), 653	tlv320_pwr_cfg_t.reserved (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_24 (C macro), 652	tlv320_pwr_cfg_t.[anonymous] (C var), 689
TLV320_MSTCFG1_FSBCLK_RATIO_256 (C macro), 652	TLV320_READ_ERROR (C macro), 643
TLV320_MSTCFG1_FSBCLK_RATIO_32 (C macro), 652	TLV320_SLAVE_ADDRESS_1 (C macro), 642
TLV320_MSTCFG1_FSBCLK_RATIO_384 (C macro), 652	TLV320_SLAVE_ADDRESS_2 (C macro), 642
TLV320_MSTCFG1_FSBCLK_RATIO_48 (C macro), 652	TLV320_SLAVE_ADDRESS_3 (C macro), 642
TLV320_MSTCFG1_FSBCLK_RATIO_512 (C macro), 652	TLV320_SLAVE_ADDRESS_4 (C macro), 642
TLV320_MSTCFG1_FSBCLK_RATIO_64 (C macro), 652	TLV320_SLAVEADDRESS_ERROR (C macro), 643
TLV320_MSTCFG1_FSBCLK_RATIO_96 (C macro), 652	TLV320_SLEEP_CFG_AREG_SELECT_EXTERNAL (C macro), 645
TLV320_MSTCFG1_FSRATE_14_7_OR_16_KHZ (C macro), 651	TLV320_SLEEP_CFG_AREG_SELECT_INTERNAL (C macro), 645
TLV320_MSTCFG1_FSRATE_176_4_OR_192_KHZ (C macro), 652	TLV320_SLEEP_CFG_I2C_BRDCAST_EN_DISABLED (C macro), 645
TLV320_MSTCFG1_FSRATE_22_05_OR_24_KHZ (C macro), 651	TLV320_SLEEP_CFG_I2C_BRDCAST_EN_ENABLED (C macro), 645
TLV320_MSTCFG1_FSRATE_29_4_OR_32_KHZ (C macro), 651	TLV320_SLEEP_CFG_SLEEP_ENZ_DISABLED (C macro), 645
TLV320_MSTCFG1_FSRATE_352_8_OR_384_KHZ (C macro), 652	TLV320_SLEEP_CFG_SLEEP_ENZ_ENABLED (C macro), 645
TLV320_MSTCFG1_FSRATE_44_1_OR_48_KHZ (C macro), 652	tlv320_sleep_cfg_t (C union), 680
TLV320_MSTCFG1_FSRATE_705_6_OR_768_KHZ (C macro), 652	tlv320_sleep_cfg_t.areg_select (C var), 681
TLV320_MSTCFG1_FSRATE_7_35_OR_8_KHZ (C macro), 651	tlv320_sleep_cfg_t.i2c_brdcast_en (C var), 681
TLV320_MSTCFG1_FSRATE_88_2_OR_96_KHZ (C macro), 652	tlv320_sleep_cfg_t.raw (C var), 681
TLV320_NO_ERROR (C macro), 643	tlv320_sleep_cfg_t.reserved_1 (C var), 681
TLV320_PAGE_COEF_BIQUAD_1_6 (C macro), 642	tlv320_sleep_cfg_t.reserved_5_6 (C var), 681
TLV320_PAGE_COEF_BIQUAD_7_12 (C macro), 642	tlv320_sleep_cfg_t.sleep_enz (C var), 681
TLV320_PAGE_COEF_MIXER_1_4 (C macro), 643	tlv320_sleep_cfg_t.vref_qchg (C var), 681
TLV320_PAGE_CONFIG_REG (C macro), 642	tlv320_sleep_cfg_t.[anonymous] (C var), 681
TLV320_PWR_CFG_ADC_POWER_DOWN (C macro), 662	TLV320_SLEEP_CFG_VREF_QCHG_100_MS (C macro), 645
TLV320_PWR_CFG_ADC_POWER_UP (C macro), 662	TLV320_SLEEP_CFG_VREF_QCHG_10_MS (C macro), 645
TLV320_PWR_CFG_DYN_CH_PUPD_DISABLED (C macro), 662	TLV320_SLEEP_CFG_VREF_QCHG_3_5_MS (C macro), 645
TLV320_PWR_CFG_DYN_CH_PUPD_ENABLED (C macro), 662	TLV320_SLEEP_CFG_VREF_QCHG_50_MS (C macro), 645
TLV320_PWR_CFG_DYN_MAXCH_SEL_1_2 (C macro), 662	TLV320_SLEEPMODE_SLEEP_DELAY (C macro), 643
TLV320_PWR_CFG_DYN_MAXCH_SEL_1_4 (C macro), 662	TLV320_SLEEPMODE_WAKEUP_DELAY (C macro), 643
TLV320_PWR_CFG_DYN_MAXCH_SEL_1_6 (C macro), 662	tlv320_state_t (C enum), 662
TLV320_PWR_CFG_DYN_MAXCH_SEL_1_8 (C macro), 662	tlv320_state_t.TLV320_DISABLE (C enumerator), 662
TLV320_PWR_CFG_MICBIAS_POWER_DOWN (C macro), 661	tlv320_state_t.TLV320_ENABLE (C enumerator), 662
TLV320_PWR_CFG_MICBIAS_POWER_UP (C macro), 662	TLV320_SW_RESET_RESET (C macro), 645
TLV320_PWR_CFG_PLL_POWER_DOWN (C macro), 662	tlv320_sw_reset_t (C union), 680
TLV320_PWR_CFG_PLL_POWER_UP (C macro), 662	tlv320_sw_reset_t.raw (C var), 680
tlv320_pwr_cfg_t (C union), 689	tlv320_sw_reset_t.reserved (C var), 680
tlv320_pwr_cfg_t.adc_pdz (C var), 689	tlv320_sw_reset_t.sw_reset (C var), 680
	tlv320_sw_reset_t.[anonymous] (C var), 680
	TLV320_TDM_SLOT_DEFAULT (C macro), 649
	TLV320_TDM_SLOT_MAX (C macro), 649

TLV320\_TDM\_SLOT\_MIN (*C macro*), 649  
`tlv320_tx_keeper_t` (*C enum*), 669  
`tlv320_tx_keeper_t.TLV320_TX_KEEPER_1_5_CYCLES` (*C enumerator*), 669  
`tlv320_tx_keeper_t.TLV320_TX_KEEPER_1CYCLE` (*C enumerator*), 669  
`tlv320_tx_keeper_t.TLV320_TX_KEEPER_DISABLED` (*C enumerator*), 669  
`tlv320_tx_keeper_t.TLV320_TX_KEEPER_EN` (*C enumerator*), 669  
`tlv320_tx_lsb_t` (*C enum*), 669  
`tlv320_tx_lsb_t.TLV320_TX_LSB_FULLCYCLE` (*C enumerator*), 669  
`tlv320_tx_lsb_t.TLV320_TX_LSB_HALFCYCLE` (*C enumerator*), 669  
`TLV320_WRITE_ERROR` (*C macro*), 643  
`total_memory_usage` (*nntool.api.NNGraph property*), 423  
`total_ops` (*nntool.api.NNGraph property*), 423  
`trace_add()` (*gv.gvsoc\_control.Proxy method*), 334  
`trace_add()` (*gvsoc\_control.Proxy method*), 582  
`trace_level()` (*gv.gvsoc\_control.Proxy method*), 334  
`trace_level()` (*gvsoc\_control.Proxy method*), 582  
`trace_remove()` (*gv.gvsoc\_control.Proxy method*), 334  
`trace_remove()` (*gvsoc\_control.Proxy method*), 582  
`tx()` (*gv.gvsoc\_control.Testbench\_uart method*), 340  
`tx()` (*gvsoc\_control.Testbench\_uart method*), 588

**U**

`uart_get()` (*gv.gvsoc\_control.Testbench method*), 336  
`uart_get()` (*gvsoc\_control.Testbench method*), 584  
`use_compressed()` (*nntool.api.NNGraph method*), 423

**V**

`validate()` (*nntool.api.validation.ValidateBase method*), 435  
`ValidateBase` (*class in nntool.api.validation*), 435  
`validated` (*nntool.api.validation.ValidationResultBase property*), 435  
`ValidateFromClass` (*class in nntool.api.validation*), 435  
`ValidateFromJSON` (*class in nntool.api.validation*), 435  
`ValidateFromName` (*class in nntool.api.validation*), 435  
`ValidateFromVWIInstances` (*class in nntool.api.validation*), 435  
`ValidationResultBase` (*class in nntool.api.validation*), 435  
`value` (*gapylib.flash.FlashSectionProperty attribute*), 382  
`value` (*gapylib.utils.CStructField attribute*), 385

**W**

`wait_running()` (*gv.gvsoc\_control.Proxy method*), 334

`wait_running()` (*gvsoc\_control.Proxy method*), 582  
`wait_stop()` (*gv.gvsoc\_control.Proxy method*), 334  
`wait_stop()` (*gvsoc\_control.Proxy method*), 582  
`write_graph_state()` (*nntool.api.NNGraph method*), 423

**X**

`Xip` (*class in gapylib.chips.gap.rom\_v2*), 395