

PicoLibSDK

version 2.08

Alternative extended C/C++ SDK library for Raspberry Pico, RP2040 and RP2350

SDK Programmer's Guide



(c) Miroslav Nemecek

May 2025

Panda38@seznam.cz , hardyplotter2@gmail.com

<https://github.com/Panda381/PicoLibSDK>

https://www.breatharian.eu/hw/picolibsdk/index_en.html

<https://github.com/pajenicko/picopad>

<https://picopad.eu/en/>

1. Introduction

1.1. Contents

1. Introduction.....	2
1.1. Contents.....	2
1.2. About.....	7
1.3. License.....	7
1.4. Building.....	8
1.5. Configuration.....	11
1.6. How to create new project.....	12
1.7. Select or create new target device.....	13
1.8. Conventions.....	14
1.9. Directory structure.....	17
1.10. Boot3 loader.....	19
1.11. Booting process.....	21
1.12. FAR File Manager.....	23
1.13. Original-SDK interface.....	25
1.14. History of versions.....	25
2. SDK Software Development Kit.....	27
2.1. ADC - Analogue to Digital Converter.....	27
2.2. Boot ROM.....	34
2.3. Clocks.....	40
2.4. CPU Control.....	46
2.5. Divider - Integer Division And Multiplication.....	52
2.6. DMA - Direct Memory Access.....	56
2.7. Double - Double-Floating-Point.....	72
2.8. FIFO - Inter-Core FIFO, Mailboxes.....	81
2.9. Flash Memory.....	83
2.10. Float - Single-Floating-Point.....	84
2.11. GPIO - General Purpose Input/Output Pins.....	92
2.12. HSTX - High-speed Serial Transmit.....	107
2.13. I2C - Inter-Integrated Circuit.....	111
2.14. Interpolator.....	115

2.15. IRQ - Interrupt Request.....	125
2.16. Multicore.....	133
2.17. PIO - Programmable Input/Output Block.....	135
2.18. PLL - Phase-Locked Loop.....	159
2.19. PowMan - Power Manager.....	162
2.20. PWM - Pulse Width Modulation.....	165
2.21. QMI - QSPI memory interface.....	174
2.22. QSPI - QSPI Flash Pins.....	177
2.23. Reset - Power-On State Machine.....	192
2.24. ROSC - Ring Oscillator.....	197
2.25. RTC - Real-Time Clock.....	200
2.26. SHA-256 - SHA-256 accelerator.....	202
2.27. SPI - Serial Peripheral Interface.....	204
2.28. SpinLock.....	215
2.29. SSI - synchronous serial interface.....	218
2.30. SysTick - SysTick System Timer.....	219
2.31. Ticks - Tick Generator.....	221
2.32. Timer - Precise Timer with Alarm.....	222
2.33. TMDS - TMDS Encoder.....	228
2.34. TRNG - True Random Number Generator.....	231
2.35. UART - Asynchronous Serial Port.....	234
2.36. Voltage Regulator.....	250
2.37. Watchdog Timer.....	252
2.38. XIP - Flash control.....	254
2.39. XOSC - Crystal Oscillator.....	255
3. Tool Library.....	257
3.1. Alarm.....	257
3.2. Calendar - Short 32-bit Calendar.....	259
3.3. Calendar 64-bit - Long 64-bit Calendar.....	263
3.4. Canvas - Drawing Surface.....	270
3.5. Color - RGBA Color Vector.....	276
3.6. Configuration of Device.....	282
3.7. CRC - Cyclic Redundancy Check.....	287
3.8. DecNum - Decode Numbers.....	311
3.9. Draw - Drawing to Display Frame Buffer.....	312
3.10. DrawCan - Drawing to Canvas.....	333
3.11. Emu - Emulators.....	334

3.12. EscPkt - Escape Packet Protocol.....	348
3.13. Event - Event Ring Buffer.....	350
3.14. FAT - FAT File System.....	352
3.15. FileSel - File Selection.....	364
3.16. List - Doubly Linked List.....	367
3.17. Malloc - Memory Allocator.....	372
3.18. Mat2D - 2D Transformation Matrix.....	377
3.19. MiniRing - Mini-Ring Buffer.....	381
3.20. MP3 - MP3 decoder and player.....	384
3.21. Print - Formatted Print, printf.....	393
3.22. PWM Sound Output.....	400
3.23. Rand - Random Number Generator.....	405
3.24. Rect - Rectangle.....	411
3.25. Ring Buffer.....	414
3.26. RingRX - Ring Buffer with DMA in Receiver Mode.....	420
3.27. RingTX - Ring Buffer with DMA in Transmitt Mode.....	424
3.28. SD Card.....	429
3.29. Stream - Data Stream.....	431
3.30. Text Strings.....	434
3.31. TextList - List of Text Strings.....	461
3.32. TPrint - Print to Attribute Text Buffer.....	467
3.33. Tree List.....	471
3.34. Video Player.....	473
4. USB Library.....	476
4.1. USB Mini-Port.....	476
4.2. USB CDC Device.....	483
4.3. USB CDC Host.....	486
4.4. USB HID Device.....	490
4.5. USB HID Host.....	495
4.6. USB Device.....	498
4.7. USB Host.....	503
4.8. USB Physical.....	509
5. Big Integers.....	513
5.1. Big Integers and Bernoulli.....	513
6. Real Numbers.....	520
6.1. Formats of Real Numbers.....	521
6.2. Constants.....	530

6.3. Get Setup.....	531
6.4. Flags and State Manipulation.....	533
6.5. Set/Get Number.....	535
6.6. Arithmetics Operations.....	547
6.7. Functions.....	554
6.8. Ln Function.....	562
6.9. Exp Function.....	566
6.10. Sqrt Function.....	569
6.11. Sin Function.....	573
6.12. Cos Function.....	576
6.13. SinCos Function.....	579
6.14. Tan Function.....	581
6.15. CoTan Function.....	584
6.16. ASin Function.....	587
6.17. ACos Function.....	591
6.18. ATan Function.....	592
6.19. ACoTan Function.....	594
6.20. Polar Coordinates.....	595
6.21. Trigonometric Test Function.....	596
6.22. Hyperbolic Functions.....	598
6.23. Random Numbers.....	605
6.24. Chebyshev Approximation.....	606
6.25. Bernoulli Numbers.....	609
6.26. Factorial.....	609
6.27. From/To Text.....	612
7. Display Drivers.....	614
7.1. ST7789 TFT.....	614
7.2. Mini-VGA.....	617
7.3. DVI (HDMI).....	621
7.4. DVIVGA (HDMI with VGA).....	624
7.5. DispHSTX.....	627
7.6. DispHSTXMini.....	628
8. Target Devices.....	629
8.1. PicoPad.....	629
8.2. PicoPadHSTX.....	635
8.3. PicoPadVGA.....	637
8.4. Picoino.....	643

8.5. PicinoMini.....	649
8.6. DemoVGA.....	654
8.7. Picotron.....	659
8.8. Raspberry Pico.....	665
9. Sample Applications.....	667
10. Instruction Set.....	688

1.2. About

PicoLibSDK is an alternative extended C/C++ SDK library for the Raspberry Pico or Pico 2 module with the RP2040 or RP2350 processor (in the ARM or RISC-V mode), but it can also be used for other modules which use these processors. Compared to the original SDK library, officially provided with the Raspberry Pico, the PicoLibSDK library tries to extend the functionality of the original library and especially to make the SDK library easier to use. But most of functions of original SDK are provided here as well, for backwards compatibility. What you can find in the PicoLibSDK library:

Boot Loader: Boot loader allowing to select and run UF2 programs from SD card.

SDK hardware control: ADC, boot ROM, clocks control, CPU control, hardware divider, DMA, doorbells, double and float arithmetics, FIFO mailboxes, flash programming, GPIO, HSTX, I2C, hardware interpolator, IRQ, multicore, PIO, PLL, PWM, QSPI, reset and power control, ROSC, RTC, SHA256, SPI, spinlocks, SysTick, alarm timer, TMDS, TRNG, watchdog, XOSC. The RP2350 processor can be used in ARM or RISC-V mode.

Tool library: alarm, 32-bit Unix calendar, long 64-bit astronomic calendar, canvas drawing, RGBA color vector, CRC check with DMA support, decode numbers, emulators, escape packet protocol, event ring buffer, FAT file system, doubly linked list, memory allocator, 2D transformation matrix, mini-ring buffer, formatted print, PWM sound output with ADPCM, random generator, rectangle, ring buffer, DMA ring buffer, SD card, streams, text strings, text list, text print, tree list, VGA drawing, video player, MP3 player.

USB library: multiplayer mini-port, CDC device and host - serial communication, HID device and host - including external keyboard and mouse.

Big intergers: calculations with large integers, calculation of Bernoulli numbers.

Real numbers: calculations with floating-point numbers with optional precision up to 3690 digits and 30-bit exponent. Scientific functions with optional calculation method - Ln, Exp, Sqrt, Sin, Cos, Tan, arcus, hyperbolic functions and many more. Linear factorials with accurate and fast calculation.

Display drivers: Prepared support of TFT and VGA display with resolution 320x240 up to 800x600, with 4, 8, 15 or 16 bits output.

Devices: Support of Picoino/PicoinoMini with 8-bit QVGA display, DemoVGA with 16-bit VGA display, Picotron with 4-bit VGA display and PicoPad with 16-bit TFT display. Some samples are also prepared for the basic Raspberry Pico without additional hardware.

1.3. License

The library source code is, with a few exceptions, completely free to use for any purpose, including commercial use. It is possible to use and modify all or parts of the library source code without restriction. Some libraries (mainly single- and double-floating-point mathematics) are mostly the copyrighted work of Raspberry Pi and are therefore subject to the licensing terms of the original authors.

1.4. Building

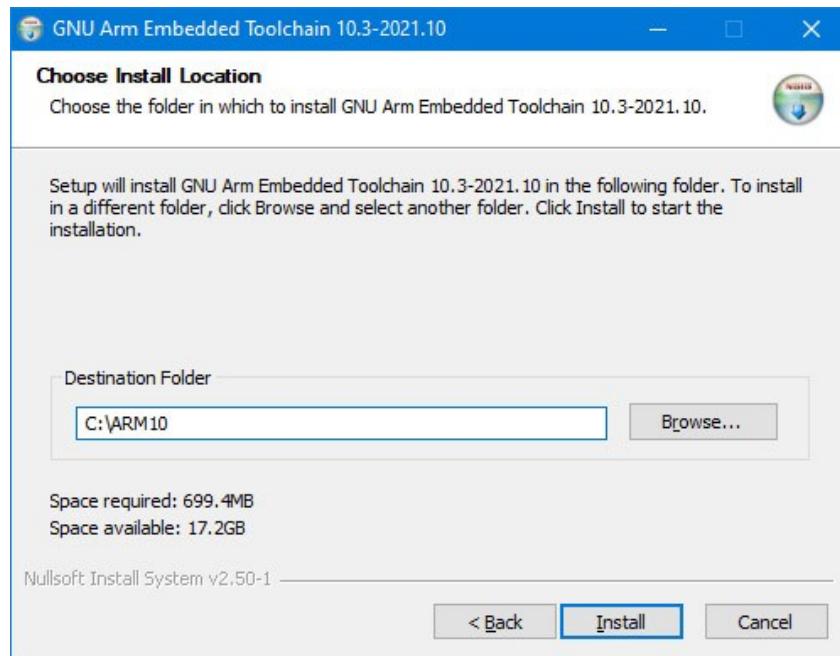
The compilation of the PicoLibSDK library and its sample examples is mainly prepared for easy use on **Windows**, but it is also ready for Linux compilation. It does not require any additional programs and utilities, only the installation of the GCC ARM compiler.

GCC ARM Installation in Windows

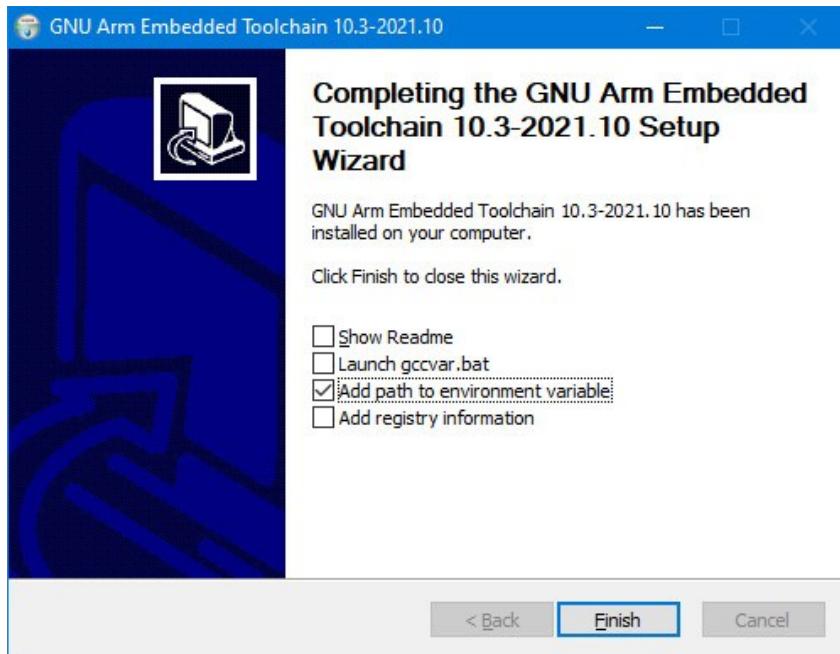
Download the GCC ARM compiler from <https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads>. You can use Windows Installer. To compile RP2350 (Pico 2), you will need GCC-ARM compiler version 13.3 or more. To compile RP2350 in RISC-V mode, you will need Hazard3 RISC-V compiler. You can use GCC-RISCV32 compiler version 13.2 or more: <https://www.embecosm.com/resources/tool-chain-downloads/#corev>. You can install both compilers in the same directory, e.g. C:\ARM.

In the last stage of the installation, enable the "**Add path to environment variable**" option. This will ensure that the compiler files will be found. Note: If you are using a shell program such as FAR, you must exit and restart the program to update the PATH system variable.

Choose Install Location:



Select “Add path to environment variable”:



How to compile in Windows

The compilation of the PicoLibSDK library was prepared primarily for ease of use in cooperation with the **FAR** file manager or another file manager that supports the console window. For each sample application, there are several BAT command files that can be easily run from the FAR by pressing a letter and Enter. The compilation is done using **c.bat** (in FAR press letter C and Enter). The **d.bat** file is used to clean up the compilation and delete generated intermediate files. Use **e.bat** to send the compiled program to the Raspberry Pico. The **a.bat** file performs all 3 operations at once - cleaning up old compilation, compile the program and send it to the Pico.

The target device for which the compilation is performed is passed as a parameter via the command line. A default target device is prepared in the c.bat file in case you do not specify it as a parameter. In the Pico folder, the default compilation is for the target device 'pico' (the Raspberry Pico module itself without any added hardware), in the DemoVGA folder the compilation is for 'demovga', in the Picoino folder the compilation is for 'picoino10' and in the PicoPad folder the default compilation is for 'picopad20' (or select 'picopad10' or 'picopad20riscv' batch file).

If the target device is not passed to the compilation files, it is compiled for the default device set in the _setup.bat file. This also applies if you compile all programs using c_all.bat. You can change the default target device by moving the :default label in the _setup.bat file.

Sending the program to Pico using **e.bat** is done by copying the file to the Pico access disk. This assumes that the Pico access disk is labeled R: (like Raspberry). If your disk has a different name, correct the disk name in the **_e1.bat** file (in the base folder of the PicoLibSDK library) or rename the disk in the system disk management (This Computer - Service - Disk Management - Change Drive Letter and Paths).

Cleaning up the compilation can be useful not only to clean up the application directory, but also during application development. Normally, all files are compiled during the first compilation. When recompiling, only the changed files are compiled. However, this only applies to *.c and *.cpp and *.s files, not *.h header files. If you make a change to a *.h file,

you may need to either modify the C file that uses that *.h file, or clean up the compilation with **d.bat** before recompiling to do a complete compilation.

The Windows compilation does not use CMake to optimize the compilation, all files are always compiled. This can be tedious, however - the programmer usually spends most of the time thinking and writing the program, and only a small part on compiling. Therefore, it may be preferable to handle the compilation more easily, rather than saving some compilation time. The final code is not affected - the linker will only use the parts of the program that are really needed in the final code. If you want to use some part of the library, you usually don't need to activate its compilation (with exceptions such as USB or real numbers), you just use necessary functions in the program.

How to compile in Linux

The PicoLibSDK library contains basic script files for program compilation. After downloading the PicoLibSDK library to the Linux, it is necessary to set the EXEC attribute of the execution scripts - the scripts are exported from the Windows environment and therefore do not have the EXEC attribute set. You can use the following command to set the script attribute in a batch:

```
find . -type f -name *.sh -exec chmod +x {} \;
```

To compile, you need at least the elf2uf2 program, which you can get from the original SDK library. Place the program in the _tools folder. Second, you will need the LoaderCrc program, which is used to calculate the checksum of applications run by the boot3 loader. You can find the source code for this program in the _tools/PicoPadLoaderCrc folder where you can compile it.

As a working environment, it is a good idea to use Midnight Commander, with which you can edit programs and immediately compile and run them using prepared scripts.

To compile the program, run the **c.sh** script. Compilation is done implicitly for the target device PicoPad 1.0. If you want to compile for a different device, you can either pass the device name as a parameter via the command line, or edit the parameters for the default target device in the **_setup.ch** script, where you can also find the names of possible target devices.

The **d.sh** script is used to clean up the compilation and delete generated intermediate files. Use **e.sh** to send the compiled program to the Raspberry Pico - to use it, edit mount point in **_e1.sh**. The **a.sh** file performs all 3 operations at once - cleaning up old compilation, compile the program and send it to the Pico. You will also need to delete the compiled files using **d.sh** if you change the contents of the *.h header file. Without clearing the compilation, the recompilation of dependent *.c files would not take place (dependency method is not used here).

The compilation does not use CMake to optimize the compilation, all files are always compiled. The final code is not affected - the linker will only use the parts of the program that are really needed in the final code. This method is used to make the compilation easier to use. If you want to use some part of the library, you usually don't need to activate its compilation (with exceptions such as USB or real numbers), you just use necessary functions in the program.

1.5. Configuration

To configure application compilation, switches defined by **#define** C preprocessor commands are used. In most cases, there is no need to change the compilation configuration - it is predefined optimally for the target device for which the application is being compiled. Changing the configuration may be necessary in cases where a module is disabled and needs to be activated. This is for example activating the STDIO output on a USB port that is disabled by default.

The default configuration file is **config.h**, which is located in the base folder of the application. For example, by entering the command:

```
#define USE_USB_STDIO 1 // use USB stdio (UsbPrint function)
```

activates the USB STDIO output (the output from printf() will go to the COM console on the host system).

Most configuration switches use a value of **1** to turn the module on, and a value of **0** to turn it off.

If a configuration switch is not specified at all in the config.h file, its default setting is used. In the first step, the default setting is used in the **_config.h** file, which is found in the home folder of the files for the target device (e.g. **_devices\picopad**).

If the configuration switch is not defined there either, the default switch setting is used, which is specified in the **config_def.h** file located in the base folder of the PicoLibSDK library.

Some configuration switches can be automatically reconfigured. This applies when a module requires another module - in that case, the other module is activated even if the user has requested that it be deactivated. An example would be the activation of USB device. If following command is given:

```
#define USE_USB_DEV_CDC 1
```

to activate the USB communication UART interface, the USE_USB_DEV (USB device) and USE_USB (USB interface) switches are also automatically activated.

1.6. How to create new project

To start a new project, the easiest way is to go to the NEW folder and copy the NEW project folder under a new name (e.g. TEST). In the new folder, open the **c.bat**, **d.bat** and **e.bat** files and change the settings of the TARGET system variable to the name of the new project (e.g. "set TARGET=TEST"). If you want the compiled project to be uploaded to another folder in the target SD card image, change the setting of the GRPDIR parameter in c.bat - this is the name of the project group folder.

Do not use the name LOADER for the project - this name is used to distinguish the boot loader during compilation.

Use only names with a maximum length of 8 characters (uppercase), as the boot3 loader only supports short names 8.3 of the FAT file system.

You can add configuration switches to the file **config.h** in your new project. For example, if you want the printf output to run to a USB virtual serial port, set the parameter:

```
#define USE_USB_STDIO      1      // use USB stdio (UsbPrint function)
```

You can add your *.h header files to the **include.h** file. These files will be visible to all your source code. Add include files at the end of the file, after the main include , e.g.:

```
#include "src/main.h"          // main code
#include "src/game.h"          // game
```

You can add your source files to the **Makefile** file. Add ASM files to the group ASRC, add C source files to the group CSRC and add C++ source files to the group SRC. For example add source of image:

```
SRC += src/main.cpp
SRC += img/logo.cpp
```

Remember that if you have some functions in a C or S (ASM) file and you want to call them from a CPP file, you have to mark the functions in the *.h header file with the **extern "C"** tag so that their names are visible in the CPP file. For example:

```
#ifdef __cplusplus
extern "C" {
#endif
```

```
void MyFunction();
```

```
#ifdef __cplusplus
}
#endif
```

1.7. Select or create new target device

PicoLibSDK can be compiled with miscellaneous target devices. In the current version of the library, programs are compiled implicitly for PicoPad version 1.0. To change the default target device, or to create a new target device, open the `_setup.bat` file in the base PicoLibSDK folder. You will see several settings there. Move the "`:default`" label in front of another parameter set to change the default target device.

You create a new device by creating a new parameter group and changing the parameters in the meaning: `DEVICE` is the name of the target device, including the version. It is used in the source code to distinguish device variants. `DEVCLASS` is the class of the target device - used primarily in `Makefile` to distinguish the compilation type. `DEVDIR` is the path to the SD card image. It is used when copying a compiled file to the target SD card.

You can also select the target device to be compiled in the `c.bat` file. In multiple compilation, the name of the target device is passed here as the `%1` parameter, which is passed forward on the last row of the file as the parameter for the `_c1.bat` file. You can prepend another row before the last row, in which you set the name of the target device as a parameter, according to the list of names given in the `_setup.bat` file. For example, if you want to compile for `picopad10` (you can leave the last row, the row added before it will take precedence):

```
..\_c1.bat picopad10  
..\_c1.bat %1
```

The following text refers only to the creation of a new device.

As you can see in the `Makefile.inc` in the base folder of the library, the `_makefile.inc` file from the device folder is included in the `Makefile`, named according to the `DEVCLASS` parameter. So, for a new device, you must create a folder in the `_devices` folder named by `DEVCLASS`, and create the `_makefile.inc` file there. This will contain a list of source files and possibly also `DEFINE` parameters. First of all, you need to add a `DEFINE` specifying that it is compiled for this device. For example:

```
DEFINE += -D USE_PICOINO=1
```

Add the include file `_config.h` from the device folder to the `config_def.h` file, conditioned on the switch for the device. It will contain additional configuration parameter settings. For example:

```
#if USE_PICOINO // use Picoino device configuration  
#include "_devices/picoino/_config.h"  
#endif
```

In the `includes.h` file in the base library folder, add a call of the `_include.h` file from the device folder, conditioned on the switch for the device. It will contain the included header files for the device. For example:

```
#if USE_PICOINO // use Picoino device configuration  
#include "_devices/picoino/_include.h"  
#endif
```

1.8. Conventions

The library source files use the Windows style notation for function names and global variables - the initial letters of words are uppercase, other letters are lowercase. Constants #define are uppercase. Local variables are lowercase. Types of structures start with the letter 's'.

Structure definitions of hardware are not used to access the peripheral registers, but direct access via register address definitions with #define is used. Similarly, the library usually does not use predefined constants to manipulate registers, but uses directly numeric values according to the datasheet. These practices are used for simplicity of access. The usual conventions using definitions require searching constantly between the datasheet and the definition files, and so rather than making the job easier, they make the job more complicated. This simplified approach only requires the use of the datasheet. This does not affect the SDK library user, as most operations are covered by functions, without having to work directly with registers.

Base Types

s8 signed 8-bit integer

u8 unsigned 8-bit integer

s16 signed 16-bit integer

u16 unsigned 16-bit integer

s32 signed 32-bit integer

u32 unsigned 32-bit integer

s64 signed 64-bit integer

u64 unsigned 64-bit integer

int signed integer

uint unsigned integer

Bool logical Boolean type with 8-bit size

True logical Boolean true value

False logical Boolean false value

NULL invalid pointer

Note: 'char' can be signed or unsigned. Better to use s8 or u8 if you need to be sure.

Note: Why Bool type of boolean variable is used? Two types of boolean variables are commonly used. In Windows notation, a boolean variable is called BOOL with the values TRUE and FALSE. It is based on the C language, where the boolean value FALSE is considered to be the zero value of an integer number, and TRUE is considered to be a non-

zero value. This type of logical variable requires a variable of size int, that is, typically 32 bits. In C++, the bool logical variable was introduced with the values true and false. It is a bit variable with a typical size of 1 byte and values of 0 and 1. The bool variable is more memory efficient than the BOOL type, but it is more demanding for the processor - it requires permanent conversion of the result of operations to the value 0 or 1. The Bool type variable is an optimal compromise between the two methods. It uses a variable of size 1 byte, treats a value of False as a zero value and a non-zero value as True. This method is memory-efficient while being fast and efficient for the processor.

Useful Constants and Macros

<code>count_of(a)</code>	count of array entries
<code>INLINE</code>	inline function
<code>NOINLINE</code>	no-inline function
<code>ALIGNED</code>	array aligned to 4-bytes
<code>PACKED</code>	packed structure with unaligned entries
<code>NOFLASH(fnc)</code>	time critical function placed into RAM
<code>ENDIAN16(k)</code>	change 16-bit endian (little endian <-> big endian)
<code>ABS(val)</code>	absolute value
<code>MAX(val1,val2)</code>	maximal value
<code>MIN(val1,val2)</code>	minimal value
<code>B0..B31</code>	bit 0..bit 31 (= 0x00000001 .. 0x80000000)
<code>BIT(pos)</code>	bit at given position
<code>BIGINT</code>	big integer value (=0x40000000)
<code>KHZ</code>	kHz multiply (= 1000)
<code>MHZ</code>	MHz multiply (= 1000000)
<code>NOCHAR</code>	no character from keyboard (= 0)
<code>NOKEY</code>	no key from keyboard (= 0)

Control characters

symbolic name / HEX code / text code / ASCII name / Control-letter / usage

<code>CH_NUL</code>	0x00	'\0'	NUL	[^] @	no character, end of text
<code>CH_ALL</code>	0x01	'\1'	SOH	[^] A	select [A]ll
<code>CH_BLOCK</code>	0x02	'\2'	STX	[^] B	mark [B]lock
<code>CH_STX</code>	=	<code>CH_BLOCK</code>	(start of packet)		
<code>CH_COPY</code>	0x03	'\3'	ETX	[^] C	[C]opy block, copy file
<code>CH_ETX</code>	=	<code>CH_COPY</code>	(end of packet)		
<code>CH_END</code>	0x04	'\4'	EOT	[^] D	en[D] of row, end of files
<code>CH_MOVE</code>	0x05	'\5'	ENQ	[^] E	rename files, mov[E] block
<code>CH_FIND</code>	0x06	'\6'	ACK	[^] F	[F]ind

CH_NEXT	0x07	'\a'	BEL	^G	[G]o next, repeat find
CH_BS	0x08	'\b'	BS	^H	backspace
CH_TAB	0x09	'\t'	HT	^I	tabulator
CH_LF	0x0A	'\n'	LF	^J	line feed
CH_PGUP	0x0B	'\v'	VT	^K	page up
CH_PGDN	0x0C	'\f'	FF	^L	page down
CH_FF	= CH_PGDN				(form feed, new page)
CH_CR	0x0D	'\r'	CR	^M	enter, next row, run file
CH_NEW	0x0E	'\16'	SO	^N	[N]ew file
CH_OPEN	0x0F	'\17'	SI	^O	[O]pen file, edit file
CH_PRINT	0x10	'\20'	DLE	^P	[P]rint file, send file
CH_QUERY	0x11	'\21'	DC1	^Q	[Q]uery, display help
CH_REPLACE	0x12	'\22'	DC2	^R	find and [R]eplace
CH_SAVE	0x13	'\23'	DC3	^S	[S]ave file
CH_INS	0x14	'\24'	DC4	^T	[T]oggle Insert switch, mark file
CH_HOME	0x15	'\25'	NAK	^U	Home, begin of row, begin of files
CH_PASTE	0x16	'\26'	SYN	^V	paste from clipboard
CH_SYN	= CH_PASTE				(synchronise transfer)
CH_CLOSE	0x17	'\27'	ETB	^W	close file
CH_CUT	0x18	'\30'	CAN	^X	cut selected text
CH_REDO	0x19	'\31'	EM	^Y	redo previously undo action
CH_UNDO	0x1A	'\32'	SUB	^Z	undo action
CH_ESC	0x1B	'\e'	ESC	^[Esc, break, menu
CH_RIGHT	0x1C	'\34'	FS	^\\	Right (+Shift: End, +Ctrl: Word right)
CH_UP	0x1D	'\35'	GS	^]	Up (+Shift: PageUp, +Ctrl: Text start)
CH_LEFT	0x1E	'\36'	RS	^_	Left (+Shift: Home, +Ctrl: Word left)
CH_DOWN	0x1F	'\37'	US	^_	Down (+Shift: PageDn, +Ctrl: Text end)
CH_SPC	0x20	' '	SPC		space
CH_DEL	0x7F	'\x7F'	DEL		delete character on cursor

1.9. Directory structure

How PicoLibSDK library files and directories are organized:

!DemoVGA - SD card contents with sample programs and loader for DemoVGA.

!Pico - sample programs for base Raspberry Pico module.

!Picoino10 - SD card contents with sample programs and loader for Picoino.

!PicoinoMini - SD card contents with sample programs and loader for PicoinoMini.

!PicoPad10 - SD card contents with sample programs and loader for PicoPad version 1.0.

!PicoPad20 - SD card contents with sample programs and loader for PicoPad version 2.0 in ARM mode.

!PicoPad20riscv - SD card contents with sample programs and loader for PicoPad version 2.0 in RISC-V mode.

!PicoPadVGA - SD card contents with sample programs and loader for PicoPadVGA.

!Picotron - SD card contents with sample programs and loader for Picotron.

_boot2 - boot2 loader stage 2, which is placed at the beginning of each UF2 program and is used to initialize the Flash memory interface. The boot2 loader is compiled using the prepared batch file c.bat and requires the checksum program boot2crc.exe.

_devices - device definitions and drivers. Currently, the ready devices are DemoVGA, Pico, Picoino 1.0, PicoinoMini, PicoPad 0.8, PicoPad 1.0, PicoPad 2.0 (ARM or RISC-V), PicoPadVGA, Picotron. The target device is selected at compile time with the c.bat file parameter ('demovga', 'pico', 'picoino10', 'picoinomini', 'picopad08', 'picopad10', 'picopad20', 'picopad20riscv', 'picopadvga', 'picotron'). If the target device is not specified when compiling the application, the default device is selected by the _setup.bat file.

_display - drivers for the displays. There is a mini-VGA display driver for 4/8/15/16-bit output to a VGA monitor with 320x240 up to 800x600 pixel resolution. TFT display driver for 16-bit RGB565 format at 320x240 pixel resolution - this driver is used by PicoPad. DispHSTX is VGA and DVI (HDMI) driver for RP2350, using HSTX interface.

_doc - documentation, SDK Programmer's Guide.

_font - prepared fonts. These are non-proportional fonts with a fixed cell width for a character of 8 pixels. The file is in BMP 2-color format. The RaspPicolImg.exe program exports the font images to the C source file.

_lib - library files of utilities, including library for large real numbers and for large integers.

_loader - boot3 program loader, with the ability to run applications from the SD card.

_sdk - SDK library for controlling the RP2040/RP2350 hardware.

_tools - support programs:

- **AS4040** - Intel 4004/4040 assembler compiler
- **BinC** - export binary file into C/C++ byte array
- **Boot2CRC** - boot2 boot loader checksum calculation
- **DviTms** - generator of DVI TMDS 16-bit 2-symbol table
- **HidComp** - compiler and decompiler of HID descriptors
- **PicoPadImg** - converting BMP images to PicoPad format
- **PicoPadImg2** - converting BMP images to DispHSTX format
- **PicoPadLoaderBin** - export boot3 loader to C source code
- **PicoPadLoaderCrc** - calculate (and set) application checksum
- **PicoPadVideo** - video converter to PicoPad/Picoino format
- **RaspPicoClock** - calculation of PLL generator settings combinations
- **RaspPicolImg** - convert BMP images to Picoino format
- **RaspPicoPal332** - palette generator for Picoino and QVGA output
- **RaspPicoRle** - RLE compression of BMP images for PicoVGA library
- **RaspPicoSnd** - export WAV audio files to C source code
- **elf2uf2** - convert ELF file to UF2 file
- **pioasm** - PIO program assembler

DemoVGA - sample programs for DemoVGA board

Pico - sample programs for Raspberry Pico

Picoino - sample programs for Picoino and PicoinoMini

PicoPad - sample programs for PicoPad

Picotron - sample programs for Picotron

_setup.bat - setting parameters for compilation according to the selected target device

c_all.bat - compilation of all sample applications for the selected target device

c_all_devices.bat - compilation of all sample applications for all target devices

config_def.h - setting the default compilation configuration

d_all.bat - clean up the compilation from temporary files

global.h - global definitions (types, constants)

includes.h - global include of all header *.h files to compile

Makefile.inc - main compilation script for make.exe

memmap_*.ld - scripts for linker

1.10. Boot3 loader

The PicoLibSDK boot3 loader is a 32 KB (or 64 KB on RP2350 Pico 2) resident program that is permanently loaded at the beginning of the Flash memory. The boot3 loader can be loaded into the processor's memory only by programming via USB cable, just like the classic procedure for programming regular programs for the Raspberry Pico.

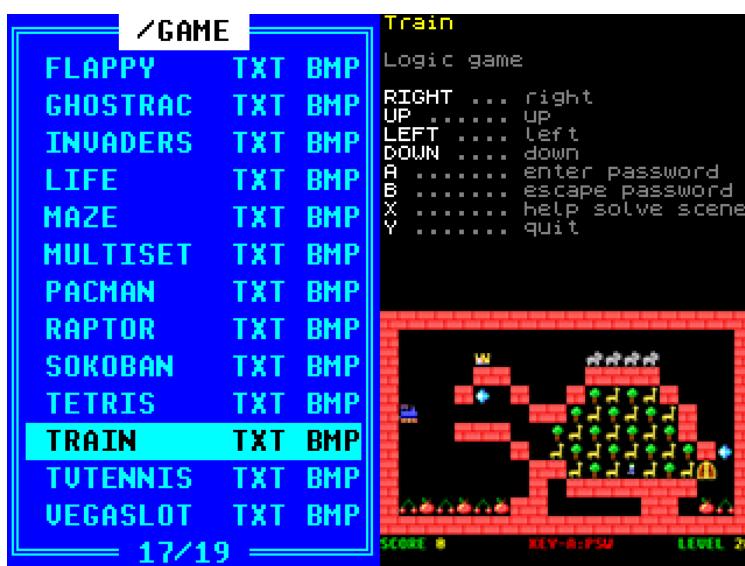
The Boot3 loader contains a standard boot2 loader (256 bytes) at its start. During system boot, the boot2 loader is started first, which initializes the Flash memory and passes on control to the boot3 loader.

The RP2350 Pico 2 can run in either ARM or RISC-V processor mode. The Boot3 loader must be of the same type as the program being run. If you change the mode between ARM and RISC-V, you must load a new boot3 loader from the PC via USB.

The boot3 loader first initializes the base devices. In the Picoino the output is to the QVGA display, implemented by the PIO, in the PicoPad the output is to the TFT display. The boot3 loader detects whether an SD card is inserted. If no SD card is inserted, it checks if there is a valid application loaded in the memory from the next address up to 32 or 64 KB (it must have a valid checksum) and if the application is OK, it runs it. If an SD card is inserted, it will not run the application, but will display the contents of the SD card and allows the user to select a program to run.

The applications on the SD card are in UF2 format. They differ from the standard format for Pico only in that they contain a boot3 loader at the beginning and that they are protected by a checksum. This allows applications to be loaded into the Pico in the classic way via USB cable, like other common applications, because the boot3 loader will be loaded into memory along with them.

Boot3 loader of PicoLibSDK:



When the boot3 loader starts an application from the SD card, it skips the initial 32 KB (or 64 KB) that the boot3 loader contains and loads the rest of the file into Flash memory. It then checks the validity of the application - it must have a valid header and a valid checksum. If everything is OK, the boot3 loader will run the application.

It is also possible to immediately start an application that is already loaded in memory - this is possible with the 'Y' button (the 'Back' button in Picoino), which is otherwise used to exit the application. The 'X' button adjusts the volume and backlight of the display.

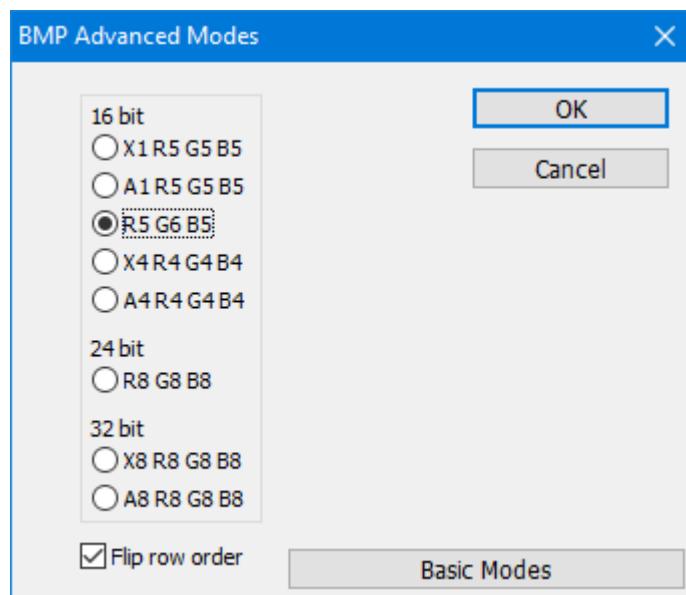
When selecting an application to run, the boot3 loader displays a preview image and description text in addition to the file name. Whether the application has an image and text available is indicated by the TXT and BMP shortcuts next to the file name.

The description text is displayed in the right half of the screen. The text can have a line length of up to 26 characters. The number of lines can be either 14, if an image is also present, or 30 lines. The following control characters can be used in the text:

- ^ is prefix of control characters
- ^^ print ^ character
- ^1..^9 print character with code 1..9
- ^A..^V print character with code 10..31
- ^0 normal gray text
- ^W white text
- ^X green text
- ^Y yellow text
- ^Z red text
- ^[start invert
- ^] stop invert

The preview image is a BMP file. For Picoino it is an 8-bit format with RGB332 palettes (that are generated by RaspPicoPal332). For PicoPad, it is a 16-bit RGB565 format - if you save such an image from Photoshop, choose the extended formats when saving and select the R5G6B5 format. For both formats, turn on the "Flip row order" option. You can get the preview images from the application as a screenshot by turning on the configuration option USE_SCREENSHOT = 1.

BMP file format for PicoPad in Photoshop:



1.11. Booting process

After the power is turned on and the reset sequence is completed, the processor starts its operation in the BootROM (16 KB or 32 KB from address 0x00000000), which is a fixed part of the RP2040/RP2350 and cannot be changed by the user. The start address ("start") is stored in the BootROM in the second entry of the interrupt vector. The first entry is the address of the stack pointer.

Both processor cores start in the BootROM in the same way. If it is core 1 (detected via Cpuid()), core 1 goes into sleep mode in a waiting loop, waiting to be woken up by the core 0 via the mailbox FIFO. Only core 0 continues its activity (main()).

The processor will first check if the BOOTSEL button is pressed. If so, it will switch to booting the program from USB. If it is not pressed, it continues booting from the external Flash memory (`_flash_boot`).

Activates the connection for the external Flash memory (the Flash memory is mapped from address 0x10000000 and has a size of 2 to 16 MB) and reads 256 bytes from the beginning of the Flash memory to end of RAM memory, to address 0x20041F00. These 256 bytes contain the stage 2 boot loader and are appended to the beginning of each application for the RP2040.

The processor checks the boot loader checksum - the calculated CRC32A for the first 252 bytes must match the checksum in the last 4 bytes. If the checksum matches, the program jumps to the beginning of the boot2 loader. The RP2350 does not run the boot2 module, but the beginning of the program. The program itself provides the boot2 call and initializes the Flash memory.

The BootROM program initialized only the basic interface for the Flash memory - in order to be able to connect basically any Flash memory. Boot2 loader sets a faster Flash memory interface. Different boot2 loaders may be needed for different Flash memories.

Boot2 loader continues by jumping to the main application program. It reads the stack pointer settings and start address from the vector table at address 0x10000100, and jumps into the application at the start address.

Applications compiled for the PicoLibSDK library may contain another boot loader at the beginning, stage 3. It is 32 KB or 64 KB in size and is used to load programs from the SD card into Flash memory.

Each program prepared in this way contains the entire 32 KB or 64 KB boot3 loader at the beginning. This allows the program to be loaded into the Flash memory using both the classic USB cable procedure and the boot3 loader from the SD card. When booting a program from an SD card, the boot3 loader does not load the first 32 KB or 64 KB of the file, but only the following part, with the program itself. The program itself thus starts at address 0x10008000 with its vector table.

Before starting the application, the Boot3 loader first checks the special application header, which is located after the vector table. It calculates the CRC32A checksum of the entire application in Flash memory, and if the checksum is OK, it jumps to the application entry point according to the vector table.

If the system starts via reset only, without loading the application into memory, the boot3 loader checks if an SD card is inserted into the device. If it is inserted, it is assumed that the user will want to start another application by selecting it, and the contents of the SD card will be displayed. If the SD card is not inserted, it is assumed to just restart the application, in which case the boot3 loader will jump to start the application.

Whether the application starts via the boot3 loader or just via the boot2 loader, in both cases it starts with the start address stored in the second entry of the vector table “_reset_handler” (the first entry is the stack pointer). This code is located in the crt0.S file.

If core1 is detected, it jumps back to BootROM to go sleep, waiting to be activated. Only core0 will continue. The program copies the read-only data from Flash memory to RAM memory - this initializes the global variables. It also transfers to RAM the codes of functions that are to be executed from RAM and not from Flash.

Next, the program clears the BSS segment, which is an area that does not require initialization, but can be expected to contain zero variables.

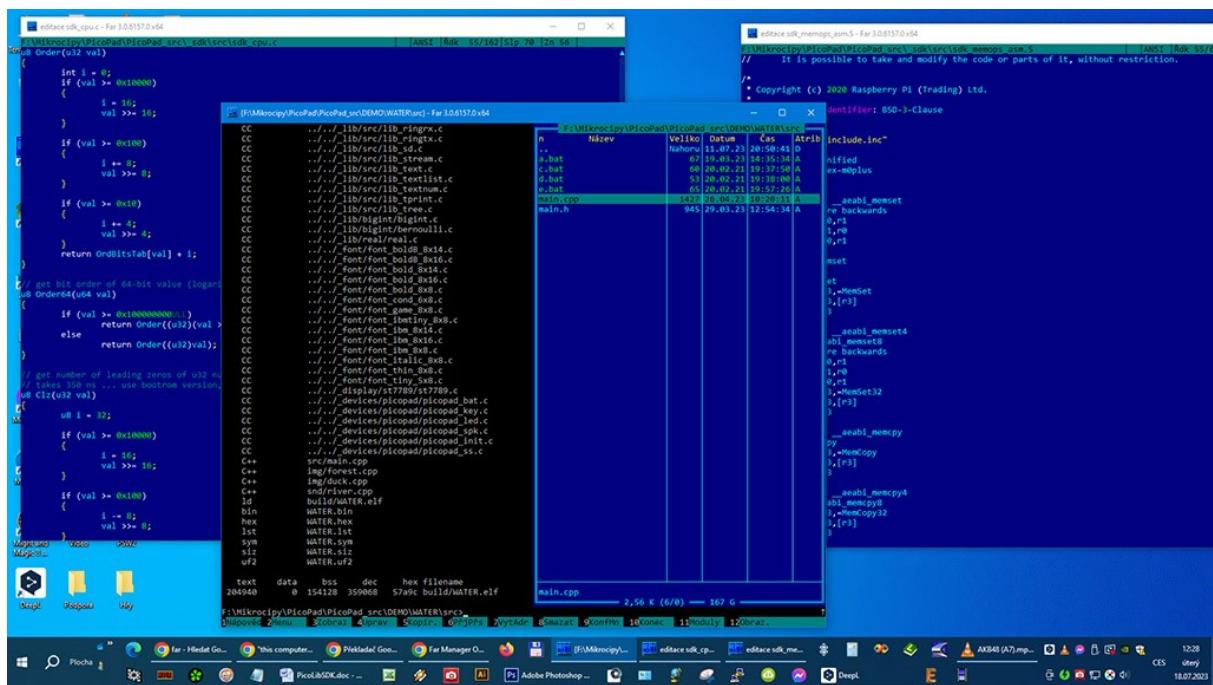
The program calls the Runtimelinit() function (in the sdk_runtime.c file), which contains the basic initialization of the system without burdening the programmer of it. The function initializes peripherals, object constructors, ROM functions, floating math, interrupt controller, system clock, memory allocator, and finally target device. After it, program starts with main() function with main program code.

When the program wants to return to the boot3 loader (exiting the program with the Y key), it does so by calling the ResetToBootLoader() function. The function sets the magic value in the scratch register of the watchdog and activates the watchdog. Watchdog will reset the processor. From the magic value boot3 loader recognizes that it is an application exit, so it displays the contents of the SD card instead of jumping into the application.

1.12. FAR File Manager

I highly recommend that you use the FAR file manager when developing applications for the microchips. FAR includes everything that is needed to develop software for microchips - it includes quality text editor, manipulation with files, searching, comparison. And above all, it includes a console window where you can still see the output from the compiler. The entire PicoLibSDK library was developed using the FAR manager. You can download FAR manager from the web site <https://www.farmanager.com/>.

Working with FAR in multiply windows:



Most important controls of the file window

Tab	change active panel
Ctrl+O	hide/show both panels
Ctrl+U	swap panels
Ctrl+P	hide/show inactive panel
Alt+F1	change disk in left panel
Alt+F2	change disk in right panel
Insert	select file
[Numeric +]	select files by mask
Shift+[Numeric +]	select all files (and directories)
Shift+[Numeric -]	unselect all files (and directories)
Ctrl+F	full path to command line
F3	view file

F4	edit file
F5	copy selected files and directories
F6	move selected files and directories
F7	create directory
F8	delete selected files and directories
F11	modules (compare files and their contents)
Alt+F7	search files and their contents
Alt+F8	command history
Shift+F4	create new file and edit

You can preset the window size in the shortcut properties.

Most important controls of text editor (F4)

F2	save file
Shift+F2	save file As
F7	search
Ctrl+F7	search and replace
Shift+F8	select code page
Esc	quit editor
Ctrl+Z	undo
Ctrl+Shift+Z	redo
Shift+arrows	select block
Ctrl+C	copy to clipboard
Ctrl+X	cut to clipboard
Ctrl+V	paste from clipboard
Delete	delete selection

1.13. Original-SDK interface

For backward compatibility, the PicoLibSDK library provides most of the functions of the original SDK library. The functions of the original SDK are mostly implemented by inline functions, redirecting the code to native PicoLibSDK library functions. Main differences:

The PicoLibSDK library simplifies some features and extends others. One simplification is that PicoLibSDK adheres less strictly to locking down functions for multitasking environments. Due to limited processor memory, multitasking is not expected. Excessive locking can slow down program functionality, so locking is concentrated on only the most necessary operations.

Functions from the original SDK for working with USB are not implemented. The USB interface is handled differently in the PicoLibSDK, in a simpler and faster way (instead of USB tasks, everything is handled more efficiently in interrupts). Interfaces are incompatible.

The functions for WiFi and Bluetooth are not implemented - these modules are not yet implemented in PicoLibSDK either.

SDK functions for accessing processor devices are preferentially addressed by device indexes in the PicoLibSDK library, while the original SDK library uses a pointer to device registers. Addressing via device indexes is more simple, but in some cases addressing via pointers may generate slightly more efficient code. For this reason the ability to address devices via pointers has been added to the PicoLibSDK library. Many functions are therefore available in 2 formats. If you do not require more code efficiency, we recommend using the simpler addressing via indexes.

The hardware integer divider is used by the original SDK library so that it detects the states of the divider and saves the state of the divider during an interrupt. The PicoLibSDK library disables global interrupts while the divider is in use - there is no need to store its state during interrupt, but interrupts can be disabled for about 10 clock cycles.

Using the original SDK interface requires enabling the **USE_ORIGSDK** switch in config.h. By default, this switch is globally enabled. It may need to be turned off if code conflicts with original SDK functions.

When importing source code from the original SDK, an important difference is the structure of the header files. In the original SDK library, the *.h files of the modules that are used are included in the code. In the PicoLibSDK library, you only need to include the include.h file and it will ensure that all files are included. If needed, library modules can be conditioned using configuration switches in the config.h file.

1.14. History of versions

03/08/2023 PicoPad prototype with pre-alpha PicoLibSDK version 0.8.

04/28/2023 PicoPad full version with alpha PicoLibSDK version 0.90

06/28/2023 alpha PicoLibSDK version 0.91, improved battery measurement

07/30/2023 version 1.00: first final release

- 08/06/2023** version 1.01: printf() print memory block, stdio to UART and FILE, add Pico and PicinoMini devices, some USB repairs
- 08/14/2023** version 1.02: added video player, added DemoVGA board, added compilation scripts for Linux
- 08/19/2023** version 1.03: SPI SD card speed fix, timer-alarm treatment for short times
- 08/26/2023** version 1.04: Replace qvga/vga/qdraw/drawtft libraries with more versatile minivga/draw libraries with 4/8/15/16 bit output and 320x240 to 800x600 pixel resolution. Slide-show player added. Added Picotron device, with 4-bit VGA output. Added BOOTSEL to boot3 loader (in Battery menu).
- 09/09/2023** version 1.05: CSYNC in VGA driver, added PicoVGA8 library, added VREG library, setup volume and backlight, calibrate crystal.
- 10/03/2023** version 1.06: added PicoPadVGA device and some drawing functions.
- 10/06/2023** version 1.07: support for RAM programs, support for saving and loading Flash boot loader
- 10/18/2023** version 1.08: simulate keypad using USB keyboard (USE_USBPAD, A->Ctrl, B->Alt, X->Shift, Y->Esc).
- 12/05/2023** version 1.09: ADPCM sound compression, original-SDK interface.
- 12/30/2023** version 1.10: Intel 4004/4040 CPU emulator, DVI (HDMI) display, DVIVGA (HDMI with VGA) display, file selector
- 01/27/2024** version 1.11: loader passes home path to the application; CPU emulators: I4004, I4040, I8008, I8048, I8052, I8080, I8085, I8086, I8088, I80186, I80188, M6502, M65C02, X80, Z80.
- 05/01/2024** version 1.12: Screen saver in loader to turn off when charging. Simple PC DOS emulator.
- 06/14/2024** version 1.13: Game Boy Emulator
- 10/08/2024** version 2.00: RP2350 Pico 2 support
- 10/27/2024** version 2.01: Fast float library for RISC-V Hazard3 core.
- 11/05/2024** version 2.02: NES Emulator
- 12/03/2024** version 2.03: Fast double library for RISC-V Hazard3 core.
- 02/24/2025** version 2.04: DispHSTX library, DrawCan library, faster compilation, configurable compilation paths
- 03/02/2025** version 2.05: Build of DispHSTX v1.01 library for Raspberry PicoSDK
- 03/05/2025** version 2.06: PWMSnd update - higher quality audio output, less noise, higher PWM output sample rate, 12-bit output, stereo and 16-bit format support
- 03/14/2025** version 2.07: MP3 decoder and player
- 05/01/2025** version 2.08: Added PicopadHSTX device and DispHSTXMini driver

2. SDK Software Development Kit

The SDK library is used to control the hardware peripherals of the processor. The SDK source files are located in the `_sdk` folder.

2.1. ADC - Analogue to Digital Converter

Files: `sdk_adc.h, sdk_adc.c`

Config: `USE_ADC (default 1)`

RP2040/RP2350 has an internal analogue-digital converter (ADC) with the following features:

- SAR ADC (Successive Approximation Register ADC)
- conversion speed 500 ksps (using an independent 48MHz clock)
- 12-bit with 8.7 ENOB (Effective Number of Bits)
- Five input mux on RP2040 and RP2350A:
 - Four inputs that are available on package pins shared with GPIO[29:26]
 - One input is dedicated to the internal temperature sensor
- Nine input mux on RP2350B:
 - Eight inputs that are available on package pins shared with GPIO[47:40]
 - One input is dedicated to the internal temperature sensor
- Four element receive sample FIFO
- Interrupt generation
- DMA interface

When using an ADC input shared with a GPIO pin, the pin's digital functions must be disabled by setting IE low and OD high in the pin's pad control register. You can use function `ADC_PinInit()`.

The ADC requires a 48MHz clock `CLK_ADC`, which could come from the `CLK_PLL_USB` clock. Capturing a sample takes 96 clock cycles ($96 \times 1/48\text{MHz} = 2 \text{ us}$ per sample (500 ksps)

The voltage measurement range is related to the supply voltage of the digital pins (`IOVDD`), not to the supply voltage of the ADC (`ADC_AVDD`). The input voltage must not exceed `IOVDD`.

The temperature sensor measures the V_{be} voltage of a biased bipolar diode, connected to the fifth ADC channel `ADC_MUX_TEMP`. Typically, $V_{be} = 0.706\text{V}$ at 27 degrees C, with a slope of -1.721mV per degree. Therefore the temperature can be approximated as follows:

$$T = 27 - (\text{ADC_voltage} - 0.706)/0.001721$$

To compile ADC library, set configuration parameter **USE_ADC** to 1 (enabled by default).

How to use ADC

- initialize ADC with `ADC_Init()`
- prepare GPIO inputs with `ADC_PinInit()`, or enable temperature sensor with `ADC_TempEnable()`
- select input with `ADC_Mux()`
- read conversion with `ADC_Single()`, it takes 2 us

void ADC_Init();

Initialize ADC: Reset ADC periphery, enable ADC and wait for power-up sequence.

void ADC_Term();

Terminate ADC: Reset ADC as after power-on.

void ADC_Disable();

Temporary disable ADC: Power off ADC and disable its clock.

void ADC_Enable();

Enable ADC after `ADC_Disable`: Power on ADC and enable its clock.

void ADC_PinInit(int pin);

pin ... pin 26 to 29, or 40 to 47 on RP2350B

Initialize GPIO to use as an ADC pin.

void ADC_PinTerm(int pin);

pin ... pin 26 to 29, or 40 to 47 on RP2350B

Terminate pin that was used as ADC input. Pin will be reset to the same state as after reset.

void ADC_Mux(int input);

input ... input 0 to 4, or 0 to 8 on RP2350B (use **ADC_MUX_***)

Select ADC input. No settling time is required when switching input. Available inputs are:

- **ADC_MUX_GPIO26** ... pin GPIO 26
- **ADC_MUX_GPIO27** ... pin GPIO 27
- **ADC_MUX_GPIO28** ... pin GPIO 28
- **ADC_MUX_GPIO29** ... pin GPIO 29
- **ADC_MUX_TEMP** ... temperature sensor

Or inputs on RP2350B:

- **ADC_MUX_GPIO40** ... pin GPIO 40

- **ADC_MUX_GPIO41** ... pin GPIO 41
- **ADC_MUX_GPIO42** ... pin GPIO 42
- **ADC_MUX_GPIO43** ... pin GPIO 43
- **ADC_MUX_GPIO44** ... pin GPIO 44
- **ADC_MUX_GPIO45** ... pin GPIO 45
- **ADC_MUX_GPIO46** ... pin GPIO 46
- **ADC_MUX_GPIO47** ... pin GPIO 47
- **ADC_MUX_TEMP** ... temperature sensor

u8 ADC_GetMux();

Get currently selected ADC input. Returns value 0 to 4 or 0 to 8 (**ADC_MUX_***).

void ADC_RoundRobin(int mask);

mask ... mask of bits B0 to B4 (or B0 to B8) of inputs. Use ORed BIT(**ADC_MUX_***), or set 0 to disable round-robin sampling.

Set round-robin sampling selector. Round-robin allows the ADC to sample multiple inputs, in an interleaved fashion, while performing free-running sampling. Each bit in round-robin mask corresponds to one of the five possible values of input selection. When the ADC completes a conversion, selection will automatically cycle to the next input whose corresponding bit is set in round-robin.

void ADC_TempEnable();

Enable temperature sensor. If the temperature sensor was not turned on, the function waits 1 millisecond for the sensor power to stabilize.

void ADC_TempDisable();

Disable temperature sensor. It saves 40 uA.

void ADC_StartOnce();

Start single conversion. The result can be read after 96 clock cycles using the **ADC_Result()** function. The **ADC_Single()** function can also be used which waits for the result of the conversion.

Bool ADC_Ready();

Check if result of conversion is ready (returns False if conversion is in progress).

void ADC_Wait();

Wait for end of conversion. It should take a maximum of 96 clock cycles (2 microseconds on 48 MHz clock).

u16 ADC_Result();

Get result of last conversion. The function returns a 12-bit value from 0 to 4095 that corresponds to a voltage range of 0 to IOVDD, where IOVDD is supply voltage of the digital pins.

u16 ADC_Single();

Do single conversion. The function includes starting the conversion, waiting for the conversion to complete and loading the conversion result. The function typically takes 96 clock cycles, i.e. 2 microseconds on a 48 MHz clock. The function returns a 12-bit value from 0 to 4095 that corresponds to a voltage range of 0 to IOVDD, where IOVDD is supply voltage of the digital pins.

u16 ADC_SingleDenoise();

Do single conversion with denoise. The function performs 16 single conversions, sums them and returns the result as 16-bit value in the range 0 to 65535 that corresponds to a voltage range of 0 to IOVDD, where IOVDD is supply voltage of the digital pins. The function takes 32 microseconds on a 48 MHz clock.

float ADC_SingleU();

Do single conversion and recalculate result to voltage on 3.3V. The function performs a single conversion with ADC_SingleDenoise noise filtering, multiplies the result by 3.3f/65536 (reference voltage/value range), and returns the resulting voltage as a float number.

int ADC_SingleUint();

Do single conversion and recalculate result to voltage on 3.3V, integer in mV (range 0..3300). The function performs a single conversion with ADC_SingleDenoise noise filtering, multiplies the result by 3300/65536 (reference voltage in [mV]/value range) and returns the resulting voltage as an integer in [mV].

void ADC_MultiEnable();

Enable continuously repeated conversion. The ADC will automatically start new conversions at regular intervals set with the ADC_ClkDiv() function. The most recent conversion result is always available via ADC_Result(). For IRQ or DMA driven streaming of samples, the ADC FIFO must be enabled with the ADC_FifoSetup() function.

void ADC_MultiDisable();

Disable continuously repeated conversion.

Bool ADC_Err();

Check if most recent ADC conversion encountered an error. In such case, result is undefined or noisy. Flag is reset by starting the next conversion.

Bool ADC_ErrSticky();

Check if some past ADC conversion encountered an error. Clear this flag with the ADC_ErrClear() function.

void ADC_ErrClear();

Clear sticky error, returned by the ADC_ErrSticky() function.

void ADC_ClkDiv(int clkdiv);

clkdiv ... clock divisor * 256

Set ADC clock divisor. If non-zero value is entered, ADC_MultiEnable() will start conversions with that interval. Interval of samples will be $T = (1 + \text{clkdiv}/256)$ cycles. One conversion takes 96 cycles, so minimal value of clkdiv is $96*256 = 0x5F00$ (period 2 us, 500 kHz). Maximal value is 0xFFFFFFFF (period 1365 us, 732 Hz).

void ADC_ClkDivFloat(float clkdiv);

clkdiv ... clock divisor

Set ADC clock divisor as float (with resolution 1/256). If non-zero value is entered, ADC_MultiEnable() will start conversions with that interval. Interval of samples will be $T = (1 + \text{clkdiv})$ cycles. One conversion takes 96 cycles, so minimal value of clkdiv is 96 (period 2 us, 500 kHz). Maximal value is 65535 (period 1365 us, 732 Hz).

void ADC_Freq(u32 freq);

freq ... sampling frequency in Hz, min. 732 Hz, max. 500000 Hz, 0 = switch off

Set ADC sampling frequency of repeated conversions (enabled by ADC_MultiEnable()). The function takes the current **CLK_ADC** frequency, recalculates it to the division ratio, limits the result to the allowed range and sets the ADC_ClkDiv() clock divider. The value 0 disables free-running conversions.

u32 ADC_GetFreq();

Get current sampling frequency in Hz (typically 732 to 500000 Hz, 0 = off).

void ADC_FifoSetup(Bool en, Bool shift, Bool err, Bool dreq_en, int dreq_thresh);

en ... write result to the FIFO after each conversion

shift ... right-shift result to be one-byte 8-bit in size (to enable 8-bit DMA transfer)

err ... bit 15 of FIFO will contain error flag for each sample

dreq_en ... enable DMA requests when FIFO contains data

dreq_thresh ... threshold for DREQ or IRQ requests (DREQ/IRQ asserted when level \geq threshold)

Setup ADC FIFO. FIFO is 4 samples long. Each sample is 16-bit long and can be transferred using interrupt IRQ or DMA DREQ, with 16-bit or 8-bit sample size.

Bool ADC_IsEmpty();

Check if ADC FIFO is empty.

Bool ADC_IsFull();

Check if ADC FIFO is full.

Bool ADC_IsUnder();

Check if ADC FIFO has been underflow. Flag can be cleared using ClearUnder() function.

Bool ADC_IsOver();

Check if ADC FIFO has been overflow. Flag can be cleared using ClearOver() function.

void ADC_ClearUnder();

Clear flag of ADC FIFO has been underflow.

void ADC_ClearOver();

Clear flag of ADC FIFO has been overflow.

u8 ADC_Level();

Get number of samples waiting in FIFO. Returns value 0 to 4.

u16 ADC_Fifo();

Get value from ADC FIFO (bit 15 can contain error flag).

u16 ADC_FifoWait();

Get value from ADC FIFO and wait to have data (bit 15 can contain error flag).

void ADC_FifoFlush();

Discard all results in FIFO.

Bool ADC_IntRaw();

Get raw interrupt flag indicating that the FIFO has reached a threshold level. Raw interrupt flag does not depend on the interrupt enable setting and does not depend on the force interrupt. This flag will be reset by reading the FIFO below the set threshold level.

void ADC_IntEnable();

Enable FIFO interrupt. If the FIFO level reaches the threshold level set with ADC_FifoSetup(), the **IRQ_ADC_FIFO** interrupt will be triggered.

void ADC_IntDisable();

Disable FIFO interrupt.

void ADC_Force();

Force FIFO interrupt request. The "Force FIFO" is used to invoke the interrupt handler by software. In the interrupt handler, the ADC_IsForced() test can be used to determine whether the interrupt occurred from hardware or software. The detected Force flag must be cleared in the interrupt handler with ADC_ForceClear().

void ADC_ForceClear();

Clear forced FIFO interrupt request.

Bool ADC_IsForced();

Check if force FIFO interrupt request is set. Flag can be reset with ADC_ForceClear().

Bool ADC_IntStatus();

Get interrupt status after masking and forcing. Interrupt status is ORed of raw interrupt flag and force interrupt request, and is masked by interrupt enable.

float ADC_Temp();

Get current temperature in °C. The function enables the temperature measurement, switches the ADC input to the temperature sensor input, performs the measurement with de-noising and calculates the temperature. If the configuration flag TEMP_NOISE is set (indicating the depth of the temperature measurement history), an additional more thorough filtering of the measurement is performed.

2.2. Boot ROM

Files: `sdk_bootrom.h`, `sdk_bootrom.c`, `sdk_memops_asm.S`

Config: -

The RP2040/RP2350 processor contains, in addition to the RAM memory, a read-only ROM memory of size 16 KB or 32 KB, starting at address 0x00000000, called boot ROM memory. The ROM contents are fixed at the time the silicon is manufactured. It contains:

- Initial startup routine
- Flash programming routines
- USB mass storage device with UF2 support
- Utility libraries such as fast floating point (only RP2040)

The Boot ROM library provides access to the boot ROM functions. The advantage of the library functions in the boot ROM is that the boot ROM memory is very quickly accessed via single-cycle bus access. Functions from the boot ROM are accessible both by addresses and by redirected standard library functions.

void* RomFunc(u32 code);

Find ROM function given by the code `ROM_FUNC_*`. Returns NULL if not found.

Codes of ROM functions

`ROM_FUNC_POPCOUNT32` ... counts '1' bits (only RP2040)

`ROM_FUNC_REVERSE32` ... reverse order of bits (only RP2040)

`ROM_FUNC_CLZ32` ... count leading zeros (only RP2040)

`ROM_FUNC_CTZ32` ... count trailing zeros (only RP2040)

`ROM_FUNC_MEMSET` ... fill memory (only RP2040)

`ROM_FUNC_MEMSET4` ... fill memory aligned to u32 (only RP2040)

`ROM_FUNC_MEMCPY` ... copy memory (only RP2040)

`ROM_FUNC_MEMCPY44` ... copy memory aligned to u32 (only RP2040)

`ROM_FUNC_RESET_USB_BOOT` ... reset CPU to BOOTSEL mode (only RP2040)

`ROM_FUNC_WAIT_FOR_VECTOR` ... wait core 1 for launch (only RP2040)

`ROM_FUNC_CONNECT_INTERNAL_FLASH` ... restore QSPI to default and connect SSI

`ROM_FUNC_FLASH_EXIT_XIP` ... exit XIP mode to SSI

`ROM_FUNC_FLASH_RANGE_ERASE` ... erase flash

`ROM_FUNC_FLASH_RANGE_PROGRAM` ... program flash

`ROM_FUNC_FLASH_FLUSH_CACHE` ... flush flash cache

`ROM_FUNC_FLASH_ENTER_CMD_XIP` ... enter XIP mode

`ROM_FUNC_REBOOT` ... reboot (only RP2350)

ROM_FUNC_BOOTROM_STATE_RESET ... resets internal bootrom state (only RP2350)
ROM_FUNC_FLASH_RESET_ADDRESS_TRANS ... reset flash addr. trans. (only RP2350)
ROM_FUNC_FLASH_SELECT_XIP_READ_MODE ... select XIP read mode (only RP2350)
ROM_FUNC_GET_SYS_INFO ... get system info (only RP2350)
ROM_FUNC_GET_PARTITION_TABLE_INFO ... get partition table info (only RP2350)
ROM_FUNC_EXPLICIT_BUY ... perform "explicit" buy of executable (only RP2350)
ROM_FUNC_VALIDATE_NS_BUFFER ... validate buffer non-secure code (only RP2350)
ROM_FUNC_SET_ROM_CALLBACK ... set callback for RomSecureCall (only RP2350)
ROM_FUNC_CHAIN_IMAGE ... searches memory region and run (only RP2350)
ROM_FUNC_LOAD_PARTITION_TABLE ... load current partition table (only RP2350)
ROM_FUNC_PICK_AB_PARTITION ... pick A or B partition (only RP2350)
ROM_FUNC_GET_B_PARTITION ... get index of B partition (only RP2350)
ROM_FUNC_GET_UF2_TARGET_PARTITION ... get UF2 partition (only RP2350)
ROM_FUNC OTP_ACCESS ... write/read data into OTP (only RP2350)
ROM_FUNC_FLASH_RUNTIME_TO_STORAGE_ADDR ... address trans. (only RP2350)
ROM_FUNC_FLASH_OP ... perform flash operation (only RP2350)
ROM_FUNC_SET_BOOTROM_STACK ... set bootrom stack (only RP2350-RISCV)
ROM_FUNC_SET_NS_API_PERMISSION ... allow/disallow NS API (only RP2350-ARM)
ROM_FUNC_SECURE_CALL ... call secure method from non-secure (only RP2350-ARM)

void* RomData(u32 code);

Find ROM data given by the code. Returns NULL if not found.

u16 RomGetFullVersion();

Get boot rom full version. Low byte = compatibility version (see below), high byte = major version: 1=RP2040, 2=RP2350.

u8 RomGetVersion();

Get boot rom compatibility version: 1 for RP2040-B0 or RP2350-A1, 2 for RP2040-B1 or RP2350-A2, 3 for RP2040-B2.

RP2040 boot rom version 1 does not contain double library and some float functions.

void RomFnInit();

Initialize ROM functions. This function is automatically called during application startup from the internal Runtimelinit() function to ensure that ROM functions are accessible to the application after startup.

ROM functions initialized by RomFncInit()

Only RP2040:

u8 popcount(u32 val);

Counts '1' bits.

u32 reverse(u32 val);

Reverse 32-bit value.

u8 clz(u32 val);

Count leading zeros.

u8 ctz(u32 val);

Count trailing zeros.

u8* MemSet(u8* dst, u8 data, u32 len);

u8* memset(u8* dst, u8 data, u32 len);

Fill memory (returns dst). For large data use faster DMA_MemFill.

u32* MemSet32(u32* dst, u8 data, u32 len);

u32* memset4(u32* dst, u8 data, u32 len);

Fill memory aligned to u32 (returns dst). For large data use faster DMA_MemFill.

u8* MemCopy(u8* dst, const u8* src, u32 len);

u8* memcpy(u8* dst, const u8* src, u32 len);

Copy memory (returns dst). For large data use faster DMA_MemCopy. Do not use on overlapped memory areas.

u32* MemCopy32(u32* dst, const u32* src, u32 len);

u32* memcpy4(u32* dst, const u32* src, u32 len);

Copy memory aligned to u32 (returns dst). For large data use faster DMA_MemCopy. Do not use on overlapped memory areas.

void ResetUsb(u32 gpio, u32 interface);

gpio = mask of pins used as indicating LED during mass storage activity

interface = 0 both interfaces (as cold boot), 1 only USB PICOBLOCK, 2 only USB Mass Storage

Reset CPU to BOOTSEL mode.

void WaitForVector();

Exit core 1 to BOOTROM (another alternative: Core1ExitBootrom()).

Common functions to RP2040 and RP2350:

void FlashInternal();

Restore QSPI to default and connect SSI to QSPI pads.

void FlashExitXip();

Exit XIP mode to SSI.

void RomFlashErase(u32 addr, u32 count, u32 block, u8 cmd);

addr = start address to erase (offset from start of flash **XIP_BASE**; must be aligned to 4 KB **FLASH_SECTOR_SIZE**)

count = number of bytes to erase (must be multiple of 4 KB **FLASH_SECTOR_SIZE**)

block = block size to erase larger block, use default 64 KB **FLASH_BLOCK_SIZE**

cmd = block erase command, to erase large block, use default 0xD8
FLASH_BLOCK_ERASE_CMD

Erase flash. ROM internal, use function FlashErase() instead.

void RomFlashProgram(u32 addr, const u8* data, u32 count);

addr = start address to program (offset from start of flash **XIP_BASE**; must be aligned to 256 B **FLASH_PAGE_SIZE**)

data = data to program

count = number of bytes to program (must be multiple of 256 B **FLASH_PAGE_SIZE**)

Program flash. ROM internal, use function FlashProgram() instead.

void FlashFlush();

Flush flash cache.

void FlashEnterXip();

Enter XIP mode.

Only RP2350:

int RomReboot(u32 flags, u32 delay, u32 p0, u32 p1);

Reboot. See 'reboot' function in datasheet for details.

void RomFlashResetTrans();

Reset flash address translation.

void RomFlashSelectMode(u32 mode, u8 clkdiv);

Select XIP read mode.

int RomGetSysInfo(u32* buf, u32 bufsize, u32 flags);

Set system info.

int RomGetPartInfo(u32* buf, u32 bufsize, u32 flags);

Get partition table info.

int RomExplicitBuy(u8* buf, u32 bufsize);

Perform "explicit" buy of executable.

void* RomValidateNsBuf(const void* addr, u32 size, u32 write, u32* ok);

Validate buffer from non-secure code.

int* RomSetCallback(u32 num, bootrom_api_callback_generic_t cb);

Set callback for RomSecureCall.

int RomChainImage(u8* wrk_base, u32 wrk_size, u32 win_base, u32 win_size);

Searches memory region and run.

int RomLoadPartTab(u8* wrk_base, u32 wrk_size, Bool force);

Load current partition table.

int RomPickABPart(u8* wrk_base, u32 wrk_size, uint part_num, u32 update);

Pick A or B partition.

int RomGetBPart(uint pi_a);

Get index of B partition.

**int RomGetUF2Part(u8* wrk_base, u32 wrk_size, u32 family,
resident_partition_t* part);**

Get UF2 partition.

int RomOtpAccess(u8* buf, u32 bufsize, u32 cmd);

Write/read data into OTP.

int* RomFlashTrans(uint* addr);

Perform address translation.

int RomFlashOp(int flags, uint* addr, u32 size, u8* buf);

Perform flash operation.

Only RP2350 RISC-V

int RomSetStack(bootrom_stack_t* stack);

Set bootrom stack.

Only RP2350 ARM

int RomSetNsApi(uint apinum, Bool allow);

Allow/disallow specific NS API.

int RomSecureCall(uint* a0, ...);

Call secure method from non-secure code.

2.3. Clocks

Files: `sdk_clocks.h`, `sdk_clocks.c`

Config: -

The Clocks library feeds a clock signal to the processor components. Several clock sources are available. Clock sources can be switched while the processor is running. All clock lines have a multiplexer, designated as auxiliary mux. This mux has a conventional design whose output will glitch when changing the select control. Two clock lines (`CLK_SYS` and `CLK_REF`) have an additional multiplexer, referred to as the glitchless mux. The glitchless mux can switch between clock sources without generating a glitch on the output. Clock glitches should be avoided at all costs because they may corrupt the logic running on that clock. This means that any clock line with only an aux mux must be disabled while switching the clock source. If the clock line has a glitchless mux (`CLK_SYS` and `CLK_REF`), then the glitchless mux should switch away from the aux mux while changing the aux mux source.

Available combinations of multiplexer clock input/output RP2040

input\output	<code>_GPOUTx</code>	<code>_REF</code>	<code>_SYS</code>	<code>_PERI</code>	<code>_USB</code>	<code>_ADC</code>	<code>_RTC</code>
<code>CLK_REF</code>	x	.	o!
<code>CLK_SYS</code>	x	.	.	x!	.	.	.
<code>CLK_PERI</code>
<code>CLK_USB</code>	x
<code>CLK_ADC</code>	x
<code>CLK_RTC</code>	x
<code>CLK_ROSC</code>	x	o!	x	x	x	x	x
<code>CLK_XOSC</code>	x	o	x	x	x	x	x
<code>CLK_PLL_SYS</code>	x!	.	x	x	x	x	x
<code>CLK_PLL_USB</code>	x	x	x	x	x!	x!	x!
<code>CLK_GPINx</code>	x	x	x	x	x	x	x

Legend: .=cannot connect, x=auxiliary mux, o=glitchless mux, !=default mux

Available combinations of multiplexer clock input/output RP2350

input\output	_GPOUTx	_REF	_SYS	_PERI	_HSTX	_USB	_ADC
CLK_REF	x	.	o!
CLK_SYS	x	.	.	x!	x!	.	.
CLK_PERI
CLK_HSTX	x
CLK_USB	x
CLK_ADC	x
CLK_ROSC	x	o!	x	x	.	x	x
CLK_XOSC	x	o	x	x	.	x	x
CLK_PLL_SYS	x!	.	x	x	x	x	x
CLK_PLL_USB	x	x	x	x	x	x!	x!
CLK_GPINx	x	x	x	x	x	x	x
CLK_LPOSC	x	o
CLK OTP_2FC	x
CLK_PLL_OPCG	x	x

Legend: .=cannot connect, x=auxiliary mux, o=glitchless mux, !=default mux

Output clock lines (some can also be used as input of another clock line)

CLK_GPOUT0	GPIO muxing output GPOUT0 (max. 50 MHz)
CLK_GPOUT1	GPIO muxing output GPOUT1 (max. 50 MHz)
CLK_GPOUT2	GPIO muxing output GPOUT2 (max. 50 MHz)
CLK_GPOUT3	GPIO muxing output GPOUT3 (max. 50 MHz)
CLK_REF	watchdog and timers reference clock (6..12 Mhz; glitchless mux)
CLK_SYS	system clock to processors, bus fabric, memory, memory mapped registers (125 MHz, up to 133 MHz or 150 MHz; has glitchless mux)
CLK_PERI	peripheral clock for UART and SPI (12..125 or 150 MHz)
CLK_USB	USB clock (must be 48 MHz)
CLK_ADC	ADC clock (must be 48 MHz)
CLK_RTC	RTC real-time counter clock (46875 Hz) (only RP2040)
CLK_HSTX	HSTX clock (only RP2350)

Input clock generators

CLK_ROSC	ring oscillator ROSC (6 MHz)
CLK_XOSC	crystal oscillator XOSC (12 MHz)
CLK_PLL_SYS	system PLL (125 MHz)
CLK_PLL_USB	USB PLL (48 MHz)
CLK_GPIN0	GPIO muxing input GPIN0
CLK_GPIN1	GPIO muxing input GPIN1
CLK_LPOSC	low power oscillator LPOSC (32768 Hz) (only RP2350)
CLK OTP_2FC	OTP CLK2FC (only RP2350)
CLK_PLL_OPCG	USB PLL primary ref OPCG (only RP2350)

typedef void (*resus_callback_t)();

Resus callback function type declaration.

clock_hw_t* ClockGetHw(int clk);

clk ... clock line or clock generator **CLK_***

Get clock line hardware interface from clock line index.

u8 ClockGetIdx(const clock_hw_t* hw);

Get clock line index from clock line hardware interface.

u32 ClockGetHz(int clk);

clk ... clock line or clock generator **CLK_***

Get current clock frequency in [Hz] of clock line or clock generator. Value is taken from the table, which is initiated during clock line setup.

u8 ClockGetSrc(int clk);

clk ... clock line or clock generator **CLK_***

Get current clock source of clock line or clock generator. Returns **CLK_*** or 0 = no clock source.

const char* ClockGetName(int clk);

clk ... clock line or clock generator **CLK_***

Get short clock name of clock line or clock generator (returns string "GPOUT0" ... "GPIN1").

void ClockStop(int clk);

clk ... clock line or clock generator **CLK_***

Stop clock **CLK_***. Cannot stop **CLK_SYS** and **CLK_REF** clocks.

u32 ClockSetup(int clk, int clksrc, u32 freq, u32 freqsrc);

clk ... output clock line **CLK_GPOUT0 .. CLK_RTC (or CLK_ADC)**

clksrc ... clock source **CLK_REF .. CLK_GPIN1** (see table of combinations)

freq ... required output frequency in Hz, 0=use source frequency without change

freqsrc ... source frequency in Hz, 0=get from table (freqsrc >= freq)

Setup clock line. Returns new frequency in Hz or 0 on error. Output frequency must be higher or equal to source frequency. During the setup process, the function can use glitchless switching if the clock line requires it. Changing the frequency divider is done in a safe way, preventing temporary overspeed.

void ClockInit();

Initialize clocks after start. This function is automatically called during application startup from the internal RuntimeInit() function.

void ClockPIISysSetup(int fbdv, int div1, int div2);

fbdv ... feedback divisor, 16..320

div1 ... post divider 1, 1..7

div2 ... post divider 2, 1..7 (should be div1 >= div2, but auto-corrected)

Set system clock PLL to new setup. Dependent clocks are not updated. Setting the PLL will change the system clock **CLK_SYS**. If other clock lines are derived from the system clock, it may be necessary to reconfigure them. Use this function rather than the PIISetup() function because it provides a temporary disconnection of the **CLK_SYS** system clock from the PLL. It may be more convenient to use the ClockPIISysFreq() function instead of this function ClockPIISysSetup().

void ClockPIISysFreq(u32 freq);

freq ... required frequency in [kHz]

Set system clock PLL to new frequency in kHz. Dependent clocks are not updated. Setting the PLL will change the system clock **CLK_SYS**. If other clock lines are derived from the system clock, it may be necessary to reconfigure them. Use this function rather than the PIISetFreq() function because it provides a temporary disconnection of the **CLK_SYS** system clock from the PLL.

int GetClkDivBySysClock(u32 freq);

freq ... required frequency in [kHz]

Get recommended flash divider by system clock in kHz.

int GetVoltageBySysClock(u32 freq);

freq ... required frequency in [kHz]

Get recommended voltage by system clock in kHz. Returns VREG_VOLTAGE_1_10 to VREG_VOLTAGE_1_30.

void ClockPIISysFreqVolt(u32 freq);

freq ... required frequency in [kHz]

Set system clock PLL to new frequency in kHz and auto-set system voltage and flash divider (dependent clocks are not updated).

u32 FreqCount(int clk);

clk ... clock line or clock generator **CLK_REF .. CLK_GPIN1 (CLK_PLL_OPCG)**

Precise measure frequency of clock line or clock generator **CLK_REF .. CLK_GPIN1 (CLK_PLL_OPCG)** with result in [Hz]. Measure interval is 128 ms, resolution +-15 Hz. The frequency measurement uses a frequency counter, which is controlled by the **CLK_REF** clock - the source is usually a 12 MHz crystal oscillator.

u32 FreqCountkHz(int clk);

clk ... clock line or clock generator **CLK_REF .. CLK_GPIN1 (CLK_PLL_OPCG)**

Fast measure frequency of clock line or clock generator **CLK_REF .. CLK_GPIN1 (CLK_PLL_OPCG)** with result in [kHz]. Measure interval is 2 ms, resolution +-1 kHz. The frequency measurement uses a frequency counter, which is controlled by the **CLK_REF** clock - the source is usually a 12 MHz crystal oscillator.

void ClockResusEnable(resus_callback_t cb);

Enable resus function with callback handler (NULL=not used). Resus event come when clk_sys is stopped

void ClockResusDisable();

Disable resus function.

u32 ClockGpoutDiv(int gpio, int clksrc, u32 clkdiv);

u32 ClockGpoutDivFloat(int gpio, int clksrc, float clkdiv);

gpio ... GPIO pin 21, 23, 24 or 25 (only GPIO21 is available on the Pico board)

clksrc ... clock source **CLK_REF..CLK_GPIN1** (see table for supported sources)

clkdiv ... **clock_divider * 256** (default 0x100, means **clock_divider=1.00**)

Enable divided clock to GPIO pin, set divider (returns new frequency in Hz or 0 on error).

u32 ClockGpoutFreq(int gpio, int clksrc, u32 freq);

gpio ... GPIO pin 21, 23, 24 or 25 (only GPIO21 is available on the Pico board)

clksrc ... clock source **CLK_REF..CLK_GPIN1** (see table for supported sources)

freq ... required frequency in Hz, 0=use source frequency

Enable divided clock to GPIO pin, set frequency (returns new frequency in Hz or 0 on error).

void ClockGpoutDisable(int gpio);

gpio ... GPIO pin 21, 23, 24 or 25 (only GPIO21 is available on the Pico board)

Disable divided clock to GPIO pin.

u32 ClockGpinSetup(int clk, int gpio, u32 freq, u32 freqsrc);

clk ... clock line index CLK_GPOUT0..CLK_RTC

gpio ... GPIO pin 20 or 22

freq ... required frequency in Hz

freqsrc ... frequency in Hz of source (must be freqsrc >= freq)

Configure a clock to come from a gpio input (returns new frequency in Hz or 0 on error).

2.4. CPU Control

Files: `sdk_cpu.h`, `sdk_cpu.c`

Config: -

The CPU Control library contains support functions for working with the CPU. RP2040/RP2350 contains two CPU cores, numbers core0 and core1.

int CpuID();

Get index of current processor core (0 or 1).

void cb();

Compiler barrier. In some cases, it is necessary to prevent the compiler from changing the order of some operations. A typical example is setting a hardware register and then setting a different hardware register, at a different address. In such case, the compiler can reorder the operations. If the order must be respected, the cb() function will be inserted between the C statements. This generates no code, but becomes a boundary for the compiler not to mix order of operations.

void nop();

Function generates “no operation” instruction (“nop” operation code) into result code. This can be used to generate a short delay of 1 clock cycle or to prevent the compiler from discarding an empty loop.

void nop2();

Function generates “short jump” instruction into result code. This can be used to generate a short delay of 2 clock cycles.

void di();

Disable global interrupts.

void ei();

Enable global interrupts.

Bool geti();

Get global interrupts. Returns True if interrupts are enabled.

void seti(Bool enable);

enable ... True to enable interrupts, False to disable interrupts

Set interrupts. Set True to enable interrupts.

u32 LockIRQ();

Save and disable interrupts. Function reads interrupt mask register, disables interrupts and returns original value of the interrupt mask register. Returned value can be used later to restore state of interrupts. This function should be used at the beginning of the function during which the interrupt service is to be disabled.

void UnlockIRQ(u32 state);

state ... old value of interrupt mask register as returned by the LockIRQ() function

Restore interrupts. Function restores the interrupt mask register settings with the value returned by the LockIRQ() function.

IRQ_LOCK;

Macro to lock the section protected against triggering an interrupt. The macro will be placed at the beginning of the function. Disables global interrupts and saves the original interrupt register state to the local variable u32 irq_state.

IRQ_UNLOCK;

Macro to unlock the section protected against triggering an interrupt. The macro will be placed at the end of the function. The interrupt register settings will be restored from the local variable u32 irq_state.

void BusyWaitCycles(u32 cycles);

Busy wait at least number of sys_clk cycles.

int GetCurrentIRQ(void);

Get current IRQ_* if the CPU is handling an exception (IRQ_INVALID = no exception, thread mode).

void* GetSp();

Get current stack pointer SP. SP is current stack pointer. On booting time, it is equal to MSP.

void sev();

Send event to both cores. Used during multi-code communication.

void wfe();

Wait for event. Used during multi-code communication.

void wfi();

Wait for interrupt. Used to wake up the core.

void isb();

Instruction synchronization barrier ISB.

void dmb();

Data memory barrier. Function generates instruction dmb into output code. Data memory barrier ensures completion of data operations and transfer of changes between processor cores. At the same time, this function also serves as a compiler barrier (like the cb() function) preventing the compiler from changing the order of operations.

void dsb();

Data synchronization barrier DSB.

u32 Endian(u32 val);

val ... 32-bit value

Reverse byte order of 32-bit value. Function exchanges Big-endian and Little-endian byte order. Little endian is “Intel” order LSB MSB, big endian is “Motorola” order MSB LSB.

u16 Swap(u16 val);

val ... 16-bit value

Swap byte order of 16-bit value. Function exchanges Big-endian and Little-endian byte order. Little endian is “Intel” order LSB MSB, big endian is “Motorola” order MSB LSB.

u32 Swap2(u32 val);

val ... two 16-bit values

Swap bytes in two 16-bit values.

u32 Ror(u32 val, u8 num);

val ... 32-bit value

num ... number of shifts

Rotate bits in 32-bit value right by 'num' bits and carry lower bits to higher bits.

u32 Rol(u32 val, u8 num);

val ... 32-bit value

num ... number of shifts

Rotate bits in 32-bit value left by 'num' bits and carry higher bits to lower bits.

u8 Reverse8(u8 val);

val ... 8-bit value

Reverse order of 8 bits.

u16 Reverse16(u16 val);

val ... 16-bit value

Reverse order of 16 bits.

u32 Reverse32(u32 val);

val ... 32-bit value

Reverse order of 32 bits. Alias: reverse(u32 val).

u64 Reverse64(u64 val);

val ... 64-bit value

Reverse order of 64 bits. Alias: __revll(u64 val).

u32 Clz(u32 val);

u32 clz(u32 val);

val ... 32-bit value

Get number of leading zeros of u32 number.

u32 Clz64(u64 num);

u32 clz64(u64 num);

val ... 64-bit value

Get number of leading zeros of u64 number.

u32 Ctz(u32 val);

u32 ctz(u32 val);

Get number of trailing zeros of u32 number.

u32 Ctz64(u64 val);

u32 ctz64(u64 val);

Get number of trailing zeros of u64 number.

u32 Popcount(u32 val);

u32 popcount(u32 val);

u32 Popcount64(u64 val);

Counts '1' bits (population count).

u8 Order8(u8 val);

val ... 8-bit value

Get bit order of 8-bit value ("logarithm"). Returns position of highest bit + 1: 1..8, 0=no bit.

u32 Order(u32 val);

val ... 32-bit value

Get bit order of 32-bit value ("logarithm"). Returns position of highest bit + 1. Returns value 1 to 32, or returns 0 if value is 0.

u32 Order64(u64 val);

val ... 64-bit value

Get bit order of 64-bit value ("logarithm"). Returns position of highest bit + 1. Returns value 1 to 64, or returns 0 if value is 0.

u32 Mask(u32 val);

val ... 32-bit value

Get mask of 32-bit value. The mask is the lowest number out of all '1's that can hold the entire input value. For example, 0x123 returns 0x1FF.

u32 RangeMask(int first, int last);

u64 RangeMask64(int first, int last);

first ... bit position of lowest '1' bit

last ... bit position of highest '1' bit

Get range mask. Function returns bits set to '1' on bit range 'first' to 'last'. For example, RangeMask(7,14) returns 0x7F80.

Stack Check

The following set of functions is used to check the overflow of the stack. This can be useful, for example, when using "real" numbers, which can easily fill the entire stack. To use the following functions, the **USE_STACKCHECK** configuration switch must be enabled.

void* StackTop();

Get stack top of current CPU core. The function detects which processor core the program is running on and accordingly gets the highest stack address for the current processor core from the linker. The stack for core 0 is marked as SCRATCH_X in the linker script, and is 4 KB in size. The stack for core 1 is labeled SCRATCH_Y and is also 4 KB in size, without the 32 bytes used by the boot loader.

void* StackBottom();

Get stack bottom of current CPU core. The function detects which processor core the program is running on and accordingly gets the lowest stack address for the current processor core from the linker. The stack for core 0 is marked as SCRATCH_X in the linker script, and is 4 KB in size. The stack for core 1 is labeled SCRATCH_Y and is also 4 KB in size, without the 32 bytes used by the boot loader.

int StackFree();

Get current free space in the stack of current CPU core. Function calculates distance from current stack pointer to stack bottom of the current CPU core.

u32 StackFeed();

Feed stack of current CPU core with the food. The function fills the space between the current stack pointer and the bottom of the stack with the constant 0x600DF00D (HEX text "Good Food"). Function returns free space in the stack, or 0 on stack error. The function is

also called automatically at system startup for processor core 0, so it is not necessary to call it from the program.

u32 StackCheck();

Check stack limit of current CPU core. The function checks the used stack space. It scans the stack from the bottom address and checks the written constant 0x600DF00D until it encounters a changed stack area - thus checking how far down the program has gone during its operation (including interrupt handlers). The function will return the remaining stack reserve.

void __attribute__((noreturn)) panic(const char *fmt, ...);

Fatal error - display panic message and halt execution.

void __attribute__((noreturn)) __breakpoint();

Stop program execution and halt.

2.5. Divider - Integer Division And Multiplication

Files: `sdk_divider.h, sdk_divider.S` (RP2040 only)

Config: -

Used Cortex M0+ processor core does not contain instruction for integer division. For this reason, the RP2040 processor is equipped with a hardware divider, accessible via registers, with a required calculation time of 8 clock cycles. Each processor core has its own hardware divider. RP2350 does not contain this divider unit, it has divide instructions. SDK contains inline functions to simulate functions of the divider module.

The use of the hardware divider in the PicoLibSDK library is different from the original SDK library. If a division operation is used in the interrupt handler, the division operation performed in the main program could be disrupted. In the original SDK, this is handled by consistency in the order of operations and possibly saving the state of the divider during interrupt handling.

In the PicoLibSDK library, collisions with possible interrupts are resolved by consistently disabling interrupts while using the hardware divider. This may make the divider slightly slower to use, but it will make it much easier to use. This requires the programmer to consistently use either the library function to divide numbers, or to disable global interrupts during custom divider use.

The Divider library module, in addition to handling integer division, also includes functions for integer multiplication.

Direct access to the divider registers (volatile u32*)

<code>DIV_UDIVIDEND</code>	unsigned dividend 'p'
<code>DIV_UDIVISOR</code>	unsigned divisor 'q'
<code>DIV_SDIVIDEND</code>	signed dividend 'p'
<code>DIV_SDIVISOR</code>	signed divisor 'q'
<code>DIV_QUOTIENT</code>	result quotient = p / q
<code>DIV_REMAINDER</code>	result remainder = p % q

void DivStartS32(s32 p, s32 q);

p ... dividend
q ... divisor

Start signed division p/q. Interrupts must be disabled before the operation begins.

void DivStartU32(u32 p, u32 q);

p ... dividend
q ... divisor

Start unsigned division p/q. Interrupts must be disabled before the operation begins.

void DivWait();

Wait for divide to complete calculation (takes 8 clock cycles).

u32 DivRemainder();

Get result remainder after complete calculation. Retype result to s32 after signed division.

u32 DivQuotient();

Get result quotient after complete calculation. Retype result to s32 after signed division.

u64 DivResult();

Get full result after complete calculation. Quotient p/q will be in low 32 bits, remainder p%q will be in high 32 bits.

void DivSaveState(hw_divider_state_t* dst);

dst ... structure to save divider state

Save divider state (for calling core). Interrupt should be disabled.

void DivLoadState(const hw_divider_state_t* src);

src ... structure to save divider state

Load divider state (for calling core). Interrupt should be disabled.

s32 DivS32(s32 a, s32 b);

a ... signed dividend

b ... signed divisor

Divide signed s32. Result contains signed quotient s32 c=a/b. Function temporary disables interrupts to protect divider state.

u32 DivU32(u32 a, u32 b);

a ... unsigned dividend

b ... unsigned divisor

Divide unsigned u32. Result contains unsigned quotient u32 c=a/b. Function temporary disables interrupts to protect divider state.

s32 ModS32(s32 a, s32 b);

a ... signed dividend

b ... signed divisor

Modulo signed s32. Result contains signed remainder s32 d=a%b. Function temporary disables interrupts to protect divider state.

u32 ModU32(u32 a, u32 b);

a ... unsigned dividend

b ... unsigned divisor

Modulo unsigned u32. Result contains unsigned remainder u32 $d=a\%b$. Function temporary disables interrupts to protect divider state.

s32 DivModS32(s32 a, s32 b, s32* rem);

a ... signed dividend

b ... signed divisor

rem ... pointer to store remainder s32 $d=a\%b$

Divide modulo signed s32. Result contains signed quotient s32 $c=a/d$. Function temporary disables interrupts to protect divider state.

u32 DivModU32(u32 a, u32 b, u32* rem);

a ... unsigned dividend

b ... unsigned divisor

rem ... pointer to store remainder u32 $d=a\%b$

Divide modulo unsigned u32. Result contains unsigned quotient u32 $c=a/d$. Function temporary disables interrupts to protect divider state.

s64 DivS64(s64 a, s64 b);

a ... signed dividend

b ... signed divisor

Divide signed s64. Result contains signed quotient s64 $c=a/b$. Function temporary disables interrupts to protect divider state.

u64 DivU64(u64 a, u64 b);

a ... unsigned dividend

b ... unsigned divisor

Divide unsigned u64. Result contains unsigned quotient u64 $c=a/b$. Function temporary disables interrupts to protect divider state.

s64 ModS64(s64 a, s64 b);

a ... signed dividend

b ... signed divisor

Modulo signed s64. Result contains signed remainder s64 $d=a\%b$. Function temporary disables interrupts to protect divider state.

u64 ModU64(u64 a, u64 b);

a ... unsigned dividend

b ... unsigned divisor

Modulo unsigned u64. Result contains unsigned remainder u64 $d=a\%b$. Function temporary disables interrupts to protect divider state.

s64 DivModS64(s64 a, s64 b, s64* rem);

a ... signed dividend

b ... signed divisor

rem ... pointer to store remainder s64 $d=a\%b$

Divide modulo signed s64. Result contains signed quotient s64 $c=a/d$. Function temporary disables interrupts to protect divider state.

u64 DivModU64(u64 a, u64 b, u64* rem);

a ... unsigned dividend

b ... unsigned divisor

rem ... pointer to store remainder u64 $d=a\%b$

Divide modulo unsigned u64. Result contains unsigned quotient u64 $c=a/d$. Function temporary disables interrupts to protect divider state.

u64 UMul(u32 a, u32 b);

Unsigned multiply $u32*u32$ with result u64.

s64 SMul(s32 a, s32 b);

Signed multiply $s32*s32$ with result s64.

u64 Sqr(u32);

Square $u32*u32$ with result u64.

u64 UMul64(u64 a, u64 b);

Unsigned multiply $u64*u64$ with result u64.

s64 SMul64(s64 a, s64 b);

Signed multiply $s64*s64$ with result s64.

Bool IsPow2(u32 a);

Check if integer number is power of 2.

2.6. DMA - Direct Memory Access

Files: `sdk_dma.h`, `sdk_dma.c`

Config: `USE_DMA (default 1)`

The RP2040 DMA controller supports 12 independent channels, transferring data memory-to-peripheral, peripheral-to-memory or memory-to-memory, with element size 32, 16 or 8 bits. The RP2350 DMA controller supports 16 channels. Channels can be chained, where one channel configures another channel.

The DMA controller of RP2350 differs from the RP2040 in that it supports address decrementing in addition to automatic address incrementing.

The configuration registers can be accessed through 4 different aliases that allow another channel to trigger a transfer by accessing the register as needed. If a non-zero value is stored in the register with the trigger, a new transfer is started. Writing a null value does not start new transfer - this is useful for ending control block chains.

The read and write addresses are automatically shifted during the transfer. Their values can be read to determine the current DMA address of the transfer. The transfer counter is also automatically decremented during the transfer. By reading it, you can see how many items are still to be transferred. When the transfer is complete, the transfer counter contains 0. When a new transfer is started, the transfer counter is reloaded from the auxiliary register where the counter value was stored when it was last set.

There are 12 or 16 channels of DMA controller. It is recommended to schedule the use of each channel in the program configuration using `#define`. In addition, it is possible to use automatic allocation of DMA channels using the "claim" functions. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

Most functions have two forms. Functions can be addressed either by the DMA channel number (parameter "int dma") or by a pointer to the hardware interface ("dma_channel_hw_t* hw"). Functions in the second case are marked with "_hw". Addressing by channel number is easier to use, addressing by pointer can generate slightly better optimized code.

DMA_CHANNELS ... constant that defines the number of DMA channels (= 12 or 16)

DMA_IRQ_NUM ... number of IRQs (RP2040: 2, RP2350: 4)

DMA_TIMERS ... constant that defines the number of DMA timers (= 4)

DMA transfer size

DMA_SIZE_8 1 byte (8 bits)

DMA_SIZE_16 2 bytes (16 bits)

DMA_SIZE_32 4 bytes (32 bits)

Flags of DMA control and status register

DMA_CTRL_ERROR read or write error, channel halts on error and raises IRQ

DMA_CTRL_READ_ERROR	read error
DMA_CTRL_WRITE_ERROR	write error
DMA_CTRL_BUSY	DMA transfer is busy
DMA_CTRL_SNIFF	sniff CRC (or normal DMA transfer otherwise)
DMA_CTRL_BSWAP	swap order of bytes
DMA_CTRL QUIET	generate IRQ when NULL is written to a trigger (or generate IRQ at end of every transfer block otherwise)
DMA_CTRL_TREQ(dreq)	select transfer request signal DREQ_*
DMA_CTRL_TREQ_FORCE	permanent request (not using DREQ)
DMA_CTRL_CHAIN(chan)	trigger channel 0..11 or 0..15 when completed disable by setting to "this" channel (=default state)
DMA_CTRL_RING_WRITE	write addresses are wrapped (or read otherwise)
DMA_CTRL_RING_SIZE(order)	size order 1..15 of address wrap region (0=no wrap) Ring buffer must be aligned to wrap size (only lowest 'order' bits are changed).
DMA_CTRL_INC_WRITE	increment or decrement write address with each transfer
DMA_CTRL_INC_WRITE_REV	decrement write address with each transfer (only RP2350)
DMA_CTRL_INC_READ	increment or decrement read address with each transfer
DMA_CTRL_INC_READ_REV	decrement read address with each transfer (only RP2350)
DMA_CTRL_SIZE(size)	data transfer size DMA_SIZE_*
DMA_CTRL_HIGH_PRIORITY	channel is preferred during DMA scheduling round
DMA_CTRL_EN	channel is enabled and responds to trigger

DMA CRC method

DMA_CRC_CRC32	CRC-32
DMA_CRC_CRC32REV	CRC-32 bit reversed data
DMA_CRC_CRC16	CRC-16-CCITT
DMA_CRC_CRC16REV	CRC-16-CCITT bit reversed data
DMA_CRC_XOR	XOR reduction over all data
DMA_CRC_SUM	simple 32-bit sum

Auxiliary flags used by the DMA_CRC() function

DMA_CRC_INV	flag to invert result (used as parameter of DMA_CRC())
DMA_CRC_REV	flag to bit-reverse result (used as parameter of DMA_CRC())

Flags of DMA sniffer control register (calculating CRC)

DMA_SNIFF_INV	inverted result on read (bitwise complement)
DMA_SNIFF_REV	bit-reverse result on read

DMA_SNIFF_BSWAP	byte reverse sniffed data (cancel effect if DMA-BSWAP is set)
DMA_SNIFF_CRC(crc)	CRC method DMA_CRC_*
DMA_SNIFF_CHAN(chan)	used DMA channel for sniffer 0..11 or 0..15
DMA_SNIFF_EN	enable sniffer

DMA_TEMP_CHAN();

Macro returns DMA channel that can be reserved and used for some system operations such as DMA_MemCopy(), DMA_MemFill() or DMA_CRC(). CPU core 0 uses channel 11, CPU code 1 uses channel 10. If the programmer does not use those functions, he can use those DMA channels for other purposes.

dma_channel_hw_t* DMA_GetHw(int dma);

dma ... DMA channel 0..11 (0..15)

Get DMA channel hardware interface from DMA channel index.

u8 DMA_GetInx(const dma_channel_hw_t* hw);

hw ... pointer to DMA channel hardware interface

Get DMA channel index from DMA channel hardware interface.

volatile u32* DMA_Alias(int dma, int alias);

dma ... DMA channel 0..11 (0..15)

alias ... DMA alias 0..3

Get DMA base address of alias 0..3 of channel 0..11.

Claim DMA channel

Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not. Functions are not atomic safe (not recommended to be used in both cores or in IRQ at the same time).

void DMA_Claim(int dma);

dma ... DMA channel 0..11 (0..15)

Claim DMA channel (mark it as used).

void DMA_ClaimMask(int mask);

mask ... mask od DMA channels (bits 0..11 or 0..15)

Claim DMA channels with mask (mark them as used).

void DMA_Unclaim(int dma);

dma ... DMA channel 0..11 (0..15)

Unclaim DMA channel (mark it as not used).

void DMA_UnclaimMask(int mask);

mask ... mask od DMA channels (bits 0..11 or 0..15)

Unclaim DMA channels with mask (mark them as not used).

Bool DMA_IsClaimed(int dma);

dma ... DMA channel 0..11 (0..15)

Check if DMA channel is claimed.

s8 DMA_ClaimFree(void);

Claim free unused DMA channel (returns -1 on error).

Claim DMA timer

Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not. Functions are not atomic safe (not recommended to be used in both cores or in IRQ at the same time).

void DMA_TimerClaim(int timer);

timer ... DMA timer 0..3

Claim DMA timer (mark it as used).

void DMA_TimerUnclaim(int timer);

timer ... DMA timer 0..3

Unclaim DMA timer (mark it as not used).

Bool DMA_TimerIsClaimed(int timer);

timer ... DMA timer 0..3

Check if DMA timer is claimed.

s8 DMA_TimerClaimFree(void);

Claim free unused DMA timer (returns -1 on error).

DMA configuration setup structure (u32 control word)

After setup, write configuration word using DMA_SetCtrl() or DMA_SetCtrlTrig() function.

u32 DMA_CfgDef(int dma);

dma ... DMA channel 0..11 (0..15)

Get default configuration word.

void DMA_CfgReadInc(u32* cfg, int read_inc);

cfg ... pointer to configuration word

read_inx ... read increment DMA_DIR_*

Config setup read increment. Default True.

void DMA_CfgWriteInc(u32* cfg, int write_inc);

cfg ... pointer to configuration word

write_inx ... write increment DMA_DIR_*

Config setup write increment. Default False.

void DMA_CfgDreq(u32* cfg, int dreq);

cfg ... pointer to configuration word

dreq ... data request channel of port DREQ_*

Config setup data request DREQ_*. Default **DREQ_FORCE** = permanent.

void DMA_CfgChain(u32* cfg, int dma);

cfg ... pointer to configuration word

dma ... dma channel to chain to

Config setup chain. To disable, set to the same channel number (= default).

void DMA_CfgSize(u32* cfg, int size);

cfg ... pointer to configuration word

size ... transfer size DMA_SIZE_*

Config setup transfer size DMA_SIZE_*. Default **DMA_SIZE_32**.

void DMA_CfgRing(u32* cfg, Bool write, int size);

cfg ... pointer to configuration word

write ... True to apply ring to write address, False to apply ring to read address

size ... order of ring size 1..15 (0=disabled, 1=2 byte, 2=4 bytes, ... 15=32 KB)

Config setup ring. Default 0 = disabled; apply on read.

void DMA_CfgBSwap(u32* cfg, Bool bswap);

cfg ... pointer to configuration word

bswap ... True to do byte swap

Config setup byte swap. Default False.

void DMA_CfgQuiet(u32* cfg, Bool quiet);

cfg ... pointer to configuration word

quiet ... True to quiet (no interrupt after every transfer)

Config setup quiet. Default False.

```
void DMA_CfgHighPrior(u32* cfg, Bool high);
```

cfg ... pointer to configuration word

high ... True to high priority

Config setup high priority. Default False.

```
void DMA_CfgEnable(u32* cfg, Bool enable);
```

cfg ... pointer to configuration word

enable ... True to enable DMA channel

Config setup enable. Default True.

```
void DMA_CfgSniff(u32* cfg, Bool sniff);
```

cfg ... pointer to configuration word

sniff ... True to enable sniff

Config setup sniff enable. Default False.

DMA status

```
Bool DMA_IsError(int dma);
```

```
Bool DMA_IsError_hw(dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Check DMA read or write error.

```
Bool DMA_IsReadError(int dma);
```

```
Bool DMA_IsReadError_hw(dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Check DMA read error.

```
Bool DMA_IsWriteError(int dma);
```

```
Bool DMA_IsWriteError_hw(dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Check DMA write error.

```
void DMA_ClearError(int dma);
```

```
void DMA_ClearError_hw(dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Clear DMA error flags.

```
Bool DMA_IsBusy(int dma);
Bool DMA_IsBusy_hw(const dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Check if DMA is busy (can stay busy if paused using enable flag).

DMA registers

```
void* DMA_GetRead(int dma);
void* DMA_GetRead_hw(const dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Get current DMA read address.

```
void DMA_SetRead(int dma, const volatile void* addr);
void DMA_SetRead_hw(dma_channel_hw_t* hw, const volatile void* addr);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

addr ... DMA read address (must be aligned to current element size)

Set DMA read address without triggering the transfer.

```
void DMA_SetReadTrig(int dma, const volatile void* addr);
void DMA_SetReadTrig_hw(dma_channel_hw_t* hw, const volatile void* addr);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

addr ... DMA read address (must be aligned to current element size)

Set DMA read address and trigger the transfer. Zero value does not trigger the transfer.

```
void* DMA_GetWrite(int dma);
void* DMA_GetWrite_hw(const dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Get current DMA write address.

```
void DMA_SetWrite(int dma, volatile void* addr);
void DMA_SetWrite_hw(dma_channel_hw_t* hw, volatile void* addr);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

addr ... DMA write address (must be aligned to current element size)

Set DMA write address without triggering the transfer.

```
void DMA_SetWriteTrig(int dma, volatile void* addr);
void DMA_SetWriteTrig_hw(dma_channel_hw_t* hw, volatile void* addr);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

addr ... DMA write address (must be aligned to current element size)

Set DMA write address and trigger the transfer. Zero value does not trigger the transfer.

```
u32 DMA_GetCount(int dma);
u32 DMA_GetCount_hw(const dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Get current DMA transfer count (in number of elements, not bytes).

```
void DMA_SetCount(int dma, u32 count);
void DMA_SetCount_hw(dma_channel_hw_t* hw, u32 count);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

count ... count of elements to transfer

Set DMA transfer count without triggering the transfer. On RP2350, count can be combined with flags DMA_COUNT_TRIGGER or DMA_COUNT_ENDLESS.

```
void DMA_SetCountTrig(int dma, u32 count);
void DMA_SetCountTrig_hw(dma_channel_hw_t* hw, u32 count);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

count ... count of elements to transfer

Set DMA transfer count and trigger the transfer. Zero value does not trigger the transfer. On RP2350, count can be combined with flags DMA_COUNT_TRIGGER or DMA_COUNT_ENDLESS.

```
u32 DMA_Next(int dma);
```

dma ... DMA channel 0..11 (0..15)

Get DMA next transfer count. This entry represents the contents of the auxiliary register where the last entered counter value is stored and from which the counter value is restored when starting a new transfer.

```
u32 DMA_GetCtrl(int dma);
u32 DMA_GetCtrl_hw(const dma_channel_hw_t* hw);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Get DMA control register. Value is combination of **DMA_CTRL_*** flags.

```
void DMA_SetCtrl(int dma, u32 ctrl);
void DMA_SetCtrl_hw(dma_channel_hw_t* hw, u32 ctrl);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

ctrl ... new value of control register

Set DMA control register without triggering the transfer. Value is combination of **DMA_CTRL_*** flags.

```
void DMA_SetCtrlTrig(int dma, u32 ctrl);
void DMA_SetCtrlTrig_hw(dma_channel_hw_t* hw, u32 ctrl);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

ctrl ... new value of control register

Set DMA control register and trigger the transfer. Value is combination of **DMA_CTRL_*** flags.

DMA setup

```
void DMA_Config(int dma, const volatile void* src, volatile void* dst, int count,
u32 ctrl);
```

```
void DMA_Config_hw(dma_channel_hw_t* hw, const volatile void* src, volatile
void* dst, u32 count, u32 ctrl);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

src ... source address (must be aligned to element size)

dst ... destination address (must be aligned to element size)

count ... number of transfers

ctrl ... control word

Set DMA config without triggering the transfer. Control word is combination of **DMA_CTRL_*** flags. On RP2350, count can be combined with flags **DMA_COUNT_TRIGGER** or **DMA_COUNT_ENDLESS**.

```
void DMA_ConfigTrig(int dma, const volatile void* src, volatile void* dst, int
count, u32 ctrl);
```

```
void DMA_ConfigTrig_hw(dma_channel_hw_t* hw, const volatile void* src,
volatile void* dst, u32 count, u32 ctrl);
```

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

src ... source address (must be aligned to element size)

dst ... destination address (must be aligned to element size)

count ... number of transfers

ctrl ... control word

Set DMA config and trigger the transfer. Control word is combination of **DMA_CTRL_*** flags. On RP2350, count can be combined with flags DMA_COUNT_TRIGGER or DMA_COUNT_ENDLESS.

void DMA_Disable(int dma);
void DMA_Disable_hw(dma_channel_hw_t* hw);

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Disable DMA channel (pause - transfer will be paused but not aborted, busy will stay active).

void DMA_Enable(int dma);
void DMA_Enable_hw(dma_channel_hw_t* hw);

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Enable DMA channel (unpause - continue transfers).

Bool DMA_IsEnabled(int dma);
Bool DMA_IsEnabled_hw(const dma_channel_hw_t* hw);

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Check if DMA is enabled.

void DMA_SetChain(int dma, int chain);
void DMA_SetChain_hw(dma_channel_hw_t* hw, int chain);

dma ... DMA channel 0..11 (0..15)

chain ... DMA chain channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Set chain. Set to itself to disable.

int DMA_Chain(int dma);
int DMA_Chain_hw(dma_channel_hw_t* hw);

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware int

Get chain channel. Returns itself if disabled.

u32 DMA_SniffCtrl();

Get DMA sniff control register. Sniff register is used to calculate CRC. Only one DMA channel can be used for sniffer. Returned word is combination of **DMA_SNIFF_*** flags.

void DMA_SetSniffCtrl(u32 ctrl);

ctrl ... control word

Set DMA sniff control register. Sniff register is used to calculate CRC. Only one DMA channel can be used for sniffer. Control word is combination of **DMA_SNIFF_*** flags.

u32 DMA_SniffData();

Get sniff control data. Returned value is result value of CRC.

void DMA_SetSniffData(u32 data);

data ... data to initialize CRC calculation

Set sniff control data. Value is initial value of CRC.

void DMA_StartMask(u32 mask);

mask ... bit mask of channels 0..11 (0..15)

Start multiply DMA transfers with mask.

void DMA_Start(int dma);

dma ... DMA channel 0..11 (0..15)

Start DMA transfer of channel.

void DMA_Wait(int dma);

void DMA_Wait_hw(const dma_channel_hw_t* hw);

dma ... DMA channel 0..11 (0..15)

hw ... pointer to DMA channel hardware interface

Wait DMA for completion.

void DMA_Abort(int dma);

dma ... DMA channel 0..11 (0..15)

Abort DMA transfer. Recommended to disable IRQ before aborting DMA to avoid interrupting the transfer at the moment of already activated but not completed IRQ. The IRQ handler could come after the transfer is completed.

DMA transfers

void DMA_MemCopy32(int dma, u32* dst, const u32* src, int count);

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u32 (must be aligned to u32)

src ... pointer to first source u32 (must be aligned to u32)

count ... number of u32 elements to transfer

Perform 32-bit DMA transfer from memory to memory, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 2 us per 1 KB.

```
void DMA_MemCopy16(int dma, u16* dst, const u16* src, int count);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u16 (must be aligned to u16)

src ... pointer to first source u16 (must be aligned to u16)

count ... number of u16 elements to transfer

Perform 16-bit DMA transfer from memory to memory, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 4 us per 1 KB.

```
void DMA_MemCopy8(int dma, u8* dst, const u8* src, int count);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u8

src ... pointer to first source u8

count ... number of u8 elements to transfer

Perform 8-bit DMA transfer from memory to memory, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 8 us per 1 KB.

```
void DMA_MemCopy(volatile void* dst, const volatile void* src, int num);
```

dst ... pointer to destination

src ... pointer to source

num ... number of bytes

Perform optimised fast DMA transfer from memory to memory, waiting for completion. Can be used simultaneously in both CPUs, but not simultaneously in an interrupt. Uses DMA_TEMP_CHAN() channel.

```
void DMA_MemFill32(int dma, u32* dst, const volatile u32* data, int count);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u32 (must be aligned to u32)

data ... pointer to 32-bit sample to fill (must be aligned to u32)

count ... number of u32 elements to fill

Fill memory with 32-bit sample using 32-bit DMA, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 2 us per 1 KB.

```
void DMA_MemFill16(int dma, u16* dst, const volatile u16* data, int count);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u16 (must be aligned to u16)

data ... pointer to 16-bit sample to fill (must be aligned to u16)

count ... number of u16 elements to fill

Fill memory with 16-bit sample using 16-bit DMA, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 4 us per 1 KB.

```
void DMA_MemFill8(int dma, u8* dst, const volatile u8* data, int count);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u8

data ... pointer to 8-bit sample to fill

count ... number of u8 elements to fill

Fill memory with 8-bit sample using 8-bit DMA, not waiting for completion (wait with DMA_Wait() function). Transfer speed is 8 us per 1 KB.

```
void DMA_MemFillDW(volatile void* dst, u32 data, int num);
```

dst ... pointer to destination

data ... 32-bit sample to fill

num ... number of bytes (last sample may be cropped)

Fill memory with 32-bit double word sample using optimised DMA transfer, waiting for completion. Can be used simultaneously in both CPUs, but not simultaneously in an interrupt. Transfer speed is 2 us per 1 KB. Uses DMA_TEMP_CHAN() channel.

```
void DMA_MemFillW(volatile void* dst, u16 data, int num);
```

dst ... pointer to destination

data ... 16-bit sample to fill

num ... number of bytes (last sample may be cropped)

Fill memory with 16-bit word sample using optimised DMA transfer, waiting for completion. Can be used simultaneously in both CPUs, but not simultaneously in an interrupt. Transfer speed is 2 us per 1 KB. Uses DMA_TEMP_CHAN() channel.

```
void DMA_MemFill(volatile void* dst, u8 data, int num);
```

dst ... pointer to destination

data ... 8-bit sample to fill

num ... number of bytes

Fill memory with 8-bit byte sample using optimised DMA transfer, waiting for completion. Can be used simultaneously in both CPUs, but not simultaneously in an interrupt. Transfer speed is 2 us per 1 KB. Uses DMA_TEMP_CHAN() channel.

```
void DMA_ToPort(int dma, volatile u32* port, const u32* src, int count, int dreq);
```

dma ... DMA channel 0..11 (0..15)

port ... pointer to u32 port (must be aligned to u32)

src ... pointer to first source u32 (must be aligned to u32)

count ... number of u32 elements to transfer

dreq ... data request channel of port DREQ_*

DMA send data from memory to port, not waiting for completion (wait with DMA_Wait() function).

```
void DMA_FromPort(int dma, u32* dst, const volatile u32* port, int count, int dreq);
```

dma ... DMA channel 0..11 (0..15)

dst ... pointer to first destination u32 (must be aligned to u32)

port ... pointer to u32 port (must be aligned to u32)

count ... number of u32 elements to transfer

dreq ... data request channel of port DREQ_*

DMA receive data from port to memory, not waiting for completion (wait with DMA_Wait() function).

```
void DMA_IRQ0EnableMask(u32 mask);
```

```
void DMA_IRQ1EnableMask(u32 mask);
```

```
void DMA_IRQ2EnableMask(u32 mask);
```

```
void DMA_IRQ3EnableMask(u32 mask);
```

mask .. bit mask of channels 0..11 (0..15)

Enable interrupt from DMA channels for **IRQ_DMA_0..3** masked (set bits 0..15 to enable channel 0..15). IRQ 2 and 3 is available only on RP2350.

```
void DMA_IRQ0Enable(int dma);
```

```
void DMA_IRQ1Enable(int dma);
```

```
void DMA_IRQ2Enable(int dma);
```

```
void DMA_IRQ3Enable(int dma);
```

dma ... DMA channel 0..11 (0..15)

Enable interrupt from DMA channel for **IRQ_DMA_0..3**. IRQ 2 and 3 is available only on RP2350.

```
void DMA_IRQ0DisableMask(u32 mask);
```

```
void DMA_IRQ1DisableMask(u32 mask);
```

```
void DMA_IRQ2DisableMask(u32 mask);
```

```
void DMA_IRQ3DisableMask(u32 mask);
```

mask .. bit mask of channels 0..11 (0..15)

Disable interrupt from DMA channels for **IRQ_DMA_0..3** masked (set bits 0..15 to disable channel 0..15). IRQ 2 and 3 is available only on RP2350.

```
void DMA_IRQ0Disable(int dma);
```

```
void DMA_IRQ1Disable(int dma);
```

```
void DMA_IRQ2Disable(int dma);
```

```
void DMA_IRQ3Disable(int dma);
```

dma ... DMA channel 0..11 (0..15)

Disable interrupt from DMA channel for **IRQ_DMA_0..3**. IRQ 2 and 3 is available only on RP2350.

```
Bool DMA_IRQ0IsPending(int dma);
Bool DMA_IRQ1IsPending(int dma);
Bool DMA_IRQ2IsPending(int dma);
Bool DMA_IRQ3IsPending(int dma);
```

dma ... DMA channel 0..11 (0..15)

Check if DMA interrupt request **IRQ_DMA_0..3** is pending. IRQ 2 and 3 is available only on RP2350.

```
void DMA_IRQ0Clear(int dma);
void DMA_IRQ1Clear(int dma);
void DMA_IRQ2Clear(int dma);
void DMA_IRQ3Clear(int dma);
```

dma ... DMA channel 0..11 (0..15)

Clear DMA interrupt request **IRQ_DMA_0..3**. IRQ 2 and 3 is available only on RP2350.

```
u32 DMA_CRC(int mode, u32 init, int dma, const void* data, int num);
```

mode ... CRC mode **DMA_CRC_*** (including **DMA_CRC_INV** and **DMA_CRC_REV**)

init ... init value

dma ... DMA channel 0..11 (0..15)

data ... pointer to data

num ... number of bytes

Calculate optimised CRC checksum using DMA (wait for completion). Calculation speed is 2 us per 1 KB.

```
u32 DMA_SUM(int mode, u32 init, int dma, const void* data, int num, int size);
```

mode ... CRC mode **DMA_CRC_SUM** or other

(including **DMA_CRC_INV** and **DMA_CRC_REV**)

init ... init value

dma ... DMA channel 0..11 (0..15)

data ... pointer to data (must be aligned to the u8/u16/u32 entry)

num ... number of u8/u16/u32 entries

size ... size of one entry **DMA_SIZE_*** (u8/u16/u32)

Calculate checksum using DMA, on aligned data (wait for completion). Calculation speed is 32-bit 2 us per 1 KB, 16-bit 4 us per 1 KB, 8-bit 8 us per 1 KB.

```
void DMA_SetTimer(int timer, int x, int y);
```

timer ... index of timer 0..3

x ... numerator (0 = timer is OFF)

y ... denominator

Set DMA fractional timer to trigger DMA on rate X/Y***CLK_SYS**. The DMA channel transfer can be triggered by one of the 4 DMA timers, by setting the interrupt source in the CONTROL register using **DMA_CTRL_TREQ(dreq)** with the value **DREQ_TIMER0** to **DREQ_TIMER3**.

2.7. Double - Double-Floating-Point

Files: `sdk_double.h, sdk_double.c, sdk_double_asm.S`

Config: `USE_DOUBLE (default 1)`

The RP2040 processor does not have floating-point hardware support and therefore it is provided in software. The RP2350 in ARM mode includes a coprocessor for accelerating double-precision operations. The computations are not fully hardware accelerated, but are significantly faster than the software implementation. The RP2350 in RISC-V mode calculates double-precision operations in software.

Auxiliary functions

`double copysign(double num, double sign);`
`double copysignd(double num, double sign);`

Compose floating point with magnitude of 'num' and sign of 'sign'.

`Bool isintd(double num);`

Check if number is an integer.

`Bool isoddintd(double num);`

Check if number is odd integer (must be an integer ..., -3, -1, 1, 3, 5,..).

`Bool ispow2d(double num);`

Check if number is power of 2.

`double Idexp(double num, int exp);`
`double Idexpd(double num, int exp);`

Multiply number by power of 2 ($\text{num} * 2^{\text{exp}}$).

Basic arithmetic

`double dadd(double x, double y);`

Addition, $x + y$.

`double dsub(double x, double y);`

Subtraction, $x - y$.

`double dmul(double x, double y);`

Multiplication, $x * y$.

double dsqr(double x);

Square, x^2 .

double ddiv(double x, double y);

Division, x / y .

double drec(double x);

Reciprocal $1 / x$.

double fmod(double x, double y);

double fmodd(double x, double y);

Get remainder of division x/y , rounded towards zero. $fmod(x, y) = x - tquot*y$, where $tquot$ is truncated (i.e. rounded towards zero) result of x/y .

double remquo(double x, double y, int* quo);

double remquod(double x, double y, int* quo);

Compute remainder and quotient of division x/y , rounded towards the even number. $remainder(x, y) = x - rquot*y$, where $rquot$ is result of x/y , rounded towards the nearest integral.

double remainder(double x, double y);

double remainderd(double x, double y);

Get remainder of division x/y , rounded towards the even number. $remainder(x, y) = x - rquot*y$, where $rquot$ is result of x/y , rounded towards the nearest integral.

double drem(double x, double y);

double dremd(double x, double y);

Obsolete synonym of functions remainder and remainderd.

Comparison

s8 dcmp(double x, double y);

Compare, $x ? y$. Returns: 0 if $x==y$, -1 if $x < y$, +1 if $x > y$.

Bool dcmpeq(double x, double y);

Compare if $x==y$.

Bool dcmplt(double x, double y);

Compare if $x < y$.

Bool dcmple(double x, double y);

Compare if $x \leq y$.

Bool dcmpge(double x, double y);

Compare if $x \geq y$.

Bool dcmpgt(double x, double y);

Compare if $x > y$.

Bool dcmpun(double x, double y);

Check if comparison is unordered (either input is NaN).

Convert integer to double

double i2d(s32 num);

double int2double(s32 num);

Convert signed int to double.

double ui2d(u32 num);

double uint2double(u32 num);

Convert unsigned int to double.

double l2d(s64 num);

double int642double(s64 num);

Convert 64-bit signed int to double.

double ul2d(s64 num);

double uint642double(u64 num);

Convert 64-bit unsigned int to double.

double fix2double(s32 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert signed fixed point to double.

double ufix2double(u32 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert unsigned fixed point to double.

double fix642double(s64 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert 64-bit signed fixed point to double.

double ufix642double(u64 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert 64-bit unsigned fixed point to double.

Convert double to integer

s32 d2iz(double num);
s32 double2int_z(double num);

Convert double to signed int, rounding to zero (C-style int conversion).

s32 d2i(double num);
s32 double2int(double num);

Convert double to signed int, rounding down.

u32 d2ui(double num);
u32 double2uint(double num);

Convert double to unsigned int, rounding down.

s64 d2lz(double num);
s64 double2int64_z(double num);

Convert double to 64-bit signed int, rounding to zero (C-style int conversion).

s64 d2l(double num);
s64 double2int64(double num);

Convert double to 64-bit signed int, rounding down.

u64 d2ul(double num);
u64 double2uint64(double num);

Convert double to 64-bit unsigned int.

s32 double2fix(double num, int e);
e ... number of bits of fractional part (binary exponent)

Convert double to signed fixed point, rounding down.

u32 double2ufix(double num, int e);
e ... number of bits of fractional part (binary exponent)

Convert double to unsigned fixed point, rounding down.

s64 double2fix64(double num, int e);
e ... number of bits of fractional part (binary exponent)

Convert double to 64-bit signed fixed point, rounding down.

```
u64 double2ufix64(double num, int e);
```

e ... number of bits of fractional part (binary exponent)

Convert double to 64-bit unsigned fixed point.

Rounding

```
double trunc(double num);  
double truncd(double num);
```

Round number towards zero.

```
double round(double num);  
double roundd(double num);
```

Round number to nearest integer.

```
double floor(double num);  
double floord(double num);
```

Round number down to integer.

```
double ceil(double num);  
double ceild(double num);
```

Round number up to integer.

Scientific functions

```
double sqrt(double x);  
double sqrt(double x);
```

Square root.

```
double deg2rad(double x);
```

Convert degrees to radians.

```
double rad2deg(double x);
```

Convert radians to degrees.

```
double sin(double x);  
double sind(double x);
```

Sine in radians.

```
double sin_deg(double x);
```

Sine in degrees.

```
double cos(double x);
double cosd(double x);
```

Cosine in radians.

```
double cos_deg(double x);
```

Cosine in degrees.

```
void sincos(double x, double* psin, double* pcos);
void sincosd(double x, double* psin, double* pcos);
```

Sine-cosine in radians.

```
void sincos_deg(double x, double* psin, double* pcos);
```

Sine-cosine in degrees.

```
double tan(double x);
double tand(double x);
```

Tangent in radians.

```
double tan_deg(double x);
```

Tangent in degrees.

```
double cotan(double x);
```

Cotangent in radians.

```
double cotan_deg(double x);
```

Cotangent in degrees.

```
double asin(double x);
double asind(double x);
```

Arc sine in radians.

```
double asin_deg(double x);
```

Arc sine in degrees.

```
double acos(double x);
double acosd(double x);
```

Arc cosine in radians.

```
double acos_deg(double x);
```

Arc cosine in degrees.

```
double atan(double x);  
double atand(double x);
```

Arc tangent in radians.

```
double atan_deg(double x);
```

Arc tangent in degrees.

```
double acotan(double x);
```

Arc cotangent in radians.

```
double acotan_deg(double x);
```

Arc cotangent in degrees.

```
double atan2(double y, double x);  
double atan2d(double y, double x);
```

Arc tangent of y/x in radians.

```
double atan2_deg(double y, double x);
```

Arc tangent of y/x in degrees.

```
double sinh(double x);  
double sinhd(double x);
```

Hyperbolic sine.

```
double cosh(double x);  
double coshd(double x);
```

Hyperbolic cosine.

```
double tanh(double x);  
double tanhd(double x);
```

Hyperbolic tangent.

```
double asinh(double x);  
double asinhd(double x);
```

Inverse hyperbolic sine.

```
double acosh(double x);  
double acoshd(double x);
```

Inverse hyperbolic cosine.

```
double atanh(double x);
double atanhd(double x);
```

Inverse hyperbolic tangent.

```
double exp(double x);
double expd(double x);
```

Natural exponent.

```
double log(double x);
double logd(double x);
```

Natural logarithm.

```
double exp2(double x);
double exp2d(double x);
```

Exponent with base 2.

```
double log2(double x);
double log2d(double x);
```

Logarithm with base 2.

```
double exp10(double x);
double exp10d(double x);
```

Exponent with base 10.

```
double log10(double x);
double log10d(double x);
```

Logarithm with base 10.

```
double expm1(double x);
double expm1d(double x);
```

$\exp(x) - 1$

```
double log1p(double x);
double log1pd(double x);
```

$\log(x + 1)$ This function can be very inaccurate

```
double fma(double x, double y, double z);
double fmadd(double x, double y, double z);
```

$x * y + z$

```
double powint(double x, int y);  
double powintd(double x, int y);
```

Power by integer, x^y .

```
double pow(double x, double y);  
double powd(double x, double y);
```

Power x^y .

```
double hypot(double x, double y);  
double hypotd(double x, double y);
```

Square root of sum of squares (hypotenuse), $\sqrt{x^2 + y^2}$.

```
double cbrt(double x);  
double cbrnd(double x);
```

Cube root, $\sqrt[3]{x}$, $x^{(1/3)}$.

```
double absd(double x);
```

Absolute value.

```
float d2f(double num);  
float double2float(double num);
```

Convert double to float.

2.8. FIFO - Inter-Core FIFO, Mailboxes

Files: `sdk_fifo.h`, `sdk_fifo.c`

Config: `USE_FIFO` (default 1)

RP2040/RP2350 contains two FIFOs (mailboxes) for passing data, messages or ordered events between the two cores. Each FIFO is 32 bits wide, and eight entries deep. One of the FIFOs can only be written by core 0, and read by core 1. The other can only be written by core 1, and read by core 0. The inter-processor FIFOs and the Cortex-M0+ event signals are used by the bootrom `wait_for_vector()` routine, where core 1 remains in a sleep state until it is woken, and provided with its initial stack pointer, entry point and vector table through the FIFO. For this reason, the programmer should not use the inter-core FIFO unless he is sufficiently familiar with the communication used by the bootrom.

Bool FifoReadReady();

Check if inter-core RX FIFO is ready to read.

Bool FifoWriteReady();

Check if inter-core TX FIFO is ready to write.

void FifoWrite(u32 data);

Write message to other core. Wait if not ready.

Bool FifoWriteTime(u32 data, u32 us);

Write message to other core with time-out in [us]. Wait if not ready, max. 71 minutes, returns False on time-out.

u32 FifoRead();

Read message from other core. Wait if not ready.

Bool FifoReadTime(u32* data, u32 us);

Read message from other core with time-out in [us]. Wait if not ready, max. 71 minutes, returns False on time-out.

void FifoHandshake(u32 code);

FIFO handshake - send code and wait for same response.

u32 FifoState();

Get FIFO status. Returns combination of flags:

FIFO_ROE sticky flag, RX FIFO was read when empty (underflow error)

FIFO_WOF sticky flag, TX FIFO was written when full (overflow error)

FIFO_RDY this core's TX FIFO is not full (ready for write)

FIFO_VLD this core's RX FIFO is not empty (ready for read)

void FifoReadFlush();

Flush all read messages.

void FifoClear();

Clear FIFO interrupt request (clear **FIFO_ROE** and **FIFO_WOF** error flags).

2.9. Flash Memory

Files: `sdk_flash.h`, `sdk_flash.c`

Config: `USE_FLASH` (default 1)

External Flash is accessed via the QSPI interface using the execute-in-place (XIP) hardware. This allows an external flash memory to be addressed and accessed by the system as though it were internal memory. Bus reads to a 16MB memory window starting at 0x10000000 are translated into a serial flash transfer.

The external Flash memory can be reprogrammed with a program. The condition is that the program controlling the programming must not run from the Flash memory, including interrupt handlers or DMA transfers. The following functions ensure that the CPU Core 0 program does not run from Flash memory and that interrupts are disabled during programming. The programmer must ensure that neither the Core 1 program nor DMA transfers run from Flash memory before using the functions.

void FlashErase(u32 addr, u32 count);

addr ... start address to erase (offset from start of flash `XIP_BASE`)

count ... number of bytes to erase (multiple of 4 KB `FLASH_SECTOR_SIZE`)

Erase flash memory. Clearing the flash memory fills the specified area with 0xFF constant. Start address to erase is offset relative to start of flash memory 0x10000000 (`XIP_BASE`). Start address must be aligned to 4 KB (sector size `FLASH_SECTOR_SIZE`). Length of data to erase must be multiple of 4 KB (sector size `FLASH_SECTOR_SIZE`). If core 1 is running, lockout it (LockoutStart) or reset it!

void FlashProgram(u32 addr, const u8* data, u32 count);

addr ... start address to program (offset from start of flash `XIP_BASE`)

data ... pointer to source data to program (must be in RAM)

count ... number of bytes to program (multiple of 256 B `FLASH_PAGE_SIZE`)

Program flash memory. Start address to program is offset relative to start of flash memory 0x10000000 (`XIP_BASE`). Start address must be aligned to 256 B (page size `FLASH_PAGE_SIZE`). Length of data to erase must be multiple of 256 B (page size `FLASH_PAGE_SIZE`). If core 1 is running, lockout it (LockoutStart) or reset it!

void FlashCmd(const u8* txbuf, u8* rxbuf, u32 count);

txbuf ... pointer to byte buffer which will be transmitted to the flash

rxbuf ... pointer to byte buffer where data receive from the flash

count ... length in bytes of txbuf and of rxbuf

Execute flash command. If core 1 is running, lockout it or reset it!

void FlashLoadInfo();

Load flash info. Called during system startup. If core 1 is running, lockout it or reset it!

2.10. Float - Single-Floating-Point

Files: `sdk_float.h`, `sdk_float.c`, `sdk_float_asm.S`

Config: `USE_FLOAT (default 1)`

The RP2040 processor does not have floating-point hardware support and therefore it is provided in software. The RP2350 in ARM mode has hardware support for single-floating point operations. RP2350 in RISC-V mode calculates single-precision operations in software.

void FloatInit();

Initialize floating-point service. This function is automatically called during application startup from the internal `RuntimelInit()` function.

Auxiliary functions

float copysignf(float num, float sign);

Compose floating point with magnitude of 'num' and sign of 'sign'.

Bool isintf(float num);

Check if number is an integer.

Bool isoddintf(float num);

Check if number is odd integer (must be an integer ..., -3, -1, 1, 3, 5,...).

Bool ispow2f(float num);

Check if number is power of 2.

float Idexpf(float num, int exp);

Multiply number by power of 2 ($\text{num} * 2^{\text{exp}}$).

Basic arithmetic

float fadd(float x, float y);

Addition, $x + y$.

float fsub(float x, float y);

Subtraction, $x - y$.

float fmul(float x, float y);

Multiplication, $x * y$.

float fsqr(float x);

Square, x^2 .

float fdiv(float x, float y);

Division, x / y .

float frec(float x);

Reciprocal $1 / x$.

float fmodf(float x, float y);

Get remainder of division x/y , rounded towards zero. $fmod(x, y) = x - tquot*y$, where $tquot$ is truncated (i.e. rounded towards zero) result of x/y .

float remquof(float x, float y, int* quo);

Compute remainder and quotient of division x/y , rounded towards the even number. $remainder(x, y) = x - rquot*y$, where $rquot$ is result of x/y , rounded towards the nearest integral.

float remainderf(float x, float y);

Get remainder of division x/y , rounded towards the even number. $remainder(x, y) = x - rquot*y$, where $rquot$ is result of x/y , rounded towards the nearest integral.

float dremf(float x, float y);

Obsolete synonym of function `remainderf`.

Comparison

s8 fcmp(float x, float y);

Compare, $x ? y$. Returns: 0 if $x==y$, -1 if $x < y$, +1 if $x > y$.

Bool fcmpeq(float x, float y);

Compare if $x == y$.

Bool fcmplt(float x, float y);

Compare if $x < y$.

Bool fcmple(float x, float y);

Compare if $x \leq y$.

Bool fcmpge(float x, float y);

Compare if $x \geq y$.

Bool fcmpgt(float x, float y);

Compare if x>y.

Bool fcmpun(float x, float y);

Check if comparison is unordered (either input is NaN).

Convert integer to float

float i2f(s32 num);

float int2float(s32 num);

Convert signed int to float.

float ui2f(u32 num);

float uint2float(u32 num);

Convert unsigned int to float.

float l2f(s64 num);

float int642float(s64 num);

Convert 64-bit signed int to float.

float ul2f(s64 num);

float uint642float(u64 num);

Convert 64-bit unsigned int to float.

float fix2float(s32 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert signed fixed point to float.

float ufix2float(u32 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert unsigned fixed point to float.

float fix642float(s64 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert 64-bit signed fixed point to float.

float ufix642float(u64 num, int e);

e ... number of bits of fractional part (binary exponent)

Convert 64-bit unsigned fixed point to float.

Convert float to integer

s32 f2iz(float num);
s32 float2int_z(float num);

Convert float to signed int, rounding to zero (C-style int conversion).

s32 f2i(float num);
s32 float2int(float num);

Convert float to signed int, rounding down.

u32 f2ui(float num);
u32 float2uint(float num);

Convert float to unsigned int, rounding down.

s64 f2lz(float num);
s64 float2int64_z(float num);

Convert float to 64-bit signed int, rounding to zero (C-style int conversion).

s64 f2l(float num);
s64 float2int64(float num);

Convert float to 64-bit signed int, rounding down.

u64 f2ul(float num);
u64 float2uint64(float num);

Convert float to 64-bit unsigned int, rounding down.

s32 float2fix(float num, int e);

e ... number of bits of fractional part (binary exponent)

Convert float to signed fixed point, rounding down.

u32 float2ufix(float num, int e);

e ... number of bits of fractional part (binary exponent)

Convert float to unsigned fixed point, rounding down.

s64 float2fix64(float num, int e);

e ... number of bits of fractional part (binary exponent)

Convert float to 64-bit signed fixed point, rounding down.

u64 float2ufix64(float num, int e);

e ... number of bits of fractional part (binary exponent)

Convert float to 64-bit unsigned fixed point, rounding down.

Rounding

float truncf(float num);

Round number towards zero.

float roundf(float num);

Round number to nearest integer.

float floorf(float num);

Round number down to integer.

float ceilf(float num);

Round number up to integer.

Scientific functions

float sqrtf(float x);

Square root.

float deg2radf(float x);

Convert degrees to radians.

float rad2degf(float x);

Convert radians to degrees.

float sinf(float x);

Sine in radians.

float sinf_deg(float x);

Sine in degrees.

float cosf(float x);

Cosine in radians.

float cosf_deg(float x);

Cosine in degrees.

void sincosf(float x, float* psin, float* pcos);

Sine-cosine in radians.

void sincosf_deg(float x, float* psin, float* pcos);

Sine-cosine in degrees.

float tanf(float x);

Tangent in radians.

float tanf_deg(float x);

Tangent in degrees.

float cotanf(float x);

Cotangent in radians.

float cotanf_deg(float x);

Cotangent in degrees.

float asinf(float x);

Arc sine in radians.

float asinf_deg(float x);

Arc sine in degrees.

float acosf(float x);

Arc cosine in radians.

float acosf_deg(float x);

Arc cosine in degrees.

float atanf(float x);

Arc tangent in radians.

float atanf_deg(float x);

Arc tangent in degrees.

float acotanf(float x);

Arc cotangent in radians.

float acotanf_deg(float x);

Arc cotangent in degrees.

float atan2f(float y, float x);

Arc tangent of y/x in radians.

float atan2f_deg(float y, float x);

Arc tangent of y/x in degrees.

float sinhf(float x);

Hyperbolic sine.

float coshf(float x);

Hyperbolic cosine.

float tanhf(float x);

Hyperbolic tangent.

float asinhf(float x);

Inverse hyperbolic sine.

float acoshf(float x);

Inverse hyperbolic cosine.

float atanhf(float x);

Inverse hyperbolic tangent.

float expf(float x);

Natural exponent.

float logf(float x);

Natural logarithm.

float exp2f(float x);

Exponent with base 2.

float log2f(float x);

Logarithm with base 2.

float exp10f(float x);

Exponent with base 10.

float log10f(float x);

Logarithm with base 10.

float expm1f(float x);

$\exp(x) - 1$

float log1pf(float x);

logf(x + 1) This function can be very inaccurate

float fmaf(float x, float y, float z);

$x^*y + z$

float powintf(float x, int y);

Power by integer, x^y .

float powf(float x, float y);

Power x^y .

float hypotf(float x, float y);

Square root of sum of squares (hypotenuse), $\sqrt{x^*x + y^*y}$.

float cbrtf(float x);

Cube root, $\sqrt[3]{x}$, $x^{(1/3)}$.

float absf(float x);

Absolute value.

double f2d(float num);

double float2double(float num);

Convert float to double.

2.11. GPIO - General Purpose Input/Output Pins

Files: `sdk_gpio.h`, `sdk_gpio.c`

Config: -

RP2040/RP2350A has 30 General Purpose Input/Output pins, located at user bank for the programmer to use. All GPIOs support digital input and output, but GPIO26 to GPIO29 in Pio module can also be used as inputs to the chip's Analogue to Digital Converter (ADC). RP2350B has 48 GPIOs. These GPIOs are also accessible in the RP2350A, they are just not routed out to the pins.

Most functions have two forms. Functions can be addressed either by the GPIO pin number (parameter "int pin") or by a pointer to the hardware interface ("io32* hw" or "iobank0_status_ctrl_hw_t* hw"). Functions in the second case are marked with "_hw". Addressing by pin number is easier to use, addressing by pointer can generate slightly better optimized code.

Important change of the RP2350 compared to the RP2040: The "isolation latches" bit was added to the GPIO settings, which is active after reset and after initialization. This bit ensures that changes made to the pin settings (direction and output state) are first saved to the auxiliary latch, and only after the "isolation latches" bit is disabled are changes reflected to the pin. This prevents the occurrence of transients on the output of the pin. After setting the pin configuration, the "isolation latches" bit must be disabled. It is also automatically disabled when the pin function is set, so it is recommended to set the pin function last.

GPIO pin functions

<code>GPIO_FNC_XIP</code>	external Flash
<code>GPIO_FNC_SPI</code>	SPI
<code>GPIO_FNC_UART</code>	UART
<code>GPIO_FNC_I2C</code>	I2C
<code>GPIO_FNC_PWM</code>	PWM
<code>GPIO_FNC_SIO</code>	SIO (GPIO)
<code>GPIO_FNC_PIO0</code>	PIO 0
<code>GPIO_FNC_PIO1</code>	PIO 1
<code>GPIO_FNC_PIO2</code>	PIO 2 (only RP2350)
<code>GPIO_FNC_GPCK</code>	clock (only pins 20..25, or pins 12..15 on RP2350)
<code>GPIO_FNC_USB</code>	USB
<code>GPIO_FNC_HSTX</code>	HSTX (only RP2350)
<code>GPIO_FNC_TRACE</code>	coresight traceclk on pins 1..5 (only RP2350)
<code>GPIO_FNC_AUX</code>	UART AUX
<code>GPIO_FNC_NULL</code>	no function (use in case of ADC input)

See datasheet for table of GPIO functions on the pins.

GPIO pad control

pin = 0..31 or 0..49, including SWCLK and SWD, or **hw** = pad control register from GPIO_PadHw()

void GPIO_Voltage3V3();

GPIO set voltage to 3.3V (DVDD >= 2V5; default state).

void GPIO_Voltage1V8();

GPIO set voltage to 1.8V (DVDD <= 1V8).

io32* GPIO_PadHw(int pin);

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

Get pointer to pad control interface.

void GPIO_PadInit(int pin);

void GPIO_PadInit_hw(io32* hw);

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Init pad to default state: slow slew rate, enable schmitt, pull-down, 4mA, in/out enable. On RP2350, pad isolation is enabled.

void GPIO_IsolationEnable(int pin);

void GPIO_IsolationEnable_hw(io32* hw);

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Only RP2350. Enable isolation latches. If pad isolation is enabled, all output settings of the pad are latched. Disable pad isolation after pad is configured.

void GPIO_IsolationDisable(int pin);

void GPIO_IsolationDisable_hw(io32* hw);

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Only RP2350. Disable isolation latches. If pad isolation is disable, all output settings are transparent to the pad.

void GPIO_OutEnable(int pin);

void GPIO_OutEnable_hw(io32* hw);

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

GPIO output enable (default state). Use the GPIO_DirOut() function to set the pin direction to the output.

```
void GPIO_OutDisable(int pin);
void GPIO_OutDisable_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

GPIO output disable. Disable has priority over `GPIO_DirOut`.

```
void GPIO_InEnable(int pin);
void GPIO_InEnable_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Enable pin input (default state).

```
void GPIO_InDisable(int pin);
void GPIO_InDisable_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Disable pin input. Pin input should be disabled when using as ADC input.

```
void GPIO_Drive2mA(int pin);
void GPIO_Drive2mA_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Set output strength to 2 mA.

```
void GPIO_Drive4mA(int pin);
void GPIO_Drive4mA_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Set output strength to 4 mA (default state).

```
void GPIO_Drive8mA(int pin);
void GPIO_Drive8mA_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Set output strength to 8 mA.

```
void GPIO_Drive12mA(int pin);
void GPIO_Drive12mA_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from `GPIO_PadHw()`

Set output strength to 12 mA.

```
void GPIO_NoPull(int pin);
void GPIO_NoPull_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Set no pulls.

```
void GPIO_PullDown(int pin);
void GPIO_PullDown_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Set pull down (default state).

```
void GPIO_PullUp(int pin);
void GPIO_PullUp_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Set pull up. Default state for SWCLK and SWD pins.

```
void GPIO_BusKeep(int pin);
void GPIO_BusKeep_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Set bus keep (weak pull up and down).

```
void GPIO_SchmittEnable(int pin);
void GPIO_SchmittEnable_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Enable schmitt trigger (use hysteresis on input; default state).

```
void GPIO_SchmittDisable(int pin);
void GPIO_SchmittDisable_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)

hw ... pointer to pad control register from GPIO_PadHw()

Disable schmitt trigger (do not use hysteresis on input).

```
void GPIO_Slow(int pin);
void GPIO_Slow_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)
hw ... pointer to pad control register from GPIO_PadHw()

Use slow slew rate control on output (default state).

```
void GPIO_Fast(int pin);
void GPIO_Fast_hw(io32* hw);
```

pin ... pin number 0..31 or 0..49 (including SWCLK and SWD)
hw ... pointer to pad control register from GPIO_PadHw()

Use fast slew rate control on output.

GPIO pin control

pin = 0..29 or 0..47, or **hw** = pin control interface from GPIO_PinHw()

```
iobank0_status_ctrl_hw_t* GPIO_PinHw(int pin);
```

pin ... pin number 0..29 or 0..47

Get pointer to pin control interface.

```
u8 GPIO_OutPeri(int pin);
u8 GPIO_OutPeri_hw(const iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Only RP2040. Get output (0 or 1) from peripheral, before override.

```
u8 GPIO_OutPad(int pin);
u8 GPIO_OutPad_hw(const iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Get output (0 or 1) to pad, after override.

```
u8 GPIO_OePeri(int pin);
u8 GPIO_OePeri_hw(const iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Only RP2040. Get output enable (0 or 1) from peripheral, before override.

```
u8 GPIO_OePad(int pin);
u8 GPIO_OePad_hw(const iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Get output enable (0 or 1) to pad, after override.

u8 GPIO_InPad(int pin);

u8 GPIO_InPad_hw(const iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Get input (0 or 1) from pad, before override.

u8 GPIO_RawIn(int pin);

pin ... pin number 0..29 or 0..47

Get raw input (0 or 1) from pad, before override and before muxing, and enables pin input.

u8 GPIO_InPeri(int pin);

u8 GPIO_InPeri_hw(const iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Only RP2040. Get input (0 or 1) to peripheral, after override.

u8 GPIO_IrqPad(int pin);

u8 GPIO_IrqPad_hw(const iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Only RP2040. Get interrupt (0 or 1) from pad, before override.

u8 GPIO_IrqProc(int pin);

u8 GPIO_IrqProc_hw(const iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Get interrupt (0 or 1) to processor, after override.

void GPIO_OutOverNormal(int pin);

void GPIO_OutOverNormal_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output pin override to normal state (default state).

void GPIO_OutOverInvert(int pin);

void GPIO_OutOverInvert_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output pin override to invert state.

void GPIO_OutOverLow(int pin);

void GPIO_OutOverLow_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output pin override to low state.

void GPIO_OutOverHigh(int pin);

void GPIO_OutOverHigh_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output pin override to high state.

void GPIO_OEOverNormal(int pin);

void GPIO_OEOverNormal_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output enable pin override to normal state (default state).

void GPIO_OEOverInvert(int pin);

void GPIO_OEOverInvert_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output enable pin override to invert state.

void GPIO_OEOverDisable(int pin);

void GPIO_OEOverDisable_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output enable pin override to disable state.

void GPIO_OEOverEnable(int pin);

void GPIO_OEOverEnable_hw(iobank0_status_ctrl_hw_t* hw);

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Output enable pin override to enable state.

```
void GPIO_InOverNormal(int pin);
void GPIO_InOverNormal_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Input pin override to normal state (default state).

```
void GPIO_InOverInvert(int pin);
void GPIO_InOverInvert_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Input pin override to invert state.

```
void GPIO_InOverLow(int pin);
void GPIO_InOverLow_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Input pin override to low state.

```
void GPIO_InOverHigh(int pin);
void GPIO_InOverHigh_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set Input pin override to high state.

```
void GPIO IRQOverNormal(int pin);
void GPIO IRQOverNormal_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set IRQ pin override to normal state (default state).

```
void GPIO IRQOverInvert(int pin);
void GPIO IRQOverInvert_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set IRQ pin override to invert state.

```
void GPIO IRQOverLow(int pin);
void GPIO IRQOverLow_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set IRQ pin override to low state.

```
void GPIO_IRQOverHigh(int pin);
void GPIO_IRQOverHigh_hw(iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Set IRQ pin override to high state.

```
void GPIO_Fnc(int pin, int fnc);
```

pin ... pin number 0..29 or 0..47

fnc ... pin function **GPIO_FNC_***

Set GPIO function **GPIO_FNC_***, reset pas overrides to normal mode, enable output, enable input, disable isolation.

```
void GPIO_FncMask(u32 mask, int fnc); // RP2040
```

```
void GPIO_FncMask(u64 mask, int fnc); // RP2350
```

mask ... mask of pins (set bit to '1' to set function of the pin)

fnc ... pin function **GPIO_FNC_***

Set GPIO function **GPIO_FNC_*** with mask. To use pin mask in range (first..last), use function PinRangeMask().

```
void GPIO_FncRaw(int pin, int fnc);
```

pin ... pin number 0..29 or 0..47

fnc ... pin function **GPIO_FNC_***

Set GPIO function **GPIO_FNC_*** raw, resets pad overrides to normal mode, no other side effects (this is the difference from the GPIO_Fnc() function).

```
u8 GPIO_GetFnc(int pin);
```

```
u8 GPIO_GetFnc_hw(const iobank0_status_ctrl_hw_t* hw);
```

pin ... pin number 0..29 or 0..47

hw ... pointer to pin control interface from GPIO_PinHw()

Get current GPIO function **GPIO_FNC_***.

GPIO pin interrupt control

pin = 0..29 or 0..47

```
void GPIO_IRQAck(int pin, int events);
```

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to acknowledge

Acknowledge IRQ interrupts for both CPU.

void GPIO_IRQEnableCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (use CpuID() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to enable

Enable IRQ interrupt for selected CPU core.

void GPIO_IRQEnable(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to enable

Enable IRQ interrupt for current CPU core.

After enabling interrupt from pin, following steps are necessary:

- set handler of **IRQ_IO_BANK0** (use function SetHandler()), or use `isr_io_bank0` handler
- not necessary, but rather clear pending interrupt of **IRQ_IO_BANK0** (use function `NVIC_IRQClear()`)
- enable NVIC interrupt of **IRQ_IO_BANK0** (use function `NVIC_IRQEnable()`)
- enable global interrupt `ei()`

void GPIO_IRQDisableCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (use CpuID() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to enable

Disable IRQ interrupt for selected CPU core.

void GPIO_IRQDisable(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to enable

Disable IRQ interrupt for current CPU core.

void GPIO_IRQForceCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (use CpuID() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for selected CPU core. Forcing will trigger IRQ interrupt. Force request should be disabled at IRQ handler.

void GPIO_IRQForce(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for current CPU core. Forcing will trigger IRQ interrupt. Force request should be disabled at IRQ handler.

void GPIO_IRQUnforceCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (use Cpuid() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for selected CPU core.

void GPIO_IRQUnforce(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for current CPU core.

u8 GPIO_IRQIsPendingCpu(int cpu, int pin);

cpu ... CPU core 0 or 1 (use Cpuid() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

Check IRQ interrupt status for selected CPU core. Returns bit mask with **IRQ_EVENT_*** of pending events.

u8 GPIO_IRQIsPending(int pin);

pin ... pin number 0..29 or 0..47

Check IRQ interrupt status for current CPU core. Returns bit mask with **IRQ_EVENT_*** of pending events.

u8 GPIO_IRQIsForcedCpu(int cpu, int pin);

cpu ... CPU core 0 or 1 (use Cpuid() to get current core, or 2 = dormant wake)

pin ... pin number 0..29 or 0..47

Check if IRQ is forced for selected CPU core. Returns bit mask with **IRQ_EVENT_*** of forced events.

u8 GPIO_IRQIsForced(int pin);

pin ... pin number 0..29 or 0..47

Check if IRQ is forced for current CPU core. Returns bit mask with **IRQ_EVENT_*** of forced events.

void GPIO_IRQEnableDorm(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to enable

Enable IRQ interrupt for dormant wake.

void GPIO_IRQDisableDorm(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to disable

Disable IRQ interrupt for dormant wake.

void GPIO_IRQForceDorm(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for dormant wake.

void GPIO_IRQUnforceDorm(int pin, int events);

pin ... pin number 0..29 or 0..47

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for dormant wake.

u8 GPIO_IRQIsPendingDorm(int pin);

pin ... pin number 0..29 or 0..47

Check IRQ interrupt status for dormant wake. Returns bit mask with **IRQ_EVENT_*** of incoming events.

u8 GPIO_IRQIsForcedDorm(int pin);

pin ... pin number 0..29 or 0..47

Check if IRQ is forced for dormant wake. Returns bit mask with **IRQ_EVENT_*** of forced events.

void GPIO_IRQSetCallback(gpio_irq_callback_t cb);

cb ... pointer to callback function

Set generic callback for current core (NULL=disable callback). It will install default IRQ handler (as used with GPIO_SetHandler). To use:

- set callback using the **GPIO_IRQSetCallback()** function,
- enable interrupts on GPIO pins using the **GPIO_IRQEnable()** function,
- enable NVIC interrupt using the **NVIC_IRQEnable(IRQ_IO_BANK0)** command.

GPIO pin data

pin = 0..29 or 0..63 (including USB and QSPI)

u8 GPIO_In(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Get input pin (returns 0 or 1).

u32 GPIO_InAll(); // RP2040

u64 GPIO_InAll(); // RP2350

Get all input pins (bit 0..63 = pin 0..63).

void GPIO_Out0(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Output 0 (“LOW” state) to the pin.

void GPIO_Out1(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Output 1 (“HIGH” state) to the pin.

void GPIO_Flip(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Flip output to the pin.

void GPIO_Out(int pin, int val);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

val ... output value 0 or 1

Output value to the pin.

void GPIO_OutAll(u32 value); // RP2040

void GPIO_OutAll(u64 value); // RP2350

value ... output values to all pins (bit 0..63 = pin 0..63)

Output all pins.

void GPIO_OutMask(u32 mask, u32 value); // RP2040

void GPIO_OutMask(u64 mask, u64 value); // RP2350

mask ... mask of pins to output (set bit to ‘1’ to output value to the pin)

val ... output values of the pins

Output masked values to the pins. To use pin mask in range (first..last), use function PinRangeMask().

void GPIO_ClrMask(u32 mask); // RP2040

void GPIO_ClrMask(u64 mask); // RP2350

mask ... mask of pins to output 0 (set bit to ‘1’ to output value 0 to the pin)

Clear output pins masked. To use pin mask in range (first..last), use function PinRangeMask().

```
void GPIO_SetMask(u32 mask); // RP2040  
void GPIO_SetMask(u64 mask); // RP2350
```

mask ... mask of pins to output 1 (set bit to ‘1’ to output value 1 to the pin)

Set output pins masked. To use pin mask in range (first..last), use function PinRangeMask().

```
void GPIO_FlipMask(u32 mask); // RP2040  
void GPIO_FlipMask(u32 mask); // RP2350
```

mask ... mask of pins to flip output (set bit to ‘1’ to flip output of the pin)

Flip output pins masked. To use pin mask in range (first..last), use function PinRangeMask().

```
u8 GPIO_GetOut(int pin);
```

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Get last output value to pin (returns 0 or 1).

```
u32 GPIO_GetOutAll(); // RP2040  
u64 GPIO_GetOutAll(); // RP2350
```

Get all last output values to pins.

```
void GPIO_DirOut(int pin);
```

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Set output direction of pin (enable output mode).

```
void GPIO_DirIn(int pin);
```

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Set input direction of pin (disable output mode).

```
void GPIO_DirOutMask(u32 mask); // RP2040  
void GPIO_DirOutMask(u64 mask); // RP2350
```

mask ... mask of pins to set output direction (set bit to ‘1’ to set output direction)

Set output direction of pin masked. To use pin mask in range (first..last), use function PinRangeMask().

```
void GPIO_DirInMask(u32 mask); // RP2040  
void GPIO_DirInMask(u64 mask); // RP2350
```

mask ... mask of pins to set input direction (set bit to ‘1’ to set input direction)

Set input direction of pin masked. To use pin mask in range (first..last), use function PinRangeMask().

```
void GPIO_SetDirMask(u32 mask, u32 outval); // RP2040  
void GPIO_SetDirMask(u64 mask, u64 outval); // RP2350
```

mask ... mask of pins to output (set bit to ‘1’ to set direction of the pin)

outval ... output directions of the pins ('1' = output)

Set direction masked.

void GPIO_SetDirAll(u32 outval); // RP2040

void GPIO_SetDirAll(u64 outval); // RP2350

outval ... output directions of the pins ('1' = output)

Set direction of all pins.

u8 GPIO_GetDir(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Get output direction of pin (returns 0=input or 1=output).

u32 GPIO_GetDirAll(); // RP2040

u64 GPIO_GetDirAll(); // RP2350

Get output direction of all pins (returns 0=input, 1=output).

void GPIO_Init(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Initialize GPIO pin base function, set input mode, LOW output value.

void GPIO_InitMask(u32 mask); // RP2040

void GPIO_InitMask(u64 mask); // RP2350

mask ... mask of pins to initialize GPIO pin base function

Initialize GPIO pin base function masked (bit '1' = initialize this pin). To use pin mask in range (first..last), use function PinRangeMask().

void GPIO_Reset(int pin);

pin ... pin number 0..29 or 0..63 (including USB and QSPI)

Reset GPIO pin (return to reset state).

void GPIO_ResetMask(u32 mask); // RP2040

void GPIO_ResetMask(u64 mask); // RP2350

mask ... mask of pins to return to reset state

Reset GPIO pins masked (return to reset state). To use pin mask in range (first..last), use function PinRangeMask().

2.12. HSTX - High-speed Serial Transmit

Files: `sdk_hstx.h, sdk_hstx.c`

Config: `USE_HSTX (default 1) (only RP2350)`

HSTX module enables high speed serial output to pins GPIO12 to GPIO19. An example of application is outputting signal to VGA or HDMI monitor.

`void HSTX_InitGPIO(int pin);`

pin ... 12..19; means bit 0..7

Configure one pin to use HSTX.

`void HSTX_InitGPIOMask(u32 mask);`

mask ... bit mask, bits 12..19 means pins 12..19

Configure pins GPIO12..19 to use HSTX. To use pin mask in range (first..last), use function RangeMask.

`void HSTX_InitGPIOAll();`

Configure all pins GPIO12..19 to use HSTX.

`void HSTX_Enable();`

Enable HSTX (start sending data).

`void HSTX_Disable();`

Disable HSTX (stop sending data).

`void HSTX_CmdEnable();`

Enable command expander - commands will be inserted to data. HSTX must be disabled to change this flag.

`void HSTX_CmdDisable();`

Disable command expander - FIFO data passed directly out. HSTX must be disabled to change this flag.

`void HSTX_PioEnable();`

Enable PIO connection (coupled mode):

- SEL_P and SEL_N indices 24 to 31 will select bits from 8-bit PIO path
- SEL_P and SEL_N indices 0 to 23 will index shift register as normal
- PIO outputs are those same outputs as direct connect PIO to pins
- HSTX must be clocked directly from sys_clk

void HSTX_PioDisable();

Disable PIO connection. SEL_P and SEL_N indices will index shift register as normal.

void HSTX_PioSel(int pio);

pio ... 0..2

Select PIO for PIO coupled mode.

void HSTX_Shift(int shift);

shift ... 0..31

Set shift - rotate shift register right after each cycle.

void HSTX_ShiftLeft(int shift);

shift ... 0..31

Set shift left - rotate shift register left after each cycle.

void HSTX_NShifts(int num);

num ... 1..32

Set number of shifts before refilling shift register from FIFO.

void HSTX_Phase(int phase);

phase ... 0..15

Set initial phase of the generated clock in number of half periods of clock cycle.

void HSTX_Clk(int clk);

clk ... 1..16

Set clock period of the generated clock in number of clock cycles.

void HSTX_SelP(int bit, int sel_p);

bit ... output bit 0..7 (= GPIO 12..19)

sel_p ... select data bit from shift register 0..31

Select data bit 0..31 for the first half of clock cycle for output bit 0..7.

void HSTX_SelN(int bit, int sel_n);

bit ... output bit 0..7 (= GPIO 12..19)

sel_n ... select data bit from shift register 0..31

Select data bit 0..31 for the second half of clock cycle for output bit 0..7.

void HSTX_Inv(int bit);

bit ... output bit 0..7 (= GPIO 12..19)

Set invert mode of output bit 0..7.

void HSTX_NonInv(int bit);

bit ... output bit 0..7 (= GPIO 12..19)

Set non-invert mode of output bit 0..7 (default state).

void HSTX_OutClk(int bit);

bit ... output bit 0..7 (= GPIO 12..19)

Select output of bit 0..7 from clock generator. Sel_p and sel_n are ignored, inv can be used.

void HSTX_OutShift(int bit);

bit ... output bit 0..7 (= GPIO 12..19)

Select output of bit 0..7 from shift register (default state).

void HSTX_RawShift(int shift);

shift .. 0..31

Set number of bits to right-rotate shift register by each time data is pushed to the output shifter, when command is raw data.

void HSTX_RawNShifts(int shifts);

shifts ... 1..32

Set number of times of shifts before refilling from FIFO, when command is raw data.

void HSTX_EncShift(int shift);

shift .. 0..31

Set number of bits to right-rotate shift register by each time data is pushed to the output shifter, when command is encoded data.

void HSTX_EncNShifts(int shifts);

shifts ... 1..32

Set number of times of shifts before refilling from FIFO, when command is encoded data.

void HSTX_L0Rot(int rot);

rot ... 0..31

Set right-rotate TMDS shifter data of lane 0.

void HSTX_L0Bits(int bits);

bits ... 1..8

Set number of data bits of TMDS lane 0.

void HSTX_L1Rot(int rot);

rot ... 0..31

Set right-rotate TMDS shifter data of lane 1.

void HSTX_L1Bits(int bits);

bits ... 1..8

Set number of data bits of TMDS lane 1.

void HSTX_L2Rot(int rot);

rot ... 0..31

Set right-rotate TMDS shifter data of lane 2.

void HSTX_L2Bits(int bits);

bits ... 1..8

Set number of data bits of TMDS lane 2.

Bool HSTX_FifolsOver();

Check if HSTX FIFO is overflow (write when full).

void HSTX_FifoClrOver();

Clear HSTX FIFO overflow flag.

Bool HSTX_FifolsEmpty();

Check if HSTX FIFO is empty.

Bool HSTX_FifolsFull();

Check if HSTX FIFO is full.

int HSTX_FifoLevel();

Get current level of HSTX FIFO (0..7).

void HSTX_FifoWrite(u32 data);

data ... data to send

Write data to HSTX FIFO.

2.13. I2C - Inter-Integrated Circuit

Files: `sdk_i2c.h`, `sdk_i2c.c`

Config: USE_I2C (default 1)

I2C is a commonly used 2-wire interface that can be used to connect devices for low speed data transfer using clock SCL and data SDA wires. RP2040/RP2350 has two identical instances of an I2C controller. Default slave address is 0x055 (`I2C_SLAVE_ADDR`).

Most functions have two forms. Functions can be addressed either by the I2C index (parameter "int i2c") or by a pointer to the hardware interface ("`i2c_hw_t*` hw", as returned from the `I2C_GetHw()` function). Functions in the second case are marked with "`_hw`". Addressing by i2c index is easier to use, addressing by pointer can generate slightly better optimized code.

Default baudrates

`I2C_BAUDRATE_SS` standard mode 100 kb/s

`I2C_BAUDRATE_FS` fast mode 400 kb/s

`I2C_BAUDRATE_FP` fast mode plus 1000 kb/s

`i2c_hw_t* I2C_GetHw(int i2c);`

i2c ... I2C index 0 or 1

Get I2C hardware interface from I2C index. Returns `i2c0_hw` or `i2c1_hw` (`i2c0` or `i2c1`).

`u8 I2C_GetIdx(const i2c_hw_t* hw);`

hw ... pointer to hardware interface (as returned by `I2C_GetHw()`)

Get I2C index from I2C hardware interface.

`void I2C_Reset(int i2c);`

i2c ... I2C index 0 or 1

Reset I2C periphery.

`void I2C_Init(int i2c, u32 baudrate);`

i2c ... I2C index 0 or 1

baudrate ... required baudrate (recommended `I2C_BAUDRATE_*` constants)

Initialize I2C to master mode.

`void I2C_Term(int i2c);`

i2c ... I2C index 0 or 1

Terminate I2C periphery (disables I2C, wait to complete and reset I2C).

```
void I2C_SlaveInit(int i2c, u8 addr, i2c_slave_handler_t handler);
```

i2c ... I2C index 0 or 1

addr ... slave address (must be in interval from 0x08 to 0x77)

handler ... slave message handler

I2C initialize to slave mode, with event handler (should called after I2C_Init()). First run I2C_Init() and then I2C_SlaveInit(). To terminate, run I2C_SlaveTerm() and then I2C_Term().

```
void I2C_SlaveTerm(int i2c);
```

i2c ... I2C index 0 or 1

Terminate slave mode and return to master mode.

```
void I2C_SlaveMode(int i2c, u8 addr);
```

i2c ... I2C index 0 or 1

addr ... slave address (must be in interval from 0x08 to 0x77)

Switch I2C to slave mode.

```
void I2C_MasterMode(int i2c);
```

i2c ... I2C index 0 or 1

Switch I2C to master mode.

```
Bool I2C_IsEnabled(int i2c);
```

```
Bool I2C_IsEnabled_hw(const i2c_hw_t* hw);
```

i2c ... I2C index 0 or 1

hw ... pointer to hardware interface (as returned by I2C_GetHw())

Check if I2C is enabled.

```
void I2C_Enable(int i2c);
```

```
void I2C_Enable_hw(i2c_hw_t* hw);
```

i2c ... I2C index 0 or 1

hw ... pointer to hardware interface (as returned by I2C_GetHw())

Enable I2C (wait to complete).

```
void I2C_Disable(int i2c);
```

```
void I2C_Disable_hw(i2c_hw_t* hw);
```

i2c ... I2C index 0 or 1

hw ... pointer to hardware interface (as returned by I2C_GetHw())

Disable I2C (wait to complete).

```

Bool I2C_IsActive(int i2c);
Bool I2C_IsActive_hw(const i2c_hw_t* hw);
    i2c ... I2C index 0 or 1
    hw ... pointer to hardware interface (as returned by I2C_GetHw())
Check if I2C transfer is active.

void I2C_Baudrate(int i2c, u32 baudrate);
    i2c ... I2C index 0 or 1
    baudrate ... new baudrate (recommended I2C_BAUDRATE_* constants)
Set I2C baudrate and enable I2C. Required only in master mode.

u32 I2C_GetBaudrate(int i2c);
    i2c ... I2C index 0 or 1
Get current I2C baudrate.

u8 I2C_TxFifo(int i2c);
u8 I2C_TxFifo_hw(const i2c_hw_t* hw);
    i2c ... I2C index 0 or 1
    hw ... pointer to hardware interface (as returned by I2C_GetHw())
Get remaining space in transmit FIFO (0 = transmit FIFO is full).

u8 I2C_RxFifo(int i2c);
u8 I2C_RxFifo_hw(const i2c_hw_t* hw);
    i2c ... I2C index 0 or 1
    hw ... pointer to hardware interface (as returned by I2C_GetHw())
Get number of entries in receive FIFO (0 = receive FIFO is empty).

void I2C_SendData(int i2c, const u8* src, int len);
void I2C_SendData_hw(i2c_hw_t* hw, const u8* src, int len);
    i2c ... I2C index 0 or 1
    hw ... pointer to hardware interface (as returned by I2C_GetHw())
    src ... source data buffer
    len ... length of data to send
Send data from buffer.

void I2C_RecvData(int i2c, u8* dst, int len);
void I2C_RecvData_hw(i2c_hw_t* hw, u8* dst, int len);
    i2c ... I2C index 0 or 1
    hw ... pointer to hardware interface (as returned by I2C_GetHw())
    src ... destination data buffer

```

len ... required length of data to receive

Receive data to buffer.

u8 I2C_GetDreq(int i2c, Bool tx);

i2c ... I2C index 0 or 1

tx ... True for sending data to I2C, False for receiving data from I2C

Get **DREQ** to use for pacing transfers.

int I2C_SendMsg(int i2c, u8 addr, const u8* src, int len, Bool nostop, u32 timeout);

i2c ... I2C index 0 or 1

addr ... slave address of device to write to, must be in interval from 0x08 to 0x77

src ... pointer to data to send

len .. length of data in bytest to send (must be > 0)

nostop ... no stop after end of transfer (next transfer begin with Restart, not Start)

timeout ... timeout in [us] of one character sent (0=no timeout)

Send message to slave. Returns number of bytes sent, can be less than 'len' in case of error.

int I2C_RecvMsg(int i2c, u8 addr, u8* dst, int len, Bool nostop, u32 timeout);

i2c ... I2C index 0 or 1

addr ... slave address of device to read from, must be in interval from 0x08 to 0x77

dst ... pointer to buffer to receive data

len .. length of data in bytest to receive (must be > 0)

nostop ... no stop after end of transfer (next transfer begin with Restart, not Start)

timeout ... timeout in [us] of one character received (0=no timeout)

Read message from slave. Returns number of bytes received, can be less than 'len' in case of error.

2.14. Interpolator

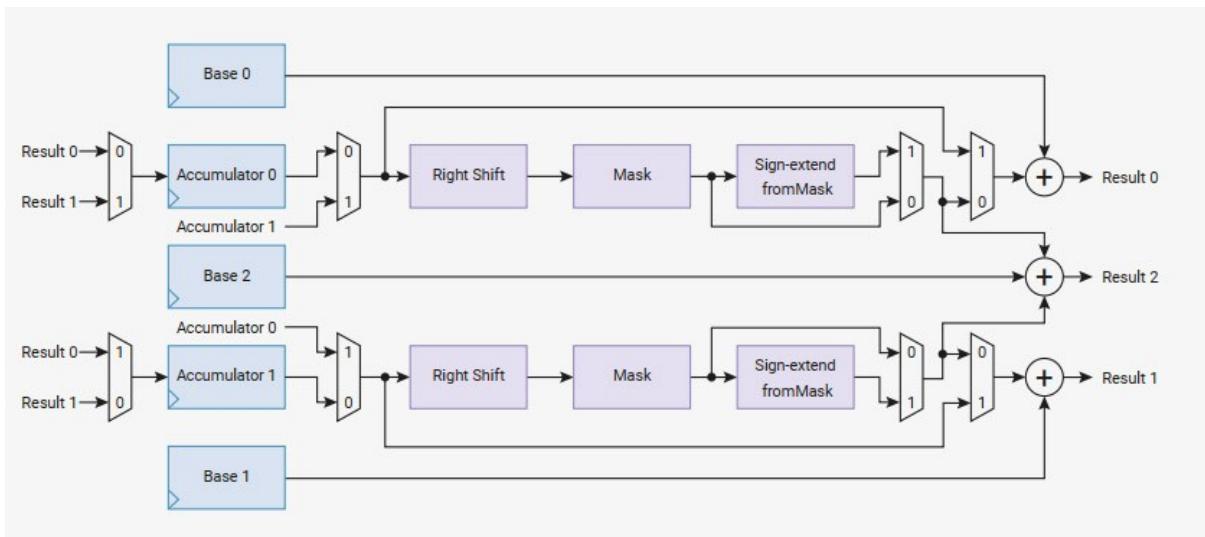
Files: `sdk_interp.h`, `sdk_interp.c`

Config: `USE_INTERP` (default 1)

Each CPU core is equipped with two interpolators which can accelerate tasks by combining certain preconfigured operations into a single processor cycle. The interpolators are used to accelerate audio operations within the SDK, but their flexible configuration makes it possible to optimise many other tasks such as quantization and dithering, table lookup address generation, affine texture mapping, decompression and linear feedback.

It is recommended to schedule the use of each interpolator lane in the program configuration using `#define`. In addition, it is possible to use automatic allocation of interpolator lanes using the "claim" functions. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

Most functions have two forms. Functions can be addressed either by the interpolator index (parameter "int interp") or by a pointer to the hardware interface ("interp_hw_t* hw", as returned from `InterpGetHw()` function). Functions in the second case are marked with "`_hw`". Addressing by interpolator index is easier to use, addressing by pointer can generate slightly better optimized code.



Direct Access to Interpolator Registers (volatile `u32*`)

<code>INTERP_ACCUM(interp, lane)</code>	R/W accumulator (lane = 0 or 1)
<code>INTERP_BASE(interp, lane)</code>	R/W base (lane = 0 or 1, or 2 to common base)
<code>INTERP_POP(interp, lane)</code>	RO read lane result and write to accumulators (lane = 0 or 1, or 2 for full result)
<code>INTERP_PEEK(interp, lane)</code>	RO read lane result without altering state (lane = 0 or 1, or 2 for full result)
<code>INTERP_CTRL(interp, lane)</code>	control register (lane = 0 or 1)
<code>INTERP_ADD(interp, lane)</code>	R/W add to accumulator (lane = 0 or 1)

INTERP_BASE01(interp) WO write LOW 16 bits to BASE0, HIGH 16 bits to BASE1

interp = index of interpolator 0 or 1, **lane** = index of lane 0 or 1, or 2 to common

interp_hw_t* InterpGetHw(int interp);

interp ... interpolator 0 or 1

Get hardware interface from interpolator index.

u8 InterpGetInx(const interp_hw_t* hw);

hw ... pointer to hardware interface (as returned from InterpGetHw())

Get interpolator index from hardware interface.

Claim interpolator lane

Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not. Functions are not atomic safe (not recommended to be used in both cores or in IRQ at the same time).

void InterpClaim(int interp, int lane);

interp ... interpolator 0 or 1

lane ... lane 0 or 1

Claim interpolator lane (mark it as used).

void InterpClaimMask(int interp, int mask);

interp ... interpolator 0 or 1

mask ... mask of lanes (bit 0 and bit 1)

Claim interpolator lanes with mask (mark them as used).

void InterpUnclaim(int interp, int lane);

interp ... interpolator 0 or 1

lane ... lane 0 or 1

Unclaim interpolator lane (mark it as not used).

void InterpUnclaimMask(int interp, int mask);

interp ... interpolator 0 or 1

mask ... mask of lanes (bit 0 and bit 1)

Unclaim interpolator lanes with mask (mark them as not used).

Bool InterpIsClaimed(int interp, int lane);

interp ... interpolator 0 or 1

lane ... lane 0 or 1

Check if interpolator lane is claimed.

Interpolator configuration structure (u32 control word)

After setup, write configuration word using InterpCfg() function.

u32 InterpCfgDef();

Get default configuration word.

void InterpCfg(int interp, int lane, u32 cfg);
void InterpCfg_hw(interp_hw_t* hw, int lane, u32 cfg);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

cfg ... configuration word

Config write configuration word.

void InterpCfgShift(u32* cfg, int shift);

cfg ... pointer to configuration word

shift ... shift value 0 to 31

Config setup interpolator shift (right-shift applied to accumulator before masking).

void InterpCfgMask(u32* cfg, int lsb, int msb);

cfg ... pointer to configuration word

lsb ... least significant bit allowed to pass, inclusive, 0 to 31

msb ... most significant bit allowed to pass, inclusive, 0 to 31 (**lsb** <= **msb**)

Config setup interpolator mask range (range of bits that are allowed to pass).

void InterpCfgCrossInput(u32* cfg, Bool en);

cfg ... pointer to configuration word

en ... True to feed opposite lane's accumulator to this input,

False to use input from own accumulator

Config enable cross input (feed opposite lane's accumulator into this lane's shift+mask input).

void InterpCfgCrossResult(u32* cfg, Bool en);

cfg ... pointer to configuration word

en ... True to feed opposite lane's result to this accumulator,

False to no auto-feed from result

Config enable cross results (feed opposite lane's result into this lane's accumulator on POP).

void InterpCfgSigned(u32* cfg);

cfg ... pointer to configuration word

Config set interpolator signed mode (values are sign-extended to 32 bits).

void InterpCfgUnsigned(u32* cfg);

cfg ... pointer to configuration word

Config set interpolator unsigned mode (default state).

void InterpCfgAddRaw(u32* cfg, Bool en);

cfg ... pointer to configuration word

en ... True to shift+mask is bypassed, False to shift+mask is applied

Config set raw add option (mask+shift is bypassed for this result, add raw input value - does not affect FULL result).

void InterpCfgBlend(u32* cfg, Bool en);

cfg ... pointer to configuration word

en ... True = LANE1 result is linear interpolation between BASE0 and BASE1, controlled by 8 LSB bits of LANE1 shift+mask value (fractional number 0 to 255 / 256), LANE0 result does not have BASE0 added (yields only 8 LSB of LANE1 shift+mask), FULL result does not have lane 1 shift+mask value added (BASE2 + lane0 shift+mask),

Config set blending mode (only on interpolator 0 of each CPU core). LANE1 SIGNED flag controls whether interpolation is signed or unsigned.

void InterpCfgClamp(u32* cfg, Bool en);

cfg ... pointer to configuration word

en ... True = LANE0 result is shifted and masked ACCUM0, clamped by lower bound of BASE0 and upper bound of BASE1.

Config set clamping mode (only on interpolator 1 of each CPU core). LANE0 SIGNED flag controls whether comparisons are signed or unsigned.

void InterpCfgMSB(u32* cfg, u8 bits);

cfg ... pointer to configuration word

bits ... two bits to be ORed into bits 29:28, value 0 to 3

0 ... base address 0x00000000

1 ... base address 0x10000000, flash memory

2 ... base address 0x20000000, SRAM memory

3 ... base address 0x30000000

Config set forced bits (two data bits ORed into bits 29:28 of lane result presented to the processor) (no effect on internal paths). Handy for using lane to generate sequence of pointers into flash or SRAM.

Interpolator setup

Save interpolator state structure:

```
typedef struct {  
    u32    accum[2];    // accumulators  
    u32    base[3];     // bases  
    u32    ctrl[2]; // controls  
} interp_hw_save_t;
```

void InterpSave(int interp, interp_hw_save_t* save);

interp ... interpolator 0 or 1

save ... pointer to destination variable of type `interp_hw_save_t`

Save interpolator state to the variable (for current CPU core).

void InterpLoad(int interp, const interp_hw_save_t* save);

interp ... interpolator 0 or 1

save ... pointer to source variable of type `interp_hw_save_t`

Load interpolator state from the variable (for current CPU core).

void InterpReset(int interp);

interp ... interpolator 0 or 1

Reset interpolator to default state.

void InterpShift(int interp, int lane, int shift);

void InterpShift_hw(interp_hw_t* hw, int lane, int shift);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from `InterpGetHw()`)

lane ... lane 0 or 1

shift ... shift value 0 to 31

Set interpolator shift. Right-shift is applied to accumulator before masking.

void InterpMask(int interp, int lane, int lsb, int msb);

void InterpMask_hw(interp_hw_t* hw, int lane, int lsb, int msb);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from `InterpGetHw()`)

lane ... lane 0 or 1

lsb ... least significant bit allowed to pass, inclusive, 0 to 31

msb ... most significant bit allowed to pass, inclusive, 0 to 31 (lsb <= msb)

Set interpolator mask range (range of bits that are allowed to pass).

void InterpSigned(int interp, int lane);
void InterpSigned_hw(interp_hw_t* hw, int lane);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

Set interpolator to signed mode (values are sign-extended to 32 bits).

void InterpUnsigned(int interp, int lane);
void InterpUnsigned_hw(interp_hw_t* hw, int lane);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

Set interpolator to unsigned mode (default state).

void InterpCrossInput(int interp, int lane, Bool en);
void InterpCrossInput_hw(interp_hw_t* hw, int lane, Bool en);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

en ... True = feed opposite lane's accumulator to this input,

False = use input from own accumulator

Set cross switch to feed opposite lane's accumulator into this lane's shift+mask input.

void InterpCrossResult(int interp, int lane, Bool en);
void InterpCrossResult_hw(interp_hw_t* hw, int lane, Bool en);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

en ... True = feed opposite lane's result to this accumulator,

False = no auto-feed from result

Set cross switch to feed opposite lane's result into this lane's accumulator on POP.

void InterpAddRaw(int interp, int lane, Bool en);
void InterpAddRaw_hw(interp_hw_t* hw, int lane, Bool en);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

en ... True = shift+mask is bypassed, False = shift+mask is applied

Set if mask+shift is bypassed for this result, add raw input value (does not affect FULL result).

```
void InterpMSB(int interp, int lane, int bits);
void InterpMSB_hw(interp_hw_t* hw, int lane, int bits);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

bits ... two bits to be ORed into bits 29:28, value 0 to 3

0 ... base address 0x00000000

1 ... base address 0x10000000, flash memory

2 ... base address 0x20000000, SRAM memory

3 ... base address 0x30000000

Set two data bits that are ORed into bits 29:28 of lane result presented to the processor (no effect on internal paths). This is handy for using lane to generate sequence of pointers into flash or SRAM memory.

```
void InterpBlend(Bool en);
```

en ... enable blending mode

Enable blending mode of interpolator 0 (cannot be used on interpolator 1). If blending mode is enabled, LANE1 result is linear interpolation between BASE0 and BASE1, controlled by 8 LSB bits of LANE1 shift+mask value (fractional number 0 to 255 / 256). LANE0 result does not have BASE0 added (yields only 8 LSB of LANE1 shift+mask). FULL result does not have lane 1 shift+mask value added (BASE2 + lane0 shift+mask). LANE1 SIGNED flag controls whether interpolation is signed or unsigned.

```
void InterpClamp(Bool en);
```

en ... enable clamping mode

Enable clamping mode of interpolator 1 (cannot be used on interpolator 0). If clamping is enabled, LANE0 result is shifted and masked ACCUM0, clamped by lower bound of BASE0 and upper bound of BASE1. LANE0 SIGNED flag controls whether comparisons are signed or unsigned.

```
Bool InterpOver(int interp, int accum);
Bool InterpOver_hw(interp_hw_t* hw, int accum);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

accum ... accumulator 0 or 1, 2 = either 0 or 1 is set

Check if any masked-off MSBs bits in ACCUM are set (overflow).

Interpolator data processing

```
void InterpAccum(int interp, int lane, u32 val);
void InterpAccum_hw(interp_hw_t* hw, int lane, u32 val);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

val ... value to set

Set interpolator accumulator.

```
u32 InterpGetAccum(int interp, int lane);
u32 InterpGetAccum_hw(const interp_hw_t* hw, int lane);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

Get interpolator accumulator.

```
void InterpAdd(int interp, int lane, u32 val);
void InterpAdd_hw(interp_hw_t* hw, int lane, u32 val);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

val ... value to add

Add atomically to interpolator accumulator.

```
u32 InterpRaw(int interp, int lane);
u32 InterpRaw_hw(const interp_hw_t* hw, int lane);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1

Get interpolator raw lane value (result of shift + mask operation).

```
void InterpBase(int interp, int lane, u32 val);
void InterpBase_hw(interp_hw_t* hw, int lane, u32 val);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... base 0, 1, or 2 to common base of both lanes

val ... value to set

Set interpolator base.

```
u32 InterpGetBase(int interp, int lane);  
u32 InterpGetBase_hw(const interp_hw_t* hw, int lane);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... base 0, 1, or 2 to common base of both lanes

Get interpolator base.

```
void InterpBaseBoth(int interp, u32 val);  
void InterpBaseBoth_hw(interp_hw_t* hw, u32 val);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

val ... value to set, lower 16 bits go to BASE0, upper 16 bits go to BASE1

Set interpolator both bases. Values can be signed or unsigned.

```
void InterpBaseBoth16(int interp, u16 val0, u16 val1);  
void InterpBaseBoth16_hw(interp_hw_t* hw, u16 val0, u16 val1);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

val0 ... value to set to BASE0

val1 ... value to set to BASE1

Set interpolator both bases. Values can be signed or unsigned.

```
u32 InterpPop(int interp, int lane);  
u32 InterpPop_hw(interp_hw_t* hw, int lane);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1, or 2 for full result

Get lane result and write it back to both accumulators.

```
u32 InterpPopFull(int interp);  
u32 InterpPopFull_hw(interp_hw_t* hw);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

Get lane full result and write to both accumulators.

```
u32 InterpPeek(int interp, int lane);  
u32 InterpPeek_hw(interp_hw_t* hw, int lane);
```

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

lane ... lane 0 or 1, or 2 for full result

Get lane result without altering any internal state.

u32 InterpPeekFull(int interp);

u32 InterpPeekFull_hw(interp_hw_t* hw);

interp ... interpolator 0 or 1

hw ... pointer to hardware interface (as returned from InterpGetHw())

Get lane full result without altering any internal state.

2.15. IRQ - Interrupt Request

Files: `sdk_irq.h`, `sdk_irq.c`

Config: `USE_IRQ (default 1)`

Each CPU-ARM core has its own interrupt controller NVIC (Nested Vector Interrupt Control), that can trigger an interrupt handler in response to an external interrupt signal. Software can set the priority of each interrupt. The interrupt priority can be set in 4 steps. A value of 0 is the highest priority, a value of 3 is the lowest priority.

The RP2350 in RISC-V mode uses a different interrupt controller, but the SDK adapts the functions to minimize the differences. In most cases, the descriptions also apply to RISC-V unless otherwise noted.

Pointers to interrupt handlers are located in vector table. Pointer to vector table is set to the vector table offset register VTOR of the NVIC controller. When the program is started, the vector table is first stored in flash memory at the beginning of the program. During runtime initialization, the vector table is copied from flash memory to RAM memory and a pointer to the table is redirected. To install your interrupt handler, you can use the default interrupt handler name. In that case, the linker will store a pointer to your interrupt handler in the flash vector table, instead of the default interrupt handler. Alternatively, you can set the pointer to the interrupt handler in the vector table in RAM while the application is running, using the `SetHandler()` function.

Vector table in PicoLibSDK is shared between both CPU cores on start of application.

Interrupt hardware priority

<code>IRQ_PRIO_REALTIME</code>	highest priority (value 0), real-time time-critical interrupts
<code>IRQ_PRIO_SYSTICK</code>	high priority (value 1), SysTick
<code>IRQ_PRIO_NORMAL</code>	normal priority (value 2), default state of interrupts
<code>IRQ_PRIO_PENDSV</code>	lowest priority (value 3), PendSV handler

Interrupt request indices and default handlers of RP2040

ARM Cortex-M0+ core interrupt request indices - Exceptions

<code>IRQ_RESET</code>	(-15) default handler <code>_reset_handler</code>
<code>IRQ_NMI</code>	(-14) default handler <code>isr_nmi</code>
<code>IRQ_HARDFAULT</code>	(-13) default handler <code>isr_hardfault</code>
<code>IRQ_SVCALL</code>	(-5) default handler <code>isr_svcall</code>
<code>IRQ_PENDSV</code>	(-2) default handler <code>isr_pendsv</code>
<code>IRQ_SYSTICK</code>	(-1) default handler <code>isr_systick</code>

RP2040 device interrupt request indices - Interrupts

<code>IRQ_TIMER_0</code>	(0) default handler <code>isr_timer_0</code>
<code>IRQ_TIMER_1</code>	(1) default handler <code>isr_timer_1</code>

IRQ_TIMER_2	(2) default handler isr_timer_2
IRQ_TIMER_3	(3) default handler isr_timer_3
IRQ_PWM_WRAP	(4) default handler isr_pwm_wrap
IRQ_USBCTRL	(5) default handler isr_usbctrl
IRQ_XIP	(6) default handler isr_xip
IRQ_PIO0_0	(7) default handler isr_pio0_0
IRQ_PIO0_1	(8) default handler isr_pio0_1
IRQ_PIO1_0	(9) default handler isr_pio1_0
IRQ_PIO1_1	(10) default handler isr_pio1_1
IRQ_DMA_0	(11) default handler isr_dma_0
IRQ_DMA_1	(12) default handler isr_dma_1
IRQ_IO_BANK0	(13) default handler isr_io_bank0
IRQ_IO_QSPI	(14) default handler isr_io_qspi
IRQ_SIO_PROC0	(15) default handler isr_sio_proc0
IRQ_SIO_PROC1	(16) default handler isr_sio_proc1
IRQ_CLOCKS	(17) default handler isr_clocks
IRQ_SPI0	(18) default handler isr_spi0
IRQ_SPI1	(19) default handler isr_spi1
IRQ_UART0	(20) default handler isr_uart0
IRQ_UART1	(21) default handler isr_uart1
IRQ_ADC_FIFO	(22) default handler isr_adc_fifo
IRQ_I2C0	(23) default handler isr_i2c0
IRQ_I2C1	(24) default handler isr_i2c1
IRQ_RTC	(25) default handler isr_rtc
IRQ_SPAREIRQ_0	(26) default handler isr_spare_0
IRQ_SPAREIRQ_1	(27) default handler isr_spare_1
IRQ_SPAREIRQ_2	(28) default handler isr_spare_2
IRQ_SPAREIRQ_3	(29) default handler isr_spare_3
IRQ_SPAREIRQ_4	(30) default handler isr_spare_4
IRQ_SPAREIRQ_5	(31) default handler isr_spare_5

Interrupt request indices and default handlers of RP2350

ARM Cortex-M33 core interrupt request indices - Exceptions

IRQ_RESET	(-15) default handler _reset_handler
IRQ_NMI	(-14) default handler isr_nmi
IRQ_HARDFAULT	(-13) default handler isr_hardfault
IRQ_MEMFAULT	(-12) default handler isr_memfault

IRQ_BUSFAULT	(-11) default handler isr_busfault
IRQ_USAGEFAULT	(-10) default handler isr_usagefault
IRQ_SECUREFAULT	(-9) default handler isr_securefault
IRQ_SVCALL	(-5) default handler isr_svcall
IRQ_DEBMONITOR	(-4) default handler isr_debmonitor
IRQ_PENDSV	(-2) default handler isr_pendsv
IRQ_SYSTICK	(-1) default handler isr_systick

RISC-V core interrupt request indices - Exceptions

IRQ_MCALL	(-14) exception 11, default handler isr_mcall
IRQ_SCALL	(-12) exception 9, default handler isr_scall
IRQ_UCALL	(-11) exception 8, default handler isr_ucall
IRQ_STOREFAULT	(-10) exception 7, default handler isr_storefault
IRQ_STOREALIGN	(-9) exception 6, default handler isr_storealign
IRQ_LOADFAULT	(-8) exception 5, default handler isr_loadfault
IRQ_LOADALIGN	(-7) exception 4, default handler isr_loadalign
IRQ_BREAKPOINT	(-6) exception 3, default handler isr_breakpoint
IRQ_INSTRILEGAL	(-5) exception 2, default handler isr_instrilegal
IRQ_INSTRFAULT	(-4) exception 1, default handler isr_instrfault
IRQ_INSTRALIGN	(-3) exception 0, default handler isr_instralign
IRQ_MTIMER	(-2) trap 7, default handler isr_mtimer
IRQ_SOFTIRQ	(-1) trap 3, default handler isr_softirq

RP2350 device interrupt request indices - Interrupts

IRQ_TIMER0_0	(0) default handler isr_timer0_0
IRQ_TIMER0_1	(1) default handler isr_timer0_1
IRQ_TIMER0_2	(2) default handler isr_timer0_2
IRQ_TIMER0_3	(3) default handler isr_timer0_3
IRQ_TIMER1_0	(4) default handler isr_timer1_0
IRQ_TIMER1_1	(5) default handler isr_timer1_1
IRQ_TIMER1_2	(6) default handler isr_timer1_2
IRQ_TIMER1_3	(7) default handler isr_timer1_3
IRQ_PWM_WRAP_0	(8) default handler isr_pwm_wrap_0
IRQ_PWM_WRAP_1	(9) default handler isr_pwm_wrap_1
IRQ_DMA_0	(10) default handler isr_dma_0
IRQ_DMA_1	(11) default handler isr_dma_1
IRQ_DMA_2	(12) default handler isr_dma_2
IRQ_DMA_3	(13) default handler isr_dma_3
IRQ_USBCTRL	(14) default handler isr_usbctrl

IRQ_PIO0_0	(15) default handler isr_pio0_0
IRQ_PIO0_1	(16) default handler isr_pio0_1
IRQ_PIO1_0	(17) default handler isr_pio1_0
IRQ_PIO1_1	(18) default handler isr_pio1_1
IRQ_PIO2_0	(19) default handler isr_pio2_0
IRQ_PIO2_1	(20) default handler isr_pio2_1
IRQ_IO_BANK0	(21) default handler isr_io_bank0
IRQ_IO_BANK0_NS	(22) default handler isr_io_bank0_ns
IRQ_IO_QSPI	(23) default handler isr_io_qspi
IRQ_IO_QSPI_NS	(24) default handler isr_io_qspi_ns
IRQ_SIO_FIFO	(25) default handler isr_sio_fifo
IRQ_SIO_BELL	(26) default handler isr_sio_bell
IRQ_SIO_FIFO_NS	(27) default handler isr_sio_fifo_ns
IRQ_SIO_BELL_NS	(28) default handler isr_sio_bell_ns
IRQ_SIO_MTIMECMP	(29) default handler isr_sio_mtimecmp
IRQ_CLOCKS	(30) default handler isr_clocks
IRQ_SPI0	(31) default handler isr_spi0
IRQ_SPI1	(32) default handler isr_spi1
IRQ_UART0	(33) default handler isr_uart0
IRQ_UART1	(34) default handler isr_uart1
IRQ_ADC_FIFO	(35) default handler isr_adc_fifo
IRQ_I2C0	(36) default handler isr_i2c0
IRQ_I2C1	(37) default handler isr_i2c1
IRQ OTP	(38) default handler isr_otp
IRQ_TRNG	(39) default handler isr_trng
IRQ_PROC0_CTI	(40) default handler isr_proc0_cti
IRQ_PROC1_CTI	(41) default handler isr_proc1_cti
IRQ_PLL_SYS	(42) default handler isr_pll_sys
IRQ_PLL_USB	(43) default handler isr_pll_usb
IRQ_POWMAN_POW	(44) default handler isr_powman_pow
IRQ_POWMAN_TIMER	(45) default handler isr_powman_timer
IRQ_SPAREIRQ_0	(46) default handler isr_spare_0
IRQ_SPAREIRQ_1	(47) default handler isr_spare_1
IRQ_SPAREIRQ_2	(48) default handler isr_spare_2
IRQ_SPAREIRQ_3	(49) default handler isr_spare_3
IRQ_SPAREIRQ_4	(50) default handler isr_spare_4
IRQ_SPAREIRQ_5	(51) default handler isr_spare_5

Common interrupt handler definition

```
typedef void (*irq_handler_t());
```

irq_handler_t* GetVTOR();

Get address of interrupt vector table of this CPU core.

void SetVTOR(irq_handler_t* addr);

addr ... pointer to new interrupt vector table

Set address of interrupt vector table of this CPU core.

void* SetThumbBit(void* addr);

Set thumb bit of the address.

void* ClrThumbBit(void* addr);

Clear thumb bit of the address.

void SetHandler(s8 irq, irq_handler_t handler);

irq ... interrupt request indice **IRQ_*** (-16..25 or 51, including exceptions)

handler ... interrupt handler

Set interrupt handler. To setup handler, vector table must be located in RAM. This is already provided during program startup by the internal function RuntimelInit(), the vector table is located in RAM at the address RamVectorTable. If vector table is not in RAM, use handlers with default names **isr_***. It is also possible to set a handler to handle the exception.

irq_handler_t GetHandler(int irq);

irq ... interrupt request indice **IRQ_*** (-16..25 or 51, including exceptions)

Get current interrupt servie handler.

void NVIC_IRQClearMask(u32 mask); // RP2040

void NVIC_IRQClearMask(u64 mask); // RP2350

mask ... bit mask of interrupts to clear (only interrupts 0..25 or 51, not exceptions)

Clear pending interrupts of NVIC masked of this CPU core. Exception handlers cannot be used. To use IRQ mask in range (first..last), use function **IrqRangeMask**.

void NVIC_IRQClear(int irq);

irq ... interrupt request indice **IRQ_*** (0..25 or 51, not exceptions)

Clear pending interrupt of NVIC of this CPU core (force interrupts). Exception handlers cannot be used.

```
void NVIC_IRQForceMask(u32 mask); // RP2040  
void NVIC_IRQForceMask(u64 mask); // RP2350
```

mask ... bit mask of interrupts to raise (only interrupts 0..25 or 51, not exceptions)

Set pending interrupts of NVIC masked of this CPU core. Exception handlers cannot be used. To use IRQ mask in range (first..last), use function `IrqRangeMask`.

```
void NVIC_IRQForce(int irq);
```

irq ... interrupt request indice **IRQ_*** (0..25 or 51, not exceptions)

Set pending interrupt of NVIC of this CPU core (force interrupt). Exception handlers cannot be used.

```
Bool NVIC IRQIsPending(int irq)
```

irq ... interrupt request indice **IRQ_*** (0..25 or 51, not exceptions)

Check if interrupt of NVIC of this CPU core is pending (`irq = 0..25 or 0..51`). Exception handlers cannot be used.

```
void NVIC IRQEnableMask(u32 mask); // RP2040  
void NVIC IRQEnableMask(u64 mask); // RP2350
```

mask ... bit mask of interrupts to enable (only interrupts 0..25 or 51, not exceptions)

Enable interrupts of NVIC masked of this CPU core. Exception handlers cannot be enabled or disabled. To use IRQ mask in range (first..last), use function `IrqRangeMask`.

```
void NVIC IRQEnable(int irq);
```

irq ... interrupt request indice **IRQ_*** (only interrupts 0..25 or 51, not exceptions)

Enable interrupt **IRQ_*** of NVIC of this CPU core. Exception handlers cannot be enabled or disabled.

```
void NVIC IRQDisableMask(u32 mask); // RP2040  
void NVIC IRQDisableMask(u32 mask); // RP2350
```

mask ... bit mask of interrupts to disable (only interrupts 0..25 or 51, not exceptions)

Disable interrupts of NVIC masked of this CPU core. Exception handlers cannot be enabled or disabled. To use IRQ mask in range (first..last), use function `IrqRangeMask`.

```
void NVIC IRQDisable(int irq);
```

irq ... interrupt request indice **IRQ_*** (only interrupts 0..25 or 51, not exceptions)

Disable interrupt **IRQ_*** of NVIC of this CPU core. Exception handlers cannot be enabled or disabled.

```
Bool NVIC IRQIsEnabled(int irq);
```

irq ... interrupt request indice **IRQ_*** (only interrupts 0..25 or 51, not exceptions)

Check if interrupt **IRQ_*** of NVIC of this CPU core is enabled. Exception handlers cannot be checked.

void NVIC_IRQHandler(int irq, u8 prio);

irq ... interrupt request indice **IRQ_*** (only interrupts 0..25 or 51, not exceptions)

prio ... priority **IRQ_PRIO_***

Set interrupt priority of NVIC of this CPU core.

void NVIC_IRQHandlerDef();

Set all interrupt priorities of NVIC of this CPU core to default value. This function is called at program start from the internal RuntimelInit() function. It sets all interrupts to the default priority **IRQ_PRIO_NORMAL**. It sets the SysTick handler to the **IRQ_PRIO_SYSTICK** priority and sets the PendSV handler to the **IRQ_PRIO_PENDSV** priority.

void NVIC_SVCallPrio(int prio);

prio ... priority **IRQ_PRIO_***

Set interrupt priority of SVCall exception of this CPU core (default **IRQ_PRIO_NORMAL**). Cannot be used in RISC-V mode.

void NVIC_PendSVPrio(int prio);

prio ... priority **IRQ_PRIO_***

Set interrupt priority of PendSV exception of this CPU core (default **IRQ_PRIO_PENDSV**). Cannot be used in RISC-V mode.

void NVIC_PendSVForce();

Trigger PendSV exception of this CPU core (reschedule multitask system). Cannot be used in RISC-V mode.

Bool NVIC_PendSVIsPending();

Check if PendSV exception of this CPU core is pending. Cannot be used in RISC-V mode.

void NVIC_PendSVClear();

Clear PendSV pending state of this CPU core. Cannot be used in RISC-V mode.

void NVIC_SysTickPrio(int prio);

prio ... priority **IRQ_PRIO_***

Set interrupt priority of SysTick exception of this CPU core (default **IRQ_PRIO_SYSTICK**). Cannot be used in RISC-V mode.

void NVIC_SysTickForce();

Trigger SysTick exception of this CPU core (force SysTick). Cannot be used in RISC-V mode.

Bool NVIC_SysTickIsPending();

Check if SysTick exception of this CPU core is pending. Cannot be used in RISC-V mode.

void NVIC_SystickClear();

Clear SysTick pending state of this CPU core. Cannot be used in RISC-V mode.

void NVIC_NMIForce();

Trigger NMI exception of this CPU core (force NMI). Cannot be used in RISC-V mode.

2.16. Multicore

Files: `sdk_multicore.h`, `sdk_multicore.c`

Config: `USE_MULTICORE` (default 1)

The RP2040/RP2350 processor contains 2 CPU cores, core0 and core1. Each core uses a different stack. core0 implicitly uses 4KB stack0 in the SCRATCH_X segment, while core1 implicitly uses 4KB stack1 in the SCRATCH_Y segment. After reset, only core0 runs. core1 "sleeps" and waits for wakeup via mailbox (FIFO).

`typedef void (*pCore1Fnc());`

Prototype of core 1 exec function. After execution, function can exit and that stops core 1.

`volatile Bool Core1IsRunning;`

Flag that can be used to test if the executed core 1 function is still running.

`void Core1Reset();`

Reset CPU core 1. This function can be used to force termination of the core 1 function.

`void Core1Exec(pCore1Fnc entry);`

entry ... function to be executed on core 1

Start core 1 function. This function must be called from core 0. Flag `Core1IsRunning` can be used to check if the core 1 is still running. When the function stops running and exits, core 1 stops. Or core 0 can terminate function on core 1 with the function `Core1Reset`. Both cores use the same VTOR interrupt table (handlers are shared) but have separate NVIC interrupt controllers.

Inter-core lockout

If it is necessary to temporarily suspend the program in the second core, the following procedure can be used. This must be done when programming Flash memory - a program from core 0 or core 1 must not run from Flash memory during programming.

To lockout core 1 when writing to flash from core 0:

- initialize lockout handler from core 0 with `LockoutInit(1)`
- start lockout of core 1 from core 0 with `LockoutStart()`
- core 1 raises `LockoutHandler()` and waits
- write to flash memory from core 0
- stop lockout of core 1 from core 0 with `LockoutStop()`
- core 1 returns from lockout handler and continues in execution
- deinitialize lockout handler from core 0 with `LockoutTerm(1)`

void LockoutInit(int core);

core ... core 0 or 1

Initialize lockout handler for core 0 or 1.

void LockoutTerm(int core);

core ... core 0 or 1

Deinitialize lockout handler for core 0 or 1 (disables IRQ handler).

void LockoutStart();

Start lockout other core.

void LockoutStop();

Stop lockout other core.

Doorbells

Doorbells are used to send signal to other core. Doorbells are valid only on RP2350.

void DoorbellSetOut(int doorbell);

doorbell ... 0..7

Activate doorbell on other core.

void DoorbellClrOut(int doorbell);

doorbell ... 0..7

Deactivate doorbell on other core.

Bool DoorbellIsOut(int doorbell);

doorbell ... 0..7

Check doorbell on other core.

void DoorbellSetIn(int doorbell);

doorbell ... 0..7

Activate doorbell on this core.

void DoorbellClrIn(int doorbell);

doorbell ... 0..7

Deactivate doorbell on this core.

Bool DoorbellIsIn(int doorbell);

doorbell ... 0..7

Check doorbell on this core.

2.17. PIO - Programmable Input/Output Block

Files: `sdk_pio.h`, `sdk_pio.c`

Config: `USE_PIO` (default 1)

The PIO Programmable Input/Output block is one of the most powerful tools of the RP2040/RP2350 processor. The PIO is essentially another processor in the processor, allowing fast and accurate signal handling. Its original focus is to handle communication protocols that are not included in the base processor, including, for example, the USB protocol. But it can also be used for other purposes, e.g. to output a VGA or HDMI video signal.

The processor RP2040 contains 2 identical PIO blocks, PIO0 and PIO1. RP2350 contains one more PIO block, PIO2. Each PIO block is equipped with a program memory of 32 instructions of 16 bits each and 4 state machine blocks, SM0 to SM3. Each state machine can independently execute sequential programs to manipulate GPIOs and transfer data.

It is recommended to schedule the use of each PIO and SM in the program configuration using `#define`. In addition, it is possible to use automatic allocation of state machines using the "claim" functions. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

The map for using program instructions works similarly. The programmer can use the usage map to find empty memory space. However, this is not a requirement; functions are not dependent on detecting space. It is preferable to plan the memory layout in advance.

Most functions have two forms. Functions can be addressed either by the PIO index (parameter "int pio") or by a pointer to the hardware interface ("`pio_hw_t* hw`", as returned from `PioGetHw()` function). Functions in the second case are marked with "`_hw`". Addressing by PIO index is easier to use, addressing by pointer can generate slightly better optimized code.

Each state machine is equipped with:

- two 32-bit shift registers (IN and OUT direction),
- two 32-bit scratch registers (X and Y),
- 4-level 32-bit bus FIFO in both direction (TX and RX), reconfigurable as 8-level 32-bit in a single direction,
- fractional clock divider,
- flexible GPIO mapping,
- DMA interface,
- IRQ flag set/clear/status.

The PIO program can be written in symbolic notation, which you can compile using the `pioasm.exe` compiler located in the `_tools` folder. Refer to the RP2040 datasheet for details on PIO programs and instructions. You can also use the following macros to write program instructions.

To use PIO:

- initialize PIO and all SMs with `PioInit()`

- load program with PioLoadProg()
- set start address with PioSetAddr() if not 0
- setup GPIO pins to use PIO with PioSetupGPIO()
- setup state machine parameters with PioSet*()
- set pin state and direction with PioSetPin() and PioSetPinDir()
- setup DMA if needed
- setup IRQ if needed (PiolrqEnable())
- start SM with PioSMEnableMask()
- transfer data with PioWriteWait() and PioReadWait()

pio_hw_t* PioGetHw(int pio);

pio ... PIO number 0..2

Get hardware interface from PIO index.

u8 PioGetIdx(const pio_hw_t* hw);

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

Get PIO index from hardware interface.

Claim state machine

Claim functions are not atomic safe - not recommended to be used in both cores or in IRQ at the same time.

void PioClaim(int pio, int sm);

pio ... PIO number 0..2

sm ... state machine 0 to 3

Claim PIO state machine (mark it as used).

void PioClaimMask(int pio, int mask);

pio ... PIO number 0..2

mask ... mask of state machines (bit 0 to bit 3)

Claim PIO state machines with mask (mark them as used).

void PioUnclaim(int pio, int sm);

pio ... PIO number 0..2

sm ... state machine 0 to 3

Unclaim PIO state machine (mark it as not used).

void PioUnclaimMask(int pio, int mask);

pio ... PIO number 0..2

mask ... mask of state machines (bit 0 to bit 3)

Unclaim PIO state machines with mask (mark them as not used).

Bool PiolsClaimed(int pio, int sm);

pio ... PIO number 0..2

sm ... state machine 0 to 3

Check if PIO state machine is claimed.

s8 PioClaimFree(int pio);

pio ... PIO number 0..2

Claim free unused PIO state machine (returns -1 on error).

Program used map

void PioClearUsedMap(int pio);

pio ... PIO number 0..2

Clear map of used instruction memory.

void PioSetUsedIns(int pio, int off);

pio ... PIO number 0..2

off ... offset of instruction in memory 0..31

Mark one instruction in memory as used.

void PioUnsetUsedIns(int pio, int off);

pio ... PIO number 0..2

off ... offset of instruction in memory 0..31

Mark one instruction in memory as not used.

Bool PiolsUsedIns(int pio, int off);

pio ... PIO number 0..2

off ... offset of instruction in memory 0..31

Check if entry in instruction memory is used.

Bool PiolsFreeProg(int pio, int off, int num);

pio ... PIO number 0..2

off ... offset of start space in memory 0..31

num ... length of space in memory 0..32

Check free space for the program.

int PioFindFreeProg(int pio, int num);

pio ... PIO number 0..2

num ... length of space in memory 0..32

Find free space for the program. Returns offset or -1 on error.

void PioSetUsedProg(int pio, int off, int num);

pio ... PIO number 0..2

off ... offset of start program in memory 0..31

num ... length of program in memory 0..32

Mark program as used.

void PioUnsetUsedProg(int pio, int off, int num);

pio ... PIO number 0..2

off ... offset of start program in memory 0..31

num ... length of program in memory 0..32

Mark program as not used.

State machine configuration structure (pio_sm_config)

After setup, use PioSetCfg() to apply configuration.

pio_sm_config PioCfgDef();

Get default configuration structure of state machine.

void PioCfg(int pio, int sm, const pio_sm_config* cfg);

void PioCfg_hw(pio_hw_t* hw, int sm, const pio_sm_config* cfg);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

cfg ... pointer to state machine configuration structure

Apply state machine configuration.

void PioCfgClkdiv(pio_sm_config* cfg, u32 clkdiv);

cfg ... pointer to state machine configuration structure

clkdiv ... clock_divider * 256 (default 0x100, means clock_divider=1.00,

1 instruction per 1 system clock)

Config set state machine clock divider (frequency = sys_clk / clock_divider).

void PioCfgClkdivFloat(pio_sm_config* cfg, float clkdiv);

cfg ... pointer to state machine configuration structure

clkdiv ... clock_divider (default 1.00, 1 instruction per 1 system clock)

Config set state machine clock divider as float (frequency = sys_clk / clock_divider).

void PioCfgSideset(pio_sm_config* cfg, int base, int count, Bool optional, Bool pindirs);

cfg ... pointer to state machine configuration structure

base ... base GPIO pin asserted by sideset output, 0 to 31 (default 0)

count ... number of bits for side set (inclusive enable bit if present), 0 to 5, def. 0

optional ... topmost side set bit is used as enable flag for whether apply side set
on that instruction (default False)

pindirs ... side set affects pin directions rather than values (default False)

Config setup sideset.

void PioCfgJmpPin(pio_sm_config* cfg, int pin);

cfg ... pointer to state machine configuration structure

pin ... GPIO pin to use as condition for JMP PIN (0 to 29, default 0)

PIO set GPIO pin for JMP PIN instruction.

**void PioCfgOutSpecial(pio_sm_config* cfg, Bool sticky, Bool has_enable_pin,
int enable_pin_index)**

cfg ... pointer to state machine configuration structure

sticky ... enable 'sticky' output (continuously re-asserting most recent
OUT/SET values to the pins; default False)

has_enable_pin ... use auxiliary OUT enable pin (default False; If True,
use a bit of OUT data as an auxiliary write enable. When used
in conjunction with OUT_STICKY, writes with an enable of False
will deassert the latest pin write. This can create useful
masking/override behaviour due to the priority ordering of
state machine pin writes SM0 < SM1 < ...)

enable_pin_index ... Which data bit to use for auxiliary OUT enable (0..31; def. 0)

Config set special OUT.

void PioCfgWrap(pio_sm_config* cfg, int wrap_target, int wrap_top);

cfg ... pointer to state machine configuration structure

wrap_target ... wrap destination, after reaching wrap_top execution
is wrapped to this address (0 to 31, default 0)

wrap_top ... wrap source, after reaching this address execution is

then wrapped to wrap_target (0 to 31, default 31)

Config set wrap address. If wrap_top instruction is jump with true condition, the jump takes priority.

void PioCfgMovStatus(pio_sm_config* cfg, Bool status_rx, int level);

cfg ... pointer to state machine configuration structure

status_rx ... True compare RX FIFO level, False compare TX FIFO level (def.)

level ... comparison level N (default 0)

PIO set "MOV x,STATUS". Instruction "MOV x,STATUS" fills all-ones if FIFO level < N, otherwise all-zeroes.

void PioCfgFifoJoin(pio_sm_config* cfg, int join);

cfg ... pointer to state machine configuration structure

join ... set FIFO joining PIO_FIFO_JOIN_* (default PIO_FIFO_JOIN_RXTX)

Config set FIFO joining.

void PioCfgOutShift(pio_sm_config* cfg, Bool shift_right, Bool autopull, int pull_threshold);

cfg ... pointer to state machine configuration structure

shift_right ... shift OSR right (default True)

autopull ... autopull enable (pull when OSR is empty, default False)

pull_threshold ... threshold in bits to shift out before auto/conditional

re-pulling OSR (1 to 32, default 32)

Config set OUT shifting.

void PioCfgInShift(pio_sm_config* cfg, Bool shift_right, Bool autopush, int push_threshold);

cfg ... pointer to state machine configuration structure

shift_right ... shift ISR right (data enters from left, default True)

autopush ... autopush enable (push when ISR is filled, default False)

push_threshold ... threshold in bits to shift in before auto/conditional

re-pushing ISR (1 to 32, default 32)

Config set IN shifting.

void PioCfgSet(pio_sm_config* cfg, int base, int count);

cfg ... pointer to state machine configuration structure

base ... base GPIO pin asserted by SET instruction, 0 to 31 (default 0)

count ... number of GPIO pins asserted by SET instruction, 0 to 5 (default 5)

Config setup SET pins.

```
void PioCfgOut(pio_sm_config* cfg, int base, int count);
```

cfg ... pointer to state machine configuration structure

base ... base GPIO pin asserted by OUT instruction, 0 to 31 (default 0)

count ... number of GPIO pins asserted by OUT instruction, 1 to 32 (default 32)

Config setup OUT pins.

```
void PioCfgIn(pio_sm_config* cfg, int base);
```

cfg ... pointer to state machine configuration structure

base ... base GPIO pin asserted by IN instruction, 0 to 31 (default 0)

Config setup IN pins.

Setup state machine using hardware registers

```
void PioSMDefault(int pio, int sm);
```

```
void PioSMDefault_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Set PIO default setup of state machine. Function is called from PioSMInit().

```
void PioSetClkdiv(int pio, int sm, u32 clkdiv);
```

```
void PioSetClkdiv_hw(pio_hw_t* hw, int sm, u32 clkdiv);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

clkdiv ... clock_divider * 256

Set PIO state machine clock divider (frequency = sys_clk / clock_divider). Default clock divider is 0x100, which means clock_divider=1.00, that is executing 1 instruction per 1 system clock.

```
void PioSetClkdivFloat (int pio, int sm, float clkdiv);
```

```
void PioSetClkdivFloat_hw(pio_hw_t* hw, int sm, float clkdiv);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

clkdiv ... clock_divider

Set PIO state machine clock divider as float (frequency = sys_clk / clock_divider). Default clock divider is 1.00, that is executing 1 instruction per 1 system clock.

```
void PioSetupSideset(int pio, int sm, int base, int count, Bool optional, Bool pindirs);
void PioSetupSideset_hw(pio_hw_t* hw, int sm, int base, int count, Bool optional, Bool pindirs);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

base ... base GPIO pin asserted by sideset output, 0 to 31 (default 0)

count ... number of bits used for side set (inclusive enable bit if present), 0 to 5

optional ... topmost side set bit is used as enable flag for whether apply side set
on that instruction (default False)

pindirs ... side set affects pin directions rather than values (default False)

Setup PIO sideset.

```
void PioSetJmpPin(int pio, int sm, int pin);
void PioSetJmpPin_hw(pio_hw_t* hw, int sm, int pin);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

pin ... GPIO pin to use as condition for JMP PIN (0 to 29, default 0)

PIO set GPIO pin for JMP PIN instruction.

```
void PioSetOutSpecial(int pio, int sm, Bool sticky, Bool has_enable_pin, int enable_pin_index);
```

```
void PioSetOutSpecial_hw(pio_hw_t* hw, int sm, Bool sticky, Bool has_enable_pin, int enable_pin_index);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

sticky ... enable 'sticky' output (default False)

has_enable_pin ... use auxiliary OUT enable pin (default False)

enable_pin_index ... which data bit to use for auxiliary OUT enable (0..31; default 0)

Set PIO special OUT. Sticky output continuously re-assert most recent OUT/SET to the pins. If "has_enable_pin" is True, use a bit of OUT data as an auxiliary write enable. When used in conjunction with 'sticky', writes with an enable of 0 will deassert the latest pin write. This can create useful masking/override behaviour due to the priority ordering of state machine pin writes (SM0 < SM1 < ...).

```
void PioSetWrap(int pio, int sm, int wrap_target, int wrap_top);
void PioSetWrap_hw(pio_hw_t* hw, int sm, int wrap_target, int wrap_top);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())
sm ... state machine 0 to 3
wrap_target ... wrap destination (0 to 31, default 0)
wrap_top ... wrap source (0 to 31, default 31)

Set PIO wrap address. After reaching wrap_top address, execution is then wrapped to wrap_target address. If wrap_top instruction is jump with true condition, the jump takes priority.

void PioSetMovStatus(int pio, int sm, Bool status_rx, int level);
void PioSetMovStatus_hw(pio_hw_t* hw, int sm, Bool status_rx, int level);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
sm ... state machine 0 to 3
status_rx ... True to compare RX FIFO level, False to compare TX FIFO level
level ... comparison level N (default 0)

Set PIO "MOV x,STATUS". Instruction "MOV x,STATUS" fills all-ones if FIFO level < N, otherwise all-zeroes.

void PioSetFifoJoin(int pio, int sm, int join);
void PioSetFifoJoin_hw(pio_hw_t* hw, int sm, int join);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
sm ... state machine 0 to 3
join ... set FIFO joining **PIO_FIFO_JOIN_*** (default **PIO_FIFO_JOIN_RXTX**)

Set PIO FIFO joining. PIO FIFO joining type:

PIO_FIFO_JOIN_RXTX	use separate RX FIFO and TX FIFO, every 4-entries deep
PIO_FIFO_JOIN_TX	use only TX FIFO, 8-entries deep
PIO_FIFO_JOIN_RX	use only RX FIFO, 8-entries deep

```
void PioSetOutShift(int pio, int sm, Bool shift_right, Bool autopull, int
pull_threshold);
void PioSetOutShift_hw(pio_hw_t* hw, int sm, Bool shift_right, Bool autopull,
int pull_threshold);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

shift_right ... shift OSR right (default True)

autopull ... autopull enable (pull when OSR is empty, default False)

pull_threshold ... threshold in bits to shift out before auto/conditional
re-pulling OSR (1 to 32, default 32)

Set PIO OUT shifting.

```
void PioSetInShift(int pio, int sm, Bool shift_right, Bool autopush, int
push_threshold);
void PioSetInShift_hw(pio_hw_t* hw, int sm, Bool shift_right, Bool autopush,
int push_threshold);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

shift_right ... shift ISR right (data enters from left, default True)

autopush ... autopush enable (push when ISR is filled, default False)

push_threshold ... threshold in bits to shift in before auto/conditional
re-pushin ISR (1 to 32, default 32)

Set PIO IN shifting.

```
void PioSetupSet(int pio, int sm, int base, int count);
void PioSetupSet_hw(pio_hw_t* hw, int sm, int base, int count);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

base ... base GPIO pin asserted by SET instruction, 0 to 31 (default 0)

count ... number of GPIO pins asserted by SET instruction, 0 to 5 (default 5)

Setup PIO SET pins.

```
void PioSetupOut(int pio, int sm, int base, int count);
void PioSetupOut_hw(pio_hw_t* hw, int sm, int base, int count);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

base ... base GPIO pin asserted by OUT instruction, 0 to 31 (default 0)

count ... number of GPIO pins asserted by OUT instruction, 1 to 32 (default 32)

Setup PIO OUT pins.

void PioSetupIn(int pio, int sm, int base);

void PioSetupIn_hw(pio_hw_t* hw, int sm, int base);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

base ... base GPIO pin asserted by IN instruction, 0 to 31 (default 0)

Setup PIO IN pins.

Setup PIO

u8 PioGetDreq(int pio, int sm, Bool tx);

pio ... PIO number 0..2

sm ... state machine 0 to 3

tx ... True for sending data from CPU to state machine,

False for receiving data from state machine to CPU

Get DREQ to use for pacing transfers to state machine.

void PioSetupGPIO(int pio, int pin, int count);

pio ... PIO number 0..2

pin ... pin base 0 to 29

count ... number of pins 0 to 29

Setup GPIO pins to connect to the PIO.

void PioSMEnable(int pio, int sm);

void PioSMEnable_hw(pio_hw_t* hw, int sm);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Enable PIO state machine (run program).

void PioSMDDisable(int pio, int sm);

void PioSMDDisable_hw(pio_hw_t* hw, int sm);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Disable PIO state machine (stop program).

```
void PioSMEnableMask(int pio, int sm_mask);
void PioSMEnableMask_hw(pio_hw_t* hw, int sm_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm_mask ... state machines mask of bits B0 to B3

Enable multiple PIO state machines. To use mask in range (first..last), use function RangeMask().

```
void PioSMEnableMaskSync(int pio, int sm_mask);
void PioSMEnableMaskSync_hw(pio_hw_t* hw, int sm_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm_mask ... state machines mask of bits B0 to B3

Enable multiple PIO state machines synchronized (resets their clock dividers). To use mask in range (first..last), use function RangeMask().

```
void PioSMDisableMask(int pio, int sm_mask);
void PioSMDisableMask_hw(pio_hw_t* hw, int sm_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm_mask ... state machines mask of bits B0 to B3

Disable multiple PIO state machines. To use mask in range (first..last), use function RangeMask().

```
void PioSMRestart(int pio, int sm);
void PioSMRestart_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Restart PIO state machine. This method clears ISR, shift counters, clock divider counter, pin write flags, delay counter, latched EXEC instruction, IRQ wait condition.

```
void PioSMRestartMask(int pio, int sm_mask);
void PioSMRestartMask_hw(pio_hw_t* hw, int sm_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm_mask ... state machines mask of bits B0 to B3

Restart multiple PIO state machines. This method clears ISR, shift counters, clock divider counter, pin write flags, delay counter, latched EXEC instruction, IRQ wait condition

```
void PioClkdivRestart(int pio, int sm);
void PioClkdivRestart_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Restart clock divider of PIO state machine (resets fractional counter).

```
void PioClkdivRestartMask(int pio, int sm_mask);
void PioClkdivRestartMask_hw(pio_hw_t* hw, int sm_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm_mask ... state machines mask of bits B0 to B3

Restart clock divider of multiple PIO state machines (resets fractional counter).

PIO program control

```
u8 PioGetPC(int pio, int sm);
u8 PioGetPC_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Get current program counter of PIO state machine (returns 0..31).

```
void PioExec(int pio, int sm, u16 instr);
```

```
void PioExec_hw(pio_hw_t* hw, int sm, u16 instr);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

instr ... instruction (can use **PIO_ENCODE_***)

PIO execute one instruction and then resume execution of main program.

```
Bool PiolsExec(int pio, int sm);
Bool PiolsExec_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Check if instruction set by PioExec() is still running.

```
void PioExecWait(int pio, int sm, u16 instr);
void PioExecWait_hw(pio_hw_t* hw, int sm, u16 instr);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

instr ... instruction

PIO execute one instruction, wait to complete and then resume execution of main program.
State machine must be enabled with valid clock divider.

```
void PioWrite(int pio, int sm, u32 data);
void PioWrite_hw(pio_hw_t* hw, int sm, u32 data);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

data ... data value

Write data to TX FIFO of PIO state machine. If TX FIFO is full, the most recent value will be overwritten.

```
u32 PioRead(int pio, int sm);
u32 PioRead_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Read data from RX FIFO of PIO state machine. If RX FIFO is empty, the return value is zero.

```
Bool PioRxIsFull(int pio, int sm);
Bool PioRxIsFull_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Check if RX FIFO of PIO state machine is full.

```
Bool PioRxIsEmpty(int pio, int sm);
Bool PioRxIsEmpty_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Check if RX FIFO of PIO state machine is empty.

```
u8 PioRxLevel(int pio, int sm);
u8 PioRxLevel_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Get number of elements in RX FIFO of PIO state machine.

```
Bool PioTxIsFull(int pio, int sm);
Bool PioTxIsFull_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Check if TX FIFO of PIO state machine is full.

```
Bool PioTxIsEmpty(int pio, int sm);
Bool PioTxIsEmpty_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Check if TX FIFO of PIO state machine is empty.

```
u8 PioTxLevel(int pio, int sm);
u8 PioTxLevel_hw(const pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Get number of elements in TX FIFO of PIO state machine.

```
void PioWriteWait(int pio, int sm, u32 data);
void PioWriteWait_hw(pio_hw_t* hw, int sm, u32 data);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

data ... data value

Write data to TX FIFO of PIO state machine, wait if TX FIFO is full.

```
u32 PioReadWait(int pio, int sm);
u32 PioReadWait_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Read data from RX FIFO of PIO state machine, wait if RX FIFO is empty.

```
void PioFifoClear(int pio, int sm);
void PioFifoClear_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Clear TX and RX FIFOs of PIO state machine.

```
void PioRxFifoClear(int pio, int sm);
void PioRxFifoClear_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Clear RX FIFO (read values).

```
void PioTxFifoClear(int pio, int sm);
void PioTxFifoClear_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Clear TX FIFO (executes OUT/PULL instruction; state machine should not be enabled).

```
void PioLoadProg(int pio, const u16* program, int len, int off);
```

pio ... PIO number 0..2

program ... array of program instructions

len ... length of program in number of instructions

off ... offset in PIO memory (program is wrapped to 32 instructions)

Load program into PIO memory. Jump instructions are auto-reallocated to new offset address (program is always compiled from offset 0).

```
void PioNopProg(int pio, int len, int off);
```

pio ... PIO number 0..2

len ... length of program, number of instructions

off ... offset in PIO memory (program is wrapped to 32 instructions)

Fill PIO program by NOP instructions (= instruction MOV Y,Y).

void PioClearProg(int pio, int len, int off);

pio ... PIO number 0..2

len ... length of program, number of instructions

off ... offset in PIO memory (program is wrapped to 32 instructions)

Clear PIO program by "JMP 0" instructions (= default reset value 0x0000).

void PioSetPin(int pio, int sm, int pin, int count, int val);

void PioSetPin_hw(pio_hw_t* hw, int sm, int pin, int count, int val);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

pin ... first pin index 0..29

count ... number of pins 1..30

val ... pin output value 0 or 1

Set output value of pin controlled by the PIO (after initialization with PioSetupGPIO(), but before running state machine). This is done by executing SET instruction on "victim" state machine (state machine should not be enabled).

void PioSetPinDir(int pio, int sm, int pin, int count, int dir);

void PioSetPinDir_hw(pio_hw_t* hw, int sm, int pin, int count, int dir);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

pin ... first pin index 0..29

count ... number of pins 1..30

dir ... pin direction, 1=output, 0=input

Set direction of pin controlled by the PIO (after initialization with PioSetupGPIO(), but before running state machine). This is done by executing SET instruction on "victim" state machine (state machine should not be enabled).

void PioSetAddr(int pio, int sm, int addr);

void PioSetAddr_hw(pio_hw_t* hw, int sm, int addr);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

addr ... program address 0 to 31

Set current program address of PIO state machine. This is done by executing JMP instruction on "victim" state machine (state machine should not be enabled).

```
void PioDebugClear(int pio, int sm);
void PioDebugClear_hw(pio_hw_t* hw, int sm);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

Reset debug flags of PIO state machine.

```
Bool PiolrqIsSet(int pio, int irq);
Bool PiolrqIsSet_hw(const pio_hw_t* hw, int irq);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

irq ... IRQ index 0 to 7

Check if IRQ is set.

```
void PiolrqClear(int pio, int irq);
void PiolrqClear_hw(pio_hw_t* hw, int irq);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

irq ... IRQ index 0 to 7

Clear IRQ request.

```
void PiolrqClearMask(int pio, int irq_mask);
void PiolrqClearMask_hw(pio_hw_t* hw, int irq_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

irq_mask ... IRQ bits B0 to B7

Clear IRQ requests by mask.

```
void PiolrqForce(int pio, int irq);
void PiolrqForce_hw(pio_hw_t* hw, int irq);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

irq ... IRQ index 0 to 7

Force IRQ request.

```
void PiolrqForceMask(int pio, int irq_mask);
void PiolrqForceMask_hw(pio_hw_t* hw, int irq_mask);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

irq_mask ... IRQ bits B0 to B7

Force IRQ requests by mask.

void PiolnBypass(int pio, int pin);
void PiolnBypass_hw(pio_hw_t* hw, int pin);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

pin ... GPIO pin 0 to 31

Switch GPIO synchronizer OFF, synchronizer will be bypassed (used for high speed inputs).

void PiolnBypassMask(int pio, u32 pin_mask);
void PiolnBypassMask_hw(pio_hw_t* hw, u32 pin_mask);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

pin_mask ... GPIO pin mask of bits B0 to B31

Switch GPIO synchronizer OFF by mask, synchronizer will be bypassed (used for high speed inputs).

void PiolnSync(int pio, int pin);
void PiolnSync_hw(pio_hw_t* hw, int pin);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

pin ... GPIO pin 0 to 31

Switch GPIO synchronizer ON, input will be synchronized (default state).

void PiolnSyncMask(int pio, u32 pin_mask);
void PiolnSyncMask_hw(pio_hw_t* hw, u32 pin_mask);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

pin_mask ... GPIO pin mask of bits B0 to B31

Switch GPIO synchronizer ON by mask, input will be synchronized (default state).

u32 PioGetPadOut(int pio);
u32 PioGetPadOut_hw(const pio_hw_t* hw);

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

Read current values currently driving from PIO to GPIOs outputs.

u32 PioGetPadOE(int pio);
u32 PioGetPadOE_hw(const pio_hw_t* hw);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
Read current values currently driving from PIO to GPIOs output enables (direction).

u8 PioMemSize(int pio);
u8 PioMemSize_hw(const pio_hw_t* hw);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
Get size of instruction memory in number of instructions (typically 32).

u8 PioSMCount(int pio);
u8 PioSMCount_hw(const pio_hw_t* hw);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
Get number of state machines of one PIO (typically 4).

u8 PioFifoDepth(int pio);
u8 PioFifoDepth_hw(const pio_hw_t* hw);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
Get depth of one FIFO in number of words (typically 4).

u8 PioIntRaw(int pio, int sm, int type);
u8 PioIntRaw_hw(const pio_hw_t* hw, int sm, int type);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
sm ... state machine 0 to 3
type ... interrupt type PIO_INT_*

Get raw unmasked interrupt state (returns 1 if raw interrupt is set).

void PioIntEnable(int pio, int sm, int irq, int type);
void PioIntEnable0_hw(pio_hw_t* hw, int sm, int type);
void PioIntEnable1_hw(pio_hw_t* hw, int sm, int type);

pio ... PIO number 0..2
hw ... pointer to PIO hardware interface (as returned from PioGetHw())
sm ... state machine 0 to 3
irq ... IRQ 0 or 1

type ... interrupt type **PIO_INT_***

Enable PIO interrupt. PIO interrupt type:

PIO_INT_RXNEMPTY RX FIFO is not empty

PIO_INT_TXNFULL TX FIFO is not full

PIO_INT_SM interrupt from state machine

```
void PioIntDisable(int pio, int sm, int irq, int type);
void PioIntDisable0_hw(pio_hw_t* hw, int sm, int type);
void PioIntDisable1_hw(pio_hw_t* hw, int sm, int type);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

irq ... IRQ 0 or 1

type ... interrupt type **PIO_INT_***

Disable PIO interrupt (default state).

```
void PioIntForce(int pio, int sm, int irq, int type);
void PioIntForce0_hw(pio_hw_t* hw, int sm, int type);
void PioIntForce1_hw(pio_hw_t* hw, int sm, int type);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

irq ... IRQ 0 or 1

type ... interrupt type **PIO_INT_***

Force PIO interrupt.

```
void PioIntUnforce(int pio, int sm, int irq, int type);
void PioIntUnforce0_hw(pio_hw_t* hw, int sm, int type);
void PioIntUnforce1_hw(pio_hw_t* hw, int sm, int type);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

irq ... IRQ 0 or 1

type ... interrupt type **PIO_INT_***

Unforce PIO interrupt (default state).

```
u8 PioIntStatus(int pio, int sm, int irq, int type);
u8 PioIntStatus0_hw(const pio_hw_t* hw, int sm, int type);
u8 PioIntStatus1_hw(const pio_hw_t* hw, int sm, int type);
```

pio ... PIO number 0..2

hw ... pointer to PIO hardware interface (as returned from PioGetHw())

sm ... state machine 0 to 3

irq ... IRQ 0 or 1

type ... interrupt type **PIO_INT_***

Get interrupt state (returns 1 if interrupt is set).

```
void PioSMInit(int pio, int sm);
```

pio ... PIO number 0..2

sm ... state machine 0 to 3

Initialize PIO state machine, prepare default state. Function is called from PioInit().

```
void PioSMInitCfg(int pio, int sm, int addr, const pio_sm_config* cfg);
```

pio ... PIO number 0..2

sm ... state machine 0 to 3

addr ... initial PC address

cfg ... pointer to configuration structure (NULL = use default configuration)

Initialite PIO state machine using configuration.

```
void PioInit(int pio);
```

pio ... PIO number 0..2

Initialize PIO with all state machines.

Encode instructions (without delay/sideset)

Instruction can be ORed with **PIO_ENCODE_SIDESET()** to encode delay/sideset.

```
PIO_ENCODE_JMP(cond, addr)
```

cond ... condition **PIO_COND_***

addr ... destination address 0 to 31

Encode JMP instruction.

```
PIO_ENCODE_WAIT(pol, src, inx)
```

pol ... polarity wait for 1 or 0

src ... source **PIO_WAIT_***

inx ... pin or IRQ index

Encode WAIT instruction.

PIO_ENCODE_IN(src, cnt)

src ... source **PIO_SRCDEST_***

cnt ... bit count 1..32

Encode IN instruction.

PIO_ENCODE_OUT(dst, cnt)

dst ... destination **PIO_SRCDEST_***

cnt ... bit count 1..32

Encode OUT instruction.

PIO_ENCODE_PUSH(iffull, block)

iffull ... 1 do nothing until threshold

block ... 1 stall if RX full

Encode PUSH instruction.

PIO_ENCODE_PULL(ifempty, block)

ifempty ... 1 do nothing until threshold

block ... 1 stall if TX empty

Encode PULL instruction.

PIO_ENCODE_MOV(dst, op, src)

dst ... destination **PIO_SRCDEST_***

op ... operation **PIO_OP_***

src ... source **PIO_SRCDEST_***

Encode MOV instruction.

PIO_ENCODE_IRQ(clr, wait, inx)

clr ... if 1 clear flag selected by inx

wait ... if 1 halt

inx ... IRQ index

Encode IRQ instruction.

PIO_ENCODE_SET(dst, data)

dst ... destination **PIO_SRCDEST_***

data ... data 5-bit value 0..31

Encode SET instruction.

PIO_NOP

Encode NOP pseudoinstruction (= MOV Y,Y side 0).

PIO_ENCODE_SIDESET(bits, opt, sideset, wait)

bits ... bits of sideset 0 to 5 including opt bit (0 = use only wait, 5 = use only sideset)
opt ... 1 use sideset, or 0 use only wait, or 0 if not using opt bit
sideset ... output sideset bits, or 0 use only wait
wait ... wait cycles 0 to 31 or less

Encode side-set of instructions (can be OR-ed with encoded instruction code).

PIO JMP instruction condition

PIO_COND_ALWAYS	always (no condition)
PIO_COND_NOTX	!X (if scratch X is zero)
PIO_COND_XDEC	X-- (if scratch X is non-zero, post-decrement)
PIO_COND_NOTY	!Y (if scratch Y is zero)
PIO_COND_YDEC	Y-- (if scratch Y is non-zero, post-decrement)
PIO_COND_NEQU	X!=Y (if scratch X not equal scratch Y)
PIO_COND_PIN	PIN (branch on input pin)
PIO_COND_NOSRE	!OSRE (if output shift register is not empty)

PIO WAIT source

PIO_WAIT_GPIO	wait for GPIO pin selected by index (not affected by mapping)
PIO_WAIT_PIN	wait for input pin selected by index (affected by mapping)
PIO_WAIT_IRQ	wait for IRQ flag selected by index

PIO instruction source/destination

PIO_SRCDST_PINS	PINS
PIO_SRCDST_X	X (scratch register X)
PIO_SRCDST_Y	Y (scratch register Y)
PIO_SRCDST_NULL	NULL (read all zeroes, write discard data)
PIO_SRCDST_DIRMOV	write PINDIRS, MOV write EXEC as instruction
PIO_SRCDST_STATUSPC	read STATUS, write PC
PIO_SRCDST_ISR	ISR
PIO_SRCDST_OSROUT	OSR, OUT write EXEC as instruction

PIO MOV operation

PIO_OP_NONE	... none
PIO_OP_INV	... invert (bitwise complement)
PIO_OP_REV	... bit-reverse

2.18. PLL - Phase-Locked Loop

Files: `sdk_pll.h`, `sdk_pll.c`

Config: `USE_PLL` (default 1)

The PLL (Phase-Locked Loop) serves as the main clock source for the processor. A reference clock (usually 12 MHz crystal oscillator) is fed to the PLL input, multiplied with the VCO using a feedback loop (400 to 1600 MHz), and then divided to the operating frequency (125 MHz). There are two PLLs in RP2040/RP2350. First (PLL0) generates system clock (125 MHz), second (PLL1) generates USB reference clock (48 MHz).

Result PLL output frequency is: $\text{freq} = (\text{XOSC} / \text{REFDIV}) * \text{FBDIV} / (\text{DIV1} * \text{DIV2})$

XOSC ... input clock, usually 12 MHz crystal oscillator

REFDIV ... divide input clock 1..63, minimum XOSC/REFDIV must be 5 MHz

FBDIV ... feedback divisor 16..320, XOSC/REFDIV*FBDIV should be 400..1600 MHz

DIV1, DIV2 ... post dividers 1..7, result system clock should be max 133 MHz

High VCO frequency minimises jittering, low VCO frequency reduces power consumption.

Some functions have two forms. Functions can be addressed either by the PLL index (parameter "int pll") or by a pointer to the hardware interface ("pll_hw_t* hw", as returned from `PiIGetHw()` function). Functions in the second case are marked with "_hw". Addressing by PLL index is easier to use, addressing by pointer can generate slightly better optimized code.

pll_hw_t* PiIGetHw(int pll);

pll ... index 0=PLL_SYS or 1=PLL_USB

Get hardware interface from the PLL index. Returns `pll_sys_hw` or `pll_usb_hw`.

u8 PiIGetInx(const pll_hw_t* hw);

hw ... pointer to hardware interface (as returned from `PiIGetHw()`)

Get PLL index from hardware interface.

u8 PiIGetRefDiv(int pll);

u8 PiIGetRefDiv_hw(const pll_hw_t* hw);

pll ... index 0=PLL_SYS or 1=PLL_USB

hw ... pointer to hardware interface (as returned from `PiIGetHw()`)

Get PLL refdiv divider (1..63).

int PiIGetFBDiv(int pll);

int PiIGetFBDiv_hw(const pll_hw_t* hw);

pll ... index 0=PLL_SYS or 1=PLL_USB

hw ... pointer to hardware interface (as returned from `PiIGetHw()`)

Get PLL feedback divisor (16..320).

```
u8 PIIGetDiv1(int pll);
u8 PIIGetDiv1_hw(const pll_hw_t* hw);
```

pll ... index 0=PLL_SYS or 1=PLL_USB
hw ... pointer to hardware interface (as returned from PIIGetHw())
Get PLL post divider 1.

```
u8 PIIGetDiv2(int pll);
u8 PIIGetDiv2_hw(const pll_hw_t* hw);
```

pll ... index 0=PLL_SYS or 1=PLL_USB
hw ... pointer to hardware interface (as returned from PIIGetHw())
Get PLL post divider 2.

```
u32 PIIGetVco(int pll);
u32 PIIGetVco_hw(const pll_hw_t* hw);
```

pll ... index 0=PLL_SYS or 1=PLL_USB
hw ... pointer to hardware interface (as returned from PIIGetHw())
Get current PLL VCO frequency in Hz, calculated using XOSC clock and current settings of refdiv and fbdv.

```
u32 PIISetup(int pll, int refdiv, int fbdv, int div1, int div2);
```

pll ... index 0=PLL_SYS or 1=PLL_USB
refdiv ... divide input reference clock, 1..63 (minimum XOSC/refdiv is 5 MHz)
fbdv ... feedback divisor, 16..320 (XOSC/refdiv*fbdv must be 400..1600 MHz)
div1 ... post divider 1, 1..7
div2 ... post divider 2, 1..7 (should be div1 >= div2, but auto-corrected)

Setup PLL (returns result frequency in Hz). XOSC must be initialized to get its frequency from the table. All clocks should be disconnected from the PLL. Result frequency = (XOSC / refdiv) * fbdv / (div1 * div2). Do not call this function directly if SYS clock is connected, call the ClockPIISysSetup() function instead.

```
Bool PIICalc(u32 reqkhz, u32 input, u32 vcomin, u32 vcomax, Bool lowvco,
u32* outkhz, u32* outvco, u16* outfbdv, u8* outpd1, u8* outpd2);
```

reqkhz ... required output frequency in kHz
input ... PLL input reference frequency in kHz (default 12000)
vcomin ... minimal VCO frequency in kHz (default 400000)
vcomax ... maximal VCO frequency in kHz (default 1600000)
lowvco ... prefer low VCO (lower power but more jitter)

outputs:

outkhz ... output achieved frequency in kHz (0=not found)
outvco ... output VCO frequency in kHz

outfbdiv ... output fbdv (16..320)
outpd1 ... output postdiv1 (1..7)
outpd2 ... output postdiv2 (1..7)

Search PLL setup. Returns True if precise frequency has been found, or near frequency used otherwise if False.

Bool PIICalcDef(u32 reqkhz, u32* outkhz, u32* outvco, u16* outfbdiv, u8* outpd1, u8* outpd2);

reqkhz ... required output frequency in kHz

outputs:

outkhz ... output achieved frequency in kHz (0=not found)
outvco ... output VCO frequency in kHz
outfbdiv ... output fbdv (16..320)
outpd1 ... output postdiv1 (1..7)
outpd2 ... output postdiv2 (1..7)

Search PLL setup, using defaults. Returns true if precise frequency has been found, or near frequency used otherwise if False.

u32 PIISetFreq(int pll, u32 freq);

pll ... index 0=PLL_SYS or 1=PLL_USB
freq ... required frequency in kHz

Setup PLL frequency in kHz (returns result frequency in kHz, or 0 if cannot setup). Do not call this function directly if SYS clock is connected, call the ClockPIISysFreq() function instead.

2.19. PowMan - Power Manager

Files: `sdk_powman.h`, `sdk_powman.c`

Config: - (only RP2350)

`void PowMan_Start();`

Start timer of power manager - start AON always-on timer.

`void PowMan_Stop();`

Stop timer of power manager - stop AON always-on timer.

`Bool PowMan_IsRunning();`

Check if timer of power manager is running.

`void PowMan_SetTime(u64 time);`

`time` ... time in [ms]

Set time in [ms] of power manager - set time of AON always-on timer.

`u32 PowMan_GetTimeLow();`

Fast get time LOW of power manager in [ms] - get time of AON always-on timer.

`u64 PowMan_GetTime();`

Get time of power manager in [ms] - get time of AON always-on timer.

`void PowMan_SetSrc(int src, u32 freq);`

`src` ... clock source `POWMAN_SRC_*`

`freq` ... clock frequency at [Hz] (must be 1000 or 1 for GPIO input)

Setup power manager timer source. GPIO with frequency 1 Hz will synchronize low power or crystal oscillator.

<code>POWMAN_SRC_LPOSC</code>	0	low power oscillator (usually 32 kHz)
<code>POWMAN_SRC_XOSC</code>	1	crystal oscillator
<code>POWMAN_SRC_GPIO12</code>	12	GPIO12 input (only 1 kHz or 1 Hz)
<code>POWMAN_SRC_GPIO14</code>	14	GPIO14 input (only 1 kHz or 1 Hz)
<code>POWMAN_SRC_GPIO20</code>	20	GPIO20 input (only 1 kHz or 1 Hz)
<code>POWMAN_SRC_GPIO22</code>	22	GPIO22 input (only 1 kHz or 1 Hz)

`u8 PowMan_PowerState();`

Get power state of domains - returns combination of `POWMAN_DOMAIN_*`, 1=powered up, 0=powered down.

`POWMAN_DOMAIN_SRAM1` B0 SRAM power domain 1, upper half

POWMAN_DOMAIN_SRAM0	B1	SRAM power domain 0, lower half
POWMAN_DOMAIN_XIP	B2	XIP cache SRAM and Boot RAM
POWMAN_DOMAIN_SWCORE	B3	switched core

Bool PowMain_SetPowerState(int state);

state ... combination of **POWMAN_DOMAIN_***, 1=powered up, 0=powered down

Set power state of domains. Returns False if new state is declined.

void PowMan_ClearAlarm();

Clear alarm. Alarm must be disabled, or will be fired again.

void PowMan_AlarmOn(u64 time);

time ... time in [ms]

Set alarm on at time in [ms].

void PowMan_AlarmOff();

Set alarm off.

void PowMan_WakeUpOn(u64 time);

time ... time in [ms]

Setup alarm to wake up at time in [ms].

void PowMan_WakeUpOff();

Disable wake up.

void PowMan_GpioOn(int event, int gpio, Bool edge, Bool high);

event ... wakeup event 0..3

gpio ... gpio to wake up from (0..47)

edge ... True=edge sensitive, False=level sensitive

high ... True=active high or rising, False=active low or falling

Setup wake up by GPIO.

void PowMan_GpioOff(int event);

event ... wakeup event 0..3

Disable wake up by GPIO.

void PowMan_WakeAllOff();

Disable all wake ups.

void VregSetVoltage(int vreg);

vreg ... voltage **VREG_VOLTAGE_*** (default **VREG_VOLTAGE_1_10**)

Set voltage **VREG_VOLTAGE_***.

RP2350: OR required value with the **VREG_VOLTAGE_UNLOCK** flag, to enable unlock higher voltages than 1.30V

u8 VregVoltage();

Get current voltage **VREG_VOLTAGE_***.

float VregVoltageFloat();

Get current voltage in volts.

Bool VregIsOk();

Check if voltage is correctly regulated.

void VregWait();

Wait for voltage regulated state.

2.20. PWM - Pulse Width Modulation

Files: `sdk_pwm.h`, `sdk_pwm.c`

Config: `USE_PWM` (default 1)

Pulse width modulation can generate smoothly varying average voltage using pulses with controlled width. RP2040 has 8 identical PWM slices, PWM0..PWM7. RP2350 has 12 identical PWM slices, PWM0..PWM11. Each slice can drive two PWM output signals, A and B.

PWM mapping on RP2040:

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

PWM mapping on RP2350:

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PWM	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
PWM	8A	8B	9A	9B	10A	10B	11A	11B	8A	8B	9A	9B	10A	10B	11A	11B

For the RP2350A variant of the QFN-60 package, although PWM channels 8 to 11 are not pinned out, PWM can still be used as timers.

Same PWM output channel can be selected on two GPIO pins. If two PWM B pins are used as input, multiply GPIO pins will be OR of those inputs.

Some functions have two forms. Functions can be addressed either by the PWM slice index (parameter "int pwm") or by a pointer to the hardware interface ("pwm_slice_hw_t* hw", as returned from `PWM_GetHw()` function). Functions in the second case are marked with `_hw`. Addressing by PWM slice index is easier to use, addressing by pointer can generate slightly better optimized code.

Clkdiv mode

<code>PWM_DIV_FREE</code>	free-running at rate of divider
<code>PWM_DIV_HIGH</code>	divider gated by PWM B pin
<code>PWM_DIV_RISE</code>	divider advances with rising edge of PWM B pin
<code>PWM_DIV_FALL</code>	divider advances with falling edge of PWM B pin

Output channels

PWM_CHAN_A (0) output channel A
PWM_CHAN_B (1) output channel B

void PWM_Reset(int pwm);

pwm ... PWM slice index 0..7 or 0..11

Reset PWM slice to default values.

u8 PWM_GpioToChan(int gpio);

PWM_GPIOTOCHAN(gpio);

gpio ... GPIO pin 0..29 or 0..47

Convert GPIO pin to PWM output channels **PWM_CHAN_*** (returns 0=A or 1=B).

u8 PWM_GpioToSlice(int gpio);

PWM_GPIOTOSLICE(gpio);

gpio ... GPIO pin 0..29 or 0..47

Convert GPIO pin to PWM slice (returns 0..7 or 0..11).

pwm_slice_hw_t* PWM_GetHw(int pwm);

pwm ... PWM slice index 0..7 or 0..11

Get PWM slice hardware interface from PWM slice index.

u8 PWM_GetIdx(const pwm_slice_hw_t* hw);

hw ... pointer to PWM slice hardware interface

Get PWM slice index from PWM slice hardware interface.

u8 PWM_GetDreq(int pwm);

pwm ... PWM slice index 0..7 or 0..11

Get DREQ index.

PWM configuration structure (pwm_config)

pwm_config PWM_CfgDef(void);

Get default configuration structure.

void PWM_Cfg(int pwm, const pwm_config* cfg, Bool start);

void PWM_Cfg_hw(pwm_slice_hw_t* hw, const pwm_config* cfg, Bool start);

pwm ... PWM slice index 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

start ... start PWM

Apply PWM configuration.

void PWM_CfgPhaseCorrect(pwm_config* cfg);

cfg ... pointer to configuration structure

Config enable phase-correct modulation.

void PWM_CfgTrailingEdge(pwm_config* cfg);

cfg ... pointer to configuration structure

Config enable trailing-edge modulation (disable phase-correct modulation).

void PWM_CfgInvEnable(pwm_config* cfg, int chan);

cfg ... pointer to configuration structure

chan ... output channel **PWM_CHAN_***

Config inverting output channel enable.

void PWM_CfgInvDisable(pwm_config* cfg, int chan);

cfg ... pointer to configuration structure

chan ... output channel **PWM_CHAN_***

Config inverting output channel disable.

void PWM_CfgDivMode(pwm_config* cfg, int divmode);

cfg ... pointer to configuration structure

divmode ... divider mode **PWM_DIV_***

Config set divider mode **PWM_DIV_*** (default free-running).

void PWM_CfgClkDiv(pwm_config* cfg, int clkdiv);

cfg ... pointer to configuration structure

clkdiv ... clock divider * 16, value 0..0xffff (lower 4 bits are fractional part,
upper 8 bits are integer part)

Config set clock divider for free-running mode.

void PWM_CfgClkDivFloat(pwm_config* cfg, float clkdiv);

cfg ... pointer to configuration structure

clkdiv ... clock divider

Config set clock divider for free-running mode, as float.

void PWM_CfgTop(pwm_config* cfg, u16 top);

cfg ... pointer to configuration structure

top ... top value

Config set counter top wrap value (counter period = top + 1).

PWM setup

void PWM_GpioInit(int gpio);

gpio ... GPIO pin 0..29 or 0..47

Initialize GPIO to function of PWM pin.

void PWM_Enable(int pwm);

void PWM_Enable_hw(pwm_slice_hw_t* hw);

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Enable PWM slice.

void PWM_EnableMask(int mask);

mask ... PWM bit mask (bits 0..7 or 0..11)

Enable multiple PWM slices by mask.

void PWM_Disable(int pwm);

void PWM_Disable_hw(pwm_slice_hw_t* hw);

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Disable PWM slice.

void PWM_DisableMask(int mask);

mask ... PWM bit mask (bits 0..7 or 0..11)

Disable multiple PWM slices by mask.

void PWM_PhaseCorrect(int pwm);

void PWM_PhaseCorrect_hw(pwm_slice_hw_t* hw);

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Enable phase-correct modulation.

void PWM_TrailingEdge(int pwm);

void PWM_TrailingEdge_hw(pwm_slice_hw_t* hw);

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Enable trailing-edge modulation (disable phase-correct modulation).

```
void PWM_InvEnable(int pwm, int chan);
void PWM_InvEnable_hw(pwm_slice_hw_t* hw, int chan);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

chan ... output channel **PWM_CHAN_***

Inverting output channel enable.

```
void PWM_InvDisable(int pwm, int chan);
void PWM_InvDisable_hw(pwm_slice_hw_t* hw, int chan);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

chan ... output channel **PWM_CHAN_***

Inverting output channel disable.

```
void PWM_DivMode(int pwm, int divmode);
void PWM_DivMode_hw(pwm_slice_hw_t* hw, int divmode);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

divmode ... divider mode **PWM_DIV_***

Set divider mode **PWM_DIV_***.

```
void PWM_Retard(int pwm);
void PWM_Retard_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Retard phase of counter by 1 (counter must be running).

```
void PWM_RetardWait(int pwm);
void PWM_RetardWait_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Retard phase of counter by 1 and wait to complete (must be running).

```
void PWM_Advance(int pwm);
void PWM_Advance_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Advance phase of counter by 1 (counter must be running).

```
void PWM_AdvanceWait(int pwm);
void PWM_AdvanceWait_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())
Advance phase of counter by 1 and wait to complete (must be running).

```
void PWM_ClkDiv(int pwm, int clkdiv);
void PWM_ClkDiv_hw(pwm_slice_hw_t* hw, int clkdiv);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())
clkdiv ... clock divider * 16, value 0..0xffff (lower 4 bits are fractional part,
upper 8 bits are integer part)

Set clock divider for free-running mode. Counter is fed from system clock **CLK_SYS**.

```
void PWM_ClkDivFloat(int pwm, float clkdiv);
void PWM_ClkDivFloat_hw(pwm_slice_hw_t* hw, float clkdiv);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())
clkdiv ... clock divider

Set clock divider for free-running mode as float. Counter is fed from system clock **CLK_SYS**.

```
u16 PWM_GetClkDiv(int pwm);
u16 PWM_GetClkDiv_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Get clock divider. Returns 12-bit value of clock divider*16. Lower 4 bits are fractional part,
upper 8 bits are integer part.

```
void PWM_Count(int pwm, u16 cnt);
void PWM_Count_hw(pwm_slice_hw_t* hw, u16 cnt);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())
cnt ... clock counter value

Set clock counter value (16-bit value).

```
u16 PWM_GetCount(int pwm);
u16 PWM_GetCount_hw(pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11
hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Get clock counter value.

```
void PWM_Comp(int pwm, int chan, u16 cmp);
void PWM_Comp_hw(pwm_slice_hw_t* hw, int chan, u16 cmp);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

chan ... output channel **PWM_CHAN_***

cmp ... compare value

Set compare value of output channel **PWM_CHAN_*** (cmp is 16-bit value).

```
void PWM_Comp2(int pwm, u16 cmp_a, u16 cmp_b);
void PWM_Comp2_hw(pwm_slice_hw_t* hw, u16 cmp_a, u16 cmp_b);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

cmp_a ... compare value for A channel

cmp_b ... compare value for B channel

Set compare values of both output channels (cmp_* are 16-bit values).

```
u16 PWM_GetComp(int pwm, int chan);
u16 PWM_GetComp_hw(const pwm_slice_hw_t* hw, int chan);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

chan ... output channel **PWM_CHAN_***

Get compare value of output channel **PWM_CHAN_***.

```
void PWM_Top(int pwm, u16 top);
void PWM_Top_hw(pwm_slice_hw_t* hw, u16 top);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

top ... top value

Set counter top wrap value (counter period = top + 1).

```
u16 PWM_GetTop(int pwm);
u16 PWM_GetTop_hw(const pwm_slice_hw_t* hw);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Get counter top wrap value.

```
void PWM_Clock(int pwm, u32 freq);
void PWM_Clock_hw(pwm_slice_hw_t* hw, u32 freq);
```

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

freq ... frequency, for clk_sys = 125 MHz in range 488 kHz to 125 MHz

Set clock frequency in Hz for free-running mode. For TOP=256 is sampling rate 1.9 kHz to 488 kHz.

**u32 PWM_GetClock(int pwm);
u32 PWM_GetClock_hw(pwm_slice_hw_t* hw);**

pwm ... PWM slice 0..7 or 0..11

hw ... pointer to PWM slice hardware interface (as returned from PWM_GetHw())

Get real frequency of the PWM clock (in Hz).

u32 PWM_FindSysClk(u32 min_hz, u32 max_hz, u32 freq);

min_hz ... minimal system frequency in Hz

max_hz ... maximal system frequency in Hz

freq ... required PWM clock frequency in Hz

Find system clock in Hz that sets the most accurate PWM clock frequency.

PWM interrupt

Bool PWM_IntRaw(int pwm);

pwm ... PWM slice 0..7 or 0..11

Get raw interrupt flag (not masked by Enable).

void PWM_IntClear(int pwm);

pwm ... PWM slice 0..7 or 0..11

Clear interrupt flag.

void PWM_IntEnable(int pwm); // IRQ_PWM_WRAP
void PWM_Int1Enable(int pwm); // RP2350: IRQ_PWM_WRAP_1

pwm ... PWM slice 0..7 or 0..11

Enable interrupt.

void PWM_IntDisable(int pwm); // IRQ_PWM_WRAP
void PWM_Int1Disable(int pwm); // RP2350: IRQ_PWM_WRAP_1

pwm ... PWM slice 0..7 or 0..11

Disable interrupt.

void PWM_IntForce(int pwm); // IRQ_PWM_WRAP
void PWM_Int1Force(int pwm); // RP2350: IRQ_PWM_WRAP_1

pwm ... PWM slice 0..7 or 0..11

Force interrupt.

```
void PWM_IntUnforce(int pwm);           // IRQ_PWM_WRAP
void PWM_Int1Unforce(int pwm);          // RP2350: IRQ_PWM_WRAP_1


pwm ... PWM slice 0..7 or 0..11


```

Unforce interrupt.

```
Bool PWM_IntIsForced(int pwm);          // IRQ_PWM_WRAP
Bool PWM_Int1IsForced(int pwm);          // RP2350: IRQ_PWM_WRAP_1


pwm ... PWM slice 0..7 or 0..11


```

Check if interrupt is forced.

```
Bool PWM_IntState(int pwm);             // IRQ_PWM_WRAP
Bool PWM_Int1State(int pwm);            // RP2350: IRQ_PWM_WRAP_1


pwm ... PWM slice 0..7 or 0..11


```

Get interrupt status (masked by EN).

2.21. QMI - QSPI memory interface

Files: **sdk_qmi.h, sdk_qmi.c**

Config: - (only RP2350)

QMI (QSPI memory interface) is used to communicate with external Flash and PSRAM, using QSPI interface. Usual device index: 0 = QSPI Flash (base address 0x10000000), 1 = PSRAM. RP2040 uses SSI interface instead of QMI.

void QMI_DirectEnable();

Enable direct mode (= controlled by software).

void QMI_DirectDisable();

Disable direct mode, set auto mode (= controlled by hardware).

Bool QMI_Busy();

Check if transfer is busy.

void QMI_Wait();

Wait if direct is busy.

void QMI_CsLow(int dev);

dev ... device 0=Flash, 1=PSRAM

Set QMI CSn level ON = LOW.

void QMI_CsHigh(int dev);

dev ... device 0=Flash, 1=PSRAM

Set QMI CSn level OFF = HIGH.

void QMI_AutoCsEnable(int dev);

dev ... device 0=Flash, 1=PSRAM

Enable auto assert CSn on busy.

void QMI_AutoCsDisable(int dev);

dev ... device 0=Flash, 1=PSRAM

Disable auto assert CSn on busy.

Bool QMI_TxIsFull();

dev ... device 0=Flash, 1=PSRAM

Check if direct TX FIFO is full.

Bool QMI_TxIsEmpty();

Check if direct TX FIFO is empty.

int QMI_TxLevel();

Get current direct TX FIFO level. Returns 0..7.

Bool QMI_RxIsEmpty();

Check if direct RX FIFO is empty.

Bool QMI_RxIsFull();

Check if direct RX FIFO is full.

int QMI_RxLevel();

Get current direct RX FIFO level. Returns 0..7.

void QMI_SetDirClkDiv(int clkdiv);

clkdiv ... 1..256 (default value 6)

Set clock divisor for direct mode.

int QMI_DirClkDiv();

Get clock divisor for direct mode. Returns 1..256, default 6.

void QMI_SetRxDelay(int delay);

delay ... 0..3 (default 0)

Set delay read data sample timing in half of sys_clk for direct mode.

int QMI_RxDelay();

Get delay read data sample timing in half of sys_clk for direct mode. Returns 0..3, default 0.

void QMI_Send(u8 data);

data ... data to send

Push data in direct mode (width = SPI, data width = 8 bits).

u8 QMI_Recv();

Read data in direct mode (data width = 8 bits).

int QMI_ClkDiv(int dev);

dev ... device 0=Flash, 1=PSRAM

Get QMI speed of auto mode. Returns 1..256, default 4.

void QMI_SetClkDiv(int dev, int clkdiv);

dev ... device 0=Flash, 1=PSRAM

clkdiv ... 1..256

Set QMI speed 1..256 of auto mode.

void QMI_FlashQspi(int clkdiv);

clkdiv ... 1..256 (default value **FLASHQSPI_CLKDIV_DEF**=2)

Set flash to fast QSPI mode. Supported devices: Winbond W25Q080, W25Q16JV, AT25SF081, S25FL132K0. Raspberry Pico cointains W25Q16JVUXIQ, Raspberry Pico 2 cointains W25Q32RVXHJQ.

void QMI_InitFlash(int clkdiv);

clkdiv ... 1..256 (default value **FLASHQSPI_CLKDIV_DEF**=2)

Initialize Flash interface. Supported devices: Winbond W25Q080, W25Q16JV, AT25SF081, S25FL132K0. Raspberry Pico cointains W25Q16JVUXIQ, Raspberry Pico 2 cointains W25Q32RVXHJQ

2.22. QSPI - QSPI Flash Pins

Files: `sdk_qspi.h`, `sdk_qspi.c`

Config: -

The RP2040/RP2350 has 6 pins that are equipped with interface supporting SPI, Dual-SPI or Quad-SPI protocol. These pins are standardly used to connect Flash memory. If not used with Flash memory, they can be used as GPIO pins.

Most functions have two forms. Functions can be addressed either by the QSPI pad or pin number (parameter "int pad" or "int pin") or by a pointer to the hardware interface ("`io32* hw`" or "`ioqspi_status_ctrl_hw_t* hw`"). Functions in the second case are marked with "`_hw`". Addressing by par/pin number is easier to use, addressing by pointer can generate slightly better optimized code.

Important change of the RP2350 compared to the RP2040: The "isolation latches" bit was added to the QSPI settings, which is active after reset and after initialization. This bit ensures that changes made to the pin settings (direction and output state) are first saved to the auxiliary latch, and only after the "isolation latches" bit is disabled are changes reflected to the pin. This prevents the occurrence of transients on the output of the pin. After setting the pin configuration, the "isolation latches" bit must be disabled.

Warning - different mapping of pin and pad numbers is used.

QSPI pad indices

<code>QSPI_PAD_SCLK</code>	0
<code>QSPI_PAD_SD0</code>	1
<code>QSPI_PAD_SD1</code>	2
<code>QSPI_PAD_SD2</code>	3
<code>QSPI_PAD_SD3</code>	4
<code>QSPI_PAD_SS</code>	5

QSPI pin indices - RP2040

<code>QSPI_PIN_SCLK</code>	0
<code>QSPI_PIN_SS</code>	1
<code>QSPI_PIN_SD0</code>	2
<code>QSPI_PIN_SD1</code>	3
<code>QSPI_PIN_SD2</code>	4
<code>QSPI_PIN_SD3</code>	5

QSPI pin indices - RP2350

<code>QSPI_PIN_DP</code>	0
<code>QSPI_PIN_DM</code>	1

QSPI_PIN_SCLK	2
QSPI_PIN_SS	3
QSPI_PIN_SD0	4
QSPI_PIN_SD1	5
QSPI_PIN_SD2	6
QSPI_PIN_SD3	7

QSPI pin functions

QSPI_FNC_XIP	0	external Flash
QSPI_FNC_SIO	5	SIO (GPIO)
QSPI_FNC_NULL	31	no function
QSPI_FNC_UART	2	UART (only RP2350)
QSPI_FNC_I2C	3	I2C (only RP2350)
QSPI_FNC_AUX	11	UART AUX (only RP2350)

QSPI interrupt event

QSPI_EVENT_LEVELLOW	0	// level low
QSPI_EVENT_LEVELHIGH	1	// level high
QSPI_EVENT_EDGELOW	2	// edge low
QSPI_EVENT_EDGEHIGH	3	// edge high

Pad control (use pads QSPI_PAD_*)

Warning: pins and pads use different numbering (use predefined constants).

void QSPI_Voltage3V3();

QSPI set voltage to 3.3V (DVDD >= 2V5; default state).

void QSPI_Voltage1V8();

QSPI set voltage to 1.8V (DVDD <= 1V8).

io32* QSPI_PadHw(int pad);

pad ... pad number 0..5 QSPI_PAD_*

Get pointer to pad control interface.

void QSPI_PadInit(int pad);

void QSPI_PadInit_hw(io32* hw);

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

Init pad to default state - slow slew rate, enable schmitt, pull-down, 4mA, in/out enable. On RP2350, pas isolation is enabled - it must be disabled after setting pad configuration.

```
void QSPI_IsolationEnable(int pad);
void QSPI_IsolationEnable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

Only RP2350. Enable isolation latches (default state). If pad isolation is enabled, all output settings of the pad are latched. Disable pad isolation after pad is configured.

```
void QSPI_IsolationDisable(int pad);
void QSPI_IsolationDisable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

Only RP2350. Disable isolation latches. If pad isolation is disable, all output settings are transparent to the pad.

```
void QSPI_OutEnable(int pad);
void QSPI_OutEnable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

QSPI output enable (default state). Use the QSPI_DirOut() function to set the pin direction to the output.

```
void QSPI_OutDisable(int pad);
void QSPI_OutDisable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

QSPI output disable. Disable has priority over QSPI_DirOut().

```
void QSPI_InEnable(int pad);
void QSPI_InEnable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

Enable pad input (default state).

```
void QSPI_InDisable(int pad);
void QSPI_InDisable_hw(io32* hw);
```

pad ... pad number 0..5 QSPI_PAD_*

hw ... pointer to pad control word (as returned by QSPI_PadHw())

Disable pad input.

```
void QSPI_Drive2mA(int pad);
void QSPI_Drive2mA_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set output strength to 2 mA.
```

```
void QSPI_Drive4mA(int pad);
void QSPI_Drive4mA_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set output strength to 4 mA (default state).
```

```
void QSPI_Drive8mA(int pad);
void QSPI_Drive8mA_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set output strength to 8 mA.
```

```
void QSPI_Drive12mA(int pad);
void QSPI_Drive12mA_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set output strength to 12 mA.
```

```
void QSPI_Drive(int pin, int drive);
void QSPI_Drive_hw(io32* hw, int drive);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
    driver ... QSPI_DRIVE_*
Set drive strength.
```

```
void QSPI_NoPull(int pad);
void QSPI_NoPull_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set no pulls.
```

```
void QSPI_PullDown(int pad);
void QSPI_PullDown_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
```

hw ... pointer to pad control word (as returned by QSPI_PadHw())
Set pull down (default state).

```
void QSPI_PullUp(int pad);
void QSPI_PullUp_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Set pull up.

```
void QSPI_BusKeep(int pad);
void QSPI_BusKeep_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Set bus keep (weak pull up and down).

```
void QSPI_SchmittEnable(int pad);
void QSPI_SchmittEnable_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Enable schmitt trigger (use hysteresis on input; default state).

```
void QSPI_SchmittDisable(int pad);
void QSPI_SchmittDisable_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Disable schmitt trigger (do not use hysteresis on input).

```
void QSPI_Slow(int pad);
void QSPI_Slow_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Use slow slew rate control on output (default state).

```
void QSPI_Fast(int pad);
void QSPI_Fast_hw(io32* hw);
    pad ... pad number 0..5 QSPI_PAD_*
    hw ... pointer to pad control word (as returned by QSPI_PadHw())
```

Use fast slew rate control on output.

Pin control (use pins QSPI_PIN_*)

Warning: pins and pads use different numbering (use predefined constants).

ioqspi_status_ctrl_hw_t* QSPI_PinHw(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Get pointer to pin control interface.

u8 QSPI_OutPeri(int pin);

u8 QSPI_OutPeri_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Only RP2040. Get output (0 or 1) from peripheral, before override.

u8 QSPI_OutPad(int pin);

u8 QSPI_OutPad_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Get output (0 or 1) to pad, after override.

u8 QSPI_OePeri(int pin);

u8 QSPI_OePeri_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Only RP2040. Get output enable (0 or 1) from peripheral, before override.

u8 QSPI_OePad(int pin);

u8 QSPI_OePad_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Get output enable (0 or 1) to pad, after override.

u8 QSPI_InPad(int pin);

u8 QSPI_InPad_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Get input (0 or 1) from pad, before override.

u8 QSPI_InPeri(int pin);

u8 QSPI_InPeri_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Only RP2040. Get input (0 or 1) to peripheral, after override.

u8 QSPI_IrqPad(int pin);

u8 QSPI_IrqPad_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Only RP2040. Get interrupt (0 or 1) from pad, before override.

u8 QSPI_IrqProc(int pin);

u8 QSPI_IrqProc_hw(const ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Get interrupt (0 or 1) to processor, after override.

void QSPI_OutOverNormal(int pin);

void QSPI_OutOverNormal_hw(ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output pin override to normal state (default state).

void QSPI_OutOverInvert(int pin);

void QSPI_OutOverInvert_hw(ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output pin override to invert state.

void QSPI_OutOverLow(int pin);

void QSPI_OutOverLow_hw(ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output pin override to low state.

void QSPI_OutOverHigh(int pin);

void QSPI_OutOverHigh_hw(ioqspi_status_ctrl_hw_t* hw);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output pin override to high state.

```
void QSPI_OEOverNormal(int pin);
void QSPI_OEOverNormal_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output enable pin override to normal state (default state).

```
void QSPI_OEOverInvert(int pin);
void QSPI_OEOverInvert_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output enable pin override to invert state.

```
void QSPI_OEOverDisable(int pin);
void QSPI_OEOverDisable_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output enable pin override to disable state.

```
void QSPI_OEOverEnable(int pin);
void QSPI_OEOverEnable_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Output enable pin override to enable state.

```
void QSPI_InOverNormal(int pin);
void QSPI_InOverNormal_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Input pin override to normal state (default state).

```
void QSPI_InOverInvert(int pin);
void QSPI_InOverInvert_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Input pin override to invert state.

```
void QSPI_InOverLow(int pin);
void QSPI_InOverLow_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Input pin override to low state.

```
void QSPI_InOverHigh(int pin);
void QSPI_InOverHigh_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set Input pin override to high state.

```
void QSPI IRQOverNormal(int pin);
void QSPI IRQOverNormal_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set IRQ pin override to normal state (default state).

```
void QSPI IRQOverInvert(int pin);
void QSPI IRQOverInvert_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set IRQ pin override to invert state.

```
void QSPI IRQOverLow(int pin);
void QSPI IRQOverLow_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set IRQ pin override to low state.

```
void QSPI IRQOverHigh(int pin);
void QSPI IRQOverHigh_hw(ioqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Set IRQ pin override to high state.

```
void QSPI_Fnc(int pin, int fnc);
void QSPI_Fnc_hw(ioqspi_status_ctrl_hw_t* hw, int fnc);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

fnc ... pin function QSPI_FNC_*

Set QSPI function QSPI_FNC_*, reset overrides to normal mode, does not affect pad settings. Default QSPI_FNC_NULL. RP2350: After setup pin, disable isolation latches with QSPI_IsolationDisable()

```
u8 QSPI_GetFnc(int pin);
u8 QSPI_GetFnc_hw(const iqspi_status_ctrl_hw_t* hw);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

hw ... pointer to pin hardware interface (as returned by QSPI_PinHw())

Get current QSPI function QSPI_FNC_*.

Pin interrupt control (use pins QSPI_PIN_*)

Warning: pins and pads use different numbering (use predefined constants).

```
u8 QSPI_IRQRaw(int pin);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Get raw interrupt status - for both CPU. Returns events - bit mask with IRQ_EVENT_* of incoming events.

```
void QSPI_IRQAck(int pin, int events);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with IRQ_EVENT_* of events to acknowledge

Acknowledge IRQ interrupts for both CPU - clear raw interrupt status.

```
void QSPI_IRQEnableCpu(int cpu, int pin, int events);
```

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with IRQ_EVENT_* of events to enable

Enable IRQ interrupt for selected CPU core.

```
void QSPI_IRQEnable(int pin, int events);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with IRQ_EVENT_* of events to enable

Enable IRQ interrupt for current CPU core.

```
void QSPI_IRQEnableDorm(int pin, int events);
```

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with IRQ_EVENT_* of events to enable

Enable IRQ interrupt for dormant wake.

```
void QSPI_IRQDisableCpu(int cpu, int pin, int events);
```

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with IRQ_EVENT_* of events to disable

Disable IRQ interrupt for selected CPU core.

void QSPI_IRQDisable(int pin, int event);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to disable

Disable IRQ interrupt for current CPU core.

void QSPI_IRQDisableDorm(int pin, int event);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to disable

Disable IRQ interrupt for dormant wake.

void QSPI_SetHandler(irq_handler_t handler);

handler ... pointer to interrupt handler function

Set IRQ handler.

void QSPI_IRQForceCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for selected CPU core. Forcing will trigger IRQ interrupt. Force request should be disabled at IRQ handler.

void QSPI_IRQForce(int pin, int events);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for current CPU core. Forcing will trigger IRQ interrupt. Force request should be disabled at IRQ handler.

void QSPI_IRQForceDorm(int pin, int events);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to force

Force IRQ interrupt for dormant wake. Forcing will trigger IRQ interrupt. Force request should be disabled at IRQ handler.

void QSPI_IRQUnforceCpu(int cpu, int pin, int events);

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for selected CPU core.

void QSPI_IRQUnforce(int pin, int events);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for current CPU core.

void QSPI_IRQUnforceDorm(int pin, int events);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

events ... bit mask with **IRQ_EVENT_*** of events to unforce

Clear force IRQ interrupt for dormant wake.

u8 QSPI_IRQIsForcedCpu(int cpu, int pin);

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check if IRQ interrupt is forced for selected CPU. Returns bit mask with **QSPI_EVENT_*** of forced events.

u8 QSPI_IRQIsForced(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check if IRQ interrupt is forced for current CPU. Returns bit mask with **QSPI_EVENT_*** of forced events.

u8 QSPI_IRQIsForcedDorm(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check if IRQ interrupt is forced for dormant wake. Returns bit mask with **QSPI_EVENT_*** of forced events.

u8 QSPI_IRQIsPendingCpu(int cpu, int pin);

cpu ... CPU core 0 or 1 (or 2 dormant), use CpuID() to get current core

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check IRQ interrupt status for selected CPU core. Returns bit mask with **QSPI_EVENT_*** of pending events.

u8 QSPI_IRQIsPending(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check IRQ interrupt status for current CPU core. Returns bit mask with **QSPI_EVENT_*** of pending events.

u8 QSPI_IRQIsPendingDorm(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Check IRQ interrupt status for dormant wake. Returns bit mask with **QSPI_EVENT_*** of pending events.

Pin output/input data (use pins QSPI_PIN_*)

Warning: pins and pads use different numbering (use predefined constants).

u8 QSPI_In(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Get input pin (returns 0 or 1).

u32 QSPI_InAll();

Get all input pins (bit 0..5 = pin 0..5).

void QSPI_Out0(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Output 0 (“LOW” state) to the pin.

void QSPI_Out1(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Output 1 (“HIGH” state) to the pin.

void QSPI_Flip(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Flip output to the pin.

void QSPI_Out(int pin, int val);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

val ... output value 0 or 1

Output value to the pin.

void QSPI_OutAll(u32 value);

value .. output value, bit 0..5 = pin 0..5

Output all pins.

void QSPI_OutMask(u32 mask, u32 value);

mask ... mask of pins to output (set bit to ‘1’ to output value to the pin)

val ... output values of the pins

Output masked values to the pins. To use pin mask in range (first..last), use function RangeMask().

void QSPI_ClrMask(u32 mask);

mask ... mask of pins to output 0 (set bit to ‘1’ to output value 0 to the pin)

Clear output pins masked. To use pin mask in range (first..last), use function RangeMask().

void QSPI_SetMask(u32 mask);

mask ... mask of pins to output 1 (set bit to '1' to output value 1 to the pin)

Set output pins masked. To use pin mask in range (first..last), use function RangeMask().

void QSPI_FlipMask(u32 mask);

mask ... mask of pins to flip output (set bit to '1' to flip output of the pin)

Flip output pins masked. To use pin mask in range (first..last), use function RangeMask().

u8 QSPI_GetOut(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Get last output value to pin (returns 0 or 1).

u8 QSPI_GetOutAll();

Get all last output values to pins.

void QSPI_DirOut(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Set output direction of pin (enable output mode).

void QSPI_DirIn(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Set input direction of pin (disable output mode).

void QSPI_DirOutMask(u32 mask);

mask ... mask of pins to set output direction (set bit to '1' to set output direction)

Set output direction of pin masked. To use pin mask in range (first..last), use function RangeMask().

void QSPI_DirInMask(u32 mask);

mask ... mask of pins to set input direction (set bit to '1' to set input direction)

Set input direction of pin masked. To use pin mask in range (first..last), use function RangeMask().

void QSPI_SetDirMask(u32 mask, u32 outval);

mask ... mask of pins to set direction (set bit to '1' to set direction of to the pin)

outval ... direction of the pins (1 = output)

Set direction masked.

void QSPI_SetDirAll(u32 outval);

outval ... direction of the pins (1 = output)

St direction of all pins.

u8 QSPI_GetDir(int pin);

pin ... pin number 0..5 or 0..7 QSPI_PIN_*

Get output direction of pin (returns 0=input or 1=output).

u8 QSPI_GetDirAll();

Get output direction of all pins (returns 0=input, 1=output).

void QSPI_InitFlash();

Initialize QSPI pins to connect Flash memory.

2.23. Reset - Power-On State Machine

Files: `sdk_reset.h`, `sdk_reset.c`

Config: -

The power-on state machine removes the reset from various hardware blocks in a specific order. There are register controls that can be used to override and see the status of the power-on state machine. This allows hardware blocks in the power-on state machine to be reset by software if necessary.

Power-on peripherals indices of RP2040

<code>POWER_PROC1</code>	16 processor core 1
<code>POWER_PROC0</code>	15 processor core 0
<code>POWER_SIO</code>	14
<code>POWER_VREG</code>	13 VREG and chip reset
<code>POWER_XIP</code>	12 flash memory interface
<code>POWER_SRAM5</code>	11
<code>POWER_SRAM4</code>	10
<code>POWER_SRAM3</code>	9
<code>POWER_SRAM2</code>	8
<code>POWER_SRAM1</code>	7
<code>POWER_SRAM0</code>	6
<code>POWER_ROM</code>	5
<code>POWER_BUSFABRIC</code>	4 bus fabric
<code>POWER_RESETS</code>	3
<code>POWER_CLOCKS</code>	2
<code>POWER_XOSC</code>	1
<code>POWER_ROSC</code>	0

Power-on peripherals indices of RP2350

<code>POWER_PROC1</code>	24 processor core 1
<code>POWER_PROC0</code>	23 processor core 0
<code>POWER_ACCESSCTRL</code>	22
<code>POWER_SIO</code>	21
<code>POWER_XIP</code>	20 flash memory interface
<code>POWER_SRAM9</code>	19
<code>POWER_SRAM8</code>	18
<code>POWER_SRAM7</code>	17

POWER_SRAM6	16
POWER_SRAM5	15
POWER_SRAM4	14
POWER_SRAM3	13
POWER_SRAM2	12
POWER_SRAM1	11
POWER_SRAM0	10
POWER_BOOTRAM	9
POWER_ROM	8
POWER_BUSFABRIC	7 bus fabric
POWER_PSM_READY	6
POWER_CLOCKS	5
POWER_RESETS	4
POWER_XOSC	3
POWER_ROSC	2
POWER OTP	1
POWER PROC COLD	0

Reset peripherals indices of RP2040

- use **BIT(RESET_*)** to use in the mask

RESET_USBCTRL	24
RESET_UART1	23
RESET_UART0	22
RESET_TIMER	21
RESET_TBMAN	20
RESET_SYSINFO	19
RESET_SYSCFG	18
RESET_SPI1	17
RESET_SPI0	16
RESET_RTC	15
RESET_PWM	14
RESET_PLL_USB	13
RESET_PLL_SYS	12
RESET_PIO1	11
RESET_PIO0	10
RESET_PADS_QSPI	9
RESET_PADS_BANK0	8

RESET_JTAG	7
RESET_IO_QSPI	6
RESET_IO_BANK0	5
RESET_I2C1	4
RESET_I2C0	3
RESET_DMA	2
RESET_BUSCTRL	1
RESET_ADC	0

Reset peripherals indices of RP2350

- use `BIT(RESET_*)` to use in the mask

RESET_USBCTRL	28
RESET_UART1	27
RESET_UART0	26
RESET_TRNG	25
RESET_TIMER1	24
RESET_TIMER0	23
RESET_TBMAN	22
RESET_SYSINFO	21
RESET_SYSCFG	20
RESET_SPI1	19
RESET_SPI0	18
RESET_SHA256	17
RESET_PWM	16
RESET_PLL_USB	15
RESET_PLL_SYS	14
RESET_PIO2	13
RESET_PIO1	12
RESET_PIO0	11
RESET_PADS_QSPI	10
RESET_PADS_BANK0	9
RESET_JTAG	8
RESET_IO_QSPI	7
RESET_IO_BANK0	6
RESET_I2C1	5
RESET_I2C0	4
RESET_HSTX	3

RESET_DMA	2
RESET_BUSCTRL	1
RESET_ADC	0

void ResetPeripheryOn(int peri);

peri ... perifery **RESET_***

Start resetting periphery.

void ResetPeripheryOnMask(u32 mask);

mask ... bit mask of peripheries (use **BIT(RESET_*)**)

Start resetting peripheries specified by the mask.

void ResetPeripheryOff(int peri);

peri ... perifery **RESET_***

Stop resetting periphery.

void ResetPeripheryOffMask(u32 mask);

mask ... bit mask of peripheries (use **BIT(RESET_*)**)

Stop resetting peripheries specified by the mask.

Bool ResetPeripheryDone(int peri);

peri ... perifery **RESET_***

Check if periphery is ready to be accessed after reset.

Bool ResetPeripheryDoneMask(u32 mask);

mask ... bit mask of peripheries (use **BIT(RESET_*)**)

Check if peripheries are ready to be accessed after reset.

void ResetPeripheryOffWait(int peri);

peri ... perifery **RESET_***

Stop resetting periphery and wait.

void ResetPeripheryOffWaitMask(u32 mask);

mask ... bit mask of peripheries (use **BIT(RESET_*)**)

Stop resetting peripheries specified by the mask and wait.

void WDPeripheryEnable(int peri);

peri ... perifery **RESET_***

Enable reset periphery by watchdog.

void WDPeripheryEnableMask(u32 mask);

mask ... bit mask of peripheries (use BIT(**RESET_***))

Enable reset peripheries by watchdog specified by the mask.

void WDPeripheryDisable(int peri);

peri ... perifery **RESET_***

Disable reset periphery by watchdog.

void WDPeripheryDisableMask(u32 mask);

mask ... bit mask of peripheries (use BIT(**RESET_***))

Disable reset periphery by watchdog specified by the mask.

void ResetPeriphery(int peri);

peri ... perifery **RESET_***

Hard reset periphery and wait to be accessed again.

void ResetPeripheryMask(u32 mask);

mask ... bit mask of peripheries (use BIT(**RESET_***))

Hard reset peripheries and wait to be accessed again.

2.24. ROSC - Ring Oscillator

Files: `sdk_rosc.h`, `sdk_rosc.c`

Config: `USE_ROSC` (default 1)

The Ring Oscillator (ROSC) is an on-chip oscillator built from a ring of inverters. It requires no external components and is started automatically during RP2040/RP2350 power up. It provides the clock to the cores during boot. During boot the ROSC runs at a nominal 6.5MHz and is guaranteed to be in the range 1.8MHz to 12MHz. As an additional function, the ROSC oscillator is used to initialize the random generator. The ROSC frequency can be programmatically set in 64 stages and depends on the number of selected inventor stages and the inventor current.

void RoscEnable();

Enable ring oscillator.

void RoscDisable();

Disable ring oscillator (cannot be disabled if CPU uses it as a clock source).

void RoscRandEnable();

Enable frequency randomisation (only RP2350).

void RoscRandDisable();

Disable frequency randomisation (only RP2350).

u8 RoscLevel;

Current ROSC frequency level set by the `RoscSetLevel()` function

void RoscSetLevel(int level);

level ... frequency level 0..63 (0 is default level)

Typical frequency ranges:

- level 0..24, low frequency range (8 stages), 81..150 MHz
- level 25..43, medium frequency range (6 stages), 101..181 MHz
- level 44..56, high frequency range (4 stages), 136..232 MHz
- level 57..63, too high frequency range (2 stages), 217..321 MHz

Set ring oscillator frequency level. The above frequencies are very approximate and may vary by tens of percent.

void RoscSetSeed(u32 seed);

seed ... seed value (default `ROSC_SEED_DEF` = 0x3F04B16D)

Set seed of ROSC randomiser (only RP2350).

void RoscSetDiv(int div);

div ... ROSC divider 1..32 (RP2040) or 1..128 (RP2350)

Set ROSC divider. Default frequency level 0 and default divider value 16 (RP2350: 8) give typical default frequency $81/16 = 5$ MHz (or 6 MHz according to the datasheet) or $81/8 = 10$ MHz.

u8 RoscGetDiv();

Get ROSC divider (RP2040: 1..32, RP2350: 1..128).

void RoscInit();

Initialize ROSC ring oscillator to its default state (typical frequency 6 MHz, or 11 MHz on RP2350). This function is called internally from the ClockInit() function during application startup.

void RoscSetFreq(int freq10);

freq10 ... required frequency in MHz * 10

Select ring oscillator frequency in range 2.5 .. 255 MHz (input value freq10 means required frequency in MHz * 10). Selected frequency is very approximate, it can vary by tens of percent. The main usage of the function is in conjunction with the XOSC oscillator so that frequency can be continuously changed during calibration.

void RoscDormant();

Start dormant mode of the ring oscillator.

void RoscWake();

Wake up ring oscillator from dormant mode.

Bool RoscIsStable();

Check if ring oscillator is running and stable.

void RoscWait();

Wait for stable state of the ring oscillator (ROSC must be enabled). Usually not needed, ring oscillator stabilizes almost immediately.

int RoscRandBit();

Get random bit from ring oscillator. This function reads random bit from ROSC oscillator. It has low randomness, it is used internally by following function.

u32 RoscRand32();

u32 RoscRand();

Get true random data from the ring oscillator, combined with LCG generator. It takes 26 us at 125 MHz, that means speed 1.2 Mbits/sec.

```
void RoscSetCount(u8 count);
```

count ... value to set

Set counter of ring oscillator.

```
u8 RoscGetCount();
```

Get counter of ring oscillator. After setting the value, counter runs down to zero and stops there.

2.25. RTC - Real-Time Clock

Files: `sdk_rtc.h`, `sdk_rtc.c`

Config: `USE_RTC (default 1) (only RP2040)`

The Real-time Clock (RTC) on RP2040 provides time in human-readable format and can be used to generate interrupts at specific times. Leap year does not work OK for years 1900 and 2100, so RTC with leap should be used in range 1901..2099.

On the RP2350 the RTC generator is replaced by the "Tick Generator".

typedef void (*pRtcAlarm());

Prototype of RTC alarm callback function.

pRtcAlarm RtcAlarm;

Variable with current RTC alarm callback function (NULL=none).

void Old2NewDateTime(const datetime_t* old_dt, sDateTime* new_dt);

old_dt ... datetime structure in old format

new_dt ... datetime structure in new format

Conversion from old `datetime_t` to new `sDateTime`.

void New2OldDateTime(const sDateTime* new_dt, datetime_t* old_dt);

new_dt ... datetime structure in new format

old_dt ... datetime structure in old format

Conversion from new `sDateTime` to old `datetime_t`.

Bool RtcRunning();

Check if RTC is running.

Bool RtcPending();

Check if RTC alarm is pending.

void RtcInit();

Initialize RTC. Not running yet - it will start after setting date and time `RtcSetDateTime()`.

void RtcSetDateTime(const sDateTime* dt);

void RtcSetOldDateTime(const datetime_t* dt);

dt ... datetime structure (see Calendar library)

Set RTC clock from datetime structure.

```
void RtcGetDateTime(sDateTime* dt);
void RtcGetOldDateTime(datetime_t* dt);
```

dt ... datetime structure (see Calendar library)

Get RTC to datetime structure.

```
void RtcAlarmEnable();
```

Enable RTC alarm.

```
void RtcAlarmDisable();
```

Disable RTC alarm.

```
Bool RtcDateTimeRepeated(const sDateTime* dt);
```

dt ... datetime structure (see Calendar library)

Check datetime structure, if alarm will be repeated (some entry has value = -1).

```
Bool RtcRepeated();
```

Check if RTC alarm is repeated.

```
void RtcAlarmStart(const sDateTime* dt, pRtcAlarm callback);
```

dt ... datetime structure (see Calendar library)

callback ... alarm callback function

Start alarm from datetime structure. Set entries to -1 to disable matching and enable repeating them. To deactivate, use RtcAlarmStop() function.

```
void RtcAlarmStop();
```

Stop alarm (deactivate callback).

```
void RtcAlarmForce();
```

Force alarm IRQ (must be unforced in IRQ handler).

```
void RtcAlarmUnforce();
```

Unforce alarm IRQ.

2.26. SHA-256 - SHA-256 accelerator

Files: `sdk_sha256.h`, `sdk_sha256.c`

Config: `USE_SHA256 (default 1) (only RP2350)`

SHA-256 on RP2350 provides hardware accelerated message 256-bit checksum. On RP2040 there is alternative slower software function. Bit stream message must be terminated with bit "1", zero bits and 64-bit block indicating length of message. Number of zero bits must ensure padding block to modulo 512 bits.

`void SHA256_Start();`

Start SHA-256 core for a new checksum. SUM registers are initialised and internal counters are cleared. Message must be written before initiating DMA transfer, core will request 16 transfers of 32-bit.

`Bool SHA256_Ready();`

Check if SHA-256 core is ready to accept mode data.

`Bool SHA256_Valid();`

Check if SHA-256 checksum in registers SUM is valid.

`Bool SHA256_ErrWrite();`

Check error - write when not ready.

`void SHA256_ErrWriteClr();`

Clear error of write when not ready.

`void SHA256_DataSize(int size);`

`size ... data size SHA256_*`

Set data size `SHA256_DATA_*`.

`SHA256_DATA_8 0 data size 8 bits`

`SHA256_DATA_16 1 data size 16 bits`

`SHA256_DATA_32 2 data size 32 bits`

`void SHA256_BswapEnable();`

Enable byte swapping of 32-bit values (convert CPU little-endian to SHA-256 big-endian; default state).

`void SHA256_BswapDisable();`

Disable byte swapping of 32-bit values (data are pre-formated in big-endian).

void SHA256_Write(u32 data);

data ... data to write

Write data - write 16 words of 32-bits.

volatile u32* SHA256_DataAddr();

Get address of the data register to write using DMA.

u32 SHA256_Sum(int inx);

inx ... index of u32 data entry 0..7

Get result data.

void SHA256_WaitReady();

Wait until hardware is ready to accept more data.

void SHA256_WaitValid();

Wait until checksum is valid.

void SHA256_GetResult(u32* dst, Bool bigend);

dst ... destination buffer (32 bytes)

bigend ... get result as big-endian

Read 256-bit result, 8 words of 32-bit. Result must be valid.

2.27. SPI - Serial Peripheral Interface

Files: `sdk_spi.h, sdk_spi.c`

Config: `USE_SPI (default 1)`

RP2040/RP2350 has two identical SPI (Serial Peripheral Interface) controllers, both based on an ARM Primecell Synchronous Serial Port (SSP). Each controller supports the following features:

- Master or Slave modes
- Motorola SPI-compatible interface
- Texas Instruments synchronous serial interface
- National Semiconductor Microwire interface
- 8 deep Tx and Rx FIFOs
- Interrupt generation to service FIFOs or indicate error conditions
- Can be driven from DMA
- Programmable clock rate
- Programmable data size 4-16 bits

Most functions have two forms. Functions can be addressed either by the SPI index (parameter "int spi") or by a pointer to the hardware interface ("spi_hw_t* hw", as returned from `SPI_GetHw()` function). Functions in the second case are marked with "`_hw`". Addressing by SPI index is easier to use, addressing by pointer can generate slightly better optimized code.

`spi_hw_t* SPI_GetHw(int spi);`

`spi` ... SPI index 0 or 1

Get SPI hardware interface from SPI index. Returns `spi0_hw` or `spi1_hw`.

`u8 SPI_GetInx(const spi_hw_t* hw);`

`hw` ... pointer to hardware interface (as returned from `SPI_GetHw()`)

Get SPI index from SPI hardware interface.

`u8 SPI_GetDreq(int spi, Bool tx);`

`u8 SPI_GetDreq_hw(const spi_hw_t* hw, Bool tx);`

`spi` ... SPI index 0 or 1

`hw` ... pointer to hardware interface (as returned from `SPI_GetHw()`)

`tx` ... True for sending data to SPI, False for receiving data from SPI

Get SPI DREQ index.

SPI setup

```
void SPI_Reset(int spi);
void SPI_Reset_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
SPI reset.

```
void SPI_Init(int spi, u32 baudrate);
void SPI_Init_hw(spi_hw_t* hw, u32 baudrate);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
baudrate ... baudrate speed

SPI initialize to Motorola 8-bit Master mode (polarity 0, phase 0).

```
void SPI_Term(int spi);
void SPI_Term_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
SPI terminate.

```
void SPI_Baudrate(int spi, u32 baudrate);
void SPI_Baudrate_hw(spi_hw_t* hw, u32 baudrate);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
baudrate ... baudrate speed

Set SPI baudrate. Check real result baudrate with SPI_GetBaudrate().

```
u32 SPI_GetBaudrate(int spi);
u32 SPI_GetBaudrate_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Get current SPI baudrate.

```
void SPI_DataSize(int spi, int size);
void SPI_DataSize_hw(spi_hw_t* hw, int size);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
size ... number of bits 4..16

Set SPI data size to 4..16 bits.

```
void SPI_Format(int spi, int form);
void SPI_Format_hw(spi_hw_t* hw, int form);

    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
    form ... frame format SPI_FORMAT_*
```

Set SPI frame format to **SPI_FORMAT_***.

Frame format:

SPI_FORMAT_SPI Motorola SPI (default format)

SPI_FORMAT_SS TI synchronous serial

SPI_FORMAT_MW National Microwire

```
void SPI_Pol(int spi, int pol);
void SPI_Pol_hw(spi_hw_t* hw, int pol);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

pol ... polarity 0=LOW, 1=HIGH

Set SPI clock polarity to steady state (only Motorola SPI frame format).

```
void SPI_Phase(int spi, int phase);
void SPI_Phase_hw(spi_hw_t* hw, u8 phase);

    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
    phase ... phase on 0=first, 1=second clock edge
```

Set SPI clock phase to data captured (only Motorola SPI frame format).

```
void SPI_LoopBackOn(int spi);
void SPI_LoopBackOn_hw(spi_hw_t* hw);

    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
```

Set loop back mode ON.

```
void SPI_LoopBackOff(int spi);
void SPI_LoopBackOff_hw(spi_hw_t* hw);

    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
```

Set loop back mode OFF (default state).

```
void SPI_Enable(int spi);
void SPI_Enable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Enable SPI.

```
void SPI_Disable(int spi);
void SPI_Disable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Disable SPI.

```
void SPI_Master(int spi);
void SPI_Master_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Set SPI master mode (default state).

```
void SPI_Slave(int spi);
void SPI_Slave_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Set SPI slave mode.

```
void SPI_OutEnable(int spi);
void SPI_OutEnable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Enable SPI output if in slave mode (default state).

```
void SPI_OutDisable(int spi);
void SPI_OutDisable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())

Disable SPI output if in slave mode.

SPI status

Bool SPI_TxIsEmpty(int spi);
Bool SPI_TxIsEmpty_hw(const spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Check if transmit FIFO is empty (can send data).

Bool SPI_TxIsFull(int spi);
Bool SPI_TxIsFull_hw(const spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Check if transmit FIFO is full (cannot send data).

Bool SPI_RxIsEmpty(int spi);
Bool SPI_RxIsEmpty_hw(const spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Check if receive FIFO is empty (cannot receive data).

Bool SPI_RxIsFull(int spi);
Bool SPI_RxIsFull_hw(const spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Check if receive FIFO is full (receiver can overflow).

Bool SPI_IsBusy(int spi);
Bool SPI_IsBusy_hw(const spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Check if SPI is busy (currently transmitting/receiving frame).

SPI interrupt

void SPI_RxOverEnable(int spi);
void SPI_RxOverEnable_hw(spi_hw_t* hw);

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

Enable interrupt on receive overrun.

```
void SPI_RxOverDisable(int spi);
void SPI_RxOverDisable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable interrupt on receive overrun.
```

```
void SPI_RxOverClear(int spi);
void SPI_RxOverClear_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Clear interrupt on receive overrun status.
```

```
u8 SPI_GetRxOverRaw(int spi);
u8 SPI_GetRxOverRaw_hw(const spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get raw status of interrupt on receive overrun (returns 0 or 1).
```

```
u8 SPI_GetRxOver(int spi);
u8 SPI_GetRxOver_hw(const spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get masked status of interrupt on receive overrun (returns 0 or 1).
```

```
void SPI_RxToutEnable(int spi);
void SPI_RxToutEnable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Enable interrupt on receive timeout.
```

```
void SPI_RxToutDisable(int spi);
void SPI_RxToutDisable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable interrupt on receive timeout.
```

```
void SPI_RxToutClear(int spi);
void SPI_RxToutClear_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Clear interrupt on receive timeout status.

```
u8 SPI_GetRxToutRaw(int spi);
u8 SPI_GetRxToutRaw_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get raw status of interrupt on receive timeout (returns 0 or 1).

```
u8 SPI_GetRxTout(int spi);
u8 SPI_GetRxTout_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get masked status of interrupt on receive timeout (returns 0 or 1).

```
void SPI_RxFifoEnable(int spi);
void SPI_RxFifoEnable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Enable interrupt on receive half FIFO.

```
void SPI_RxFifoDisable(int spi);
void SPI_RxFifoDisable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable interrupt on receive half FIFO.

```
u8 SPI_GetRxFifoRaw(int spi);
u8 SPI_GetRxFifoRaw_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get raw status of interrupt on receive half FIFO (returns 0 or 1).

```
u8 SPI_GetRxFifo(int spi);
u8 SPI_GetRxFifo_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get masked status of interrupt on receive half FIFO (returns 0 or 1).

```
void SPI_TxFifoEnable(int spi);
void SPI_TxFifoEnable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Enable interrupt on send half FIFO.

```
void SPI_TxFifoDisable(int spi);
void SPI_TxFifoDisable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable interrupt on send half FIFO.

```
u8 SPI_GetTxFifoRaw(int spi);
u8 SPI_GetTxFifoRaw_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get raw status of interrupt on send half FIFO (returns 0 or 1).

```
u8 SPI_GetTxFifo(int spi);
u8 SPI_GetTxFifo_hw(const spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Get masked status of interrupt on send half FIFO (returns 0 or 1).

SPI DMA

```
void SPI_TxDmaEnable(int spi);
void SPI_TxDmaEnable_hw(spi_hw_t* hw);
```

spi ... spi 0 or 1
hw ... pointer to hardware interface (as returned from SPI_GetHw())
Enable transmit DMA.

```
void SPI_TxDmaDisable(int spi);
void SPI_TxDmaDisable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable transmit DMA.
```

```
void SPI_RxDmaEnable(int spi);
void SPI_RxDmaEnable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Enable receive DMA.
```

```
void SPI_RxDmaDisable(int spi);
void SPI_RxDmaDisable_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Disable receive DMA.
```

SPI data transfer

```
void SPI_Write(int spi, u16 data);
void SPI_Write_hw(spi_hw_t* hw, u16 data);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
    data ... data to send
Send data to transmit FIFO (does not check status register).
```

```
u16 SPI_Read(int spi);
u16 SPI_Read_hw(const spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Receive data from receive FIFO (does not check status register).
```

```
void SPI_RxFlush(int spi);
void SPI_RxFlush_hw(spi_hw_t* hw);
    spi ... spi 0 or 1
    hw ... pointer to hardware interface (as returned from SPI_GetHw())
Flush receive FIFO.
```

```
void SPI_Send8Recv(int spi, const u8* src, u8* dst, int len);
void SPI_Send8Recv_hw(spi_hw_t* hw, const u8* src, u8* dst, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

src ... pointer to source buffer with data to send

dst ... pointer to destination buffer to data to receive

len ... length of data to send and receive (number of bytes)

Send and receive 8-bit data simultaneously.

```
void SPI_Send8(int spi, const u8* src, int len);
void SPI_Send8_hw(spi_hw_t* hw, const u8* src, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

src ... pointer to source buffer with data to send

len ... length of data to send (number of bytes)

Send 8-bit data, discard any data received back.

```
void SPI_Send8DMA(int spi, int dma, const u8* src, int len);
void SPI_Send8DMA_hw(spi_hw_t* hw, int dma, const u8* src, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

dma ... DMA channel 0..11

src ... pointer to source buffer with data to send

len ... length of data to send (number of bytes)

Send 8-bit data using DMA.

```
void SPI_Recv8(int spi, u8 repeated_tx, u8* dst, int len);
void SPI_Recv8_hw(spi_hw_t* hw, u8 repeated_tx, u8* dst, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

repeated_tx ... byte to send repeated

dst ... pointer to destination buffer to data to receive

len ... length of data to receive (number of bytes)

Receive 8-bit data, send repeated byte (usually 0).

```
void SPI_Send16Recv(int spi, const u16* src, u16* dst, int len);
void SPI_Send16Recv_hw(spi_hw_t* hw, const u16* src, u16* dst, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

src ... pointer to source buffer with data to send

dst ... pointer to destination buffer to data to receive

len ... length of data to send and receive (number of u16 entries)

Send and receive 16-bit data simultaneously.

```
void SPI_Send16(int spi, const u16* src, int len);
void SPI_Send16_hw(spi_hw_t* hw, const u16* src, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

src ... pointer to source buffer with data to send

len ... length of data to send (number of u16 entries)

Send 16-bit data, discard any data received back.

```
void SPI_Recv16(int spi, u16 repeated_tx, u16* dst, int len);
void SPI_Recv16_hw(spi_hw_t* hw, u16 repeated_tx, u16* dst, int len);
```

spi ... spi 0 or 1

hw ... pointer to hardware interface (as returned from SPI_GetHw())

repeated_tx ... word to send repeated

dst ... pointer to destination buffer to data to receive

len ... length of data to receive (number of u16 entries)

Receive 16-bit data, send repeated word (usually 0).

2.28. SpinLock

Files: `sdk_spinlock.h`, `sdk_spinlock.c`

Config: `USE_SPINLOCK` (default 1)

The SIO provides 32 hardware spinlocks, which can be used to manage mutually-exclusive access to shared software resources. Each spinlock is a one-bit flag, mapped to a different address. Generally software will acquire a lock by repeatedly polling the lock bit ("spinning" on the lock) until it is successfully claimed. This is inefficient if the lock is held for long periods, so generally the spinlocks should be used to protect the short critical sections of higher-level primitives such as mutexes, semaphores and queues.

SpinLocks are used to lock resources between processor cores. SpinLock cannot be used to lock sharing between an interrupt and the main program. You must use an interrupt disabled to lock the resource against interrupt service.

It is recommended to schedule the use of each spinlock in the program configuration using `#define`. In addition, it is possible to use automatic allocation of spinlocks using the "claim" functions. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

Most functions have two forms. Functions can be addressed either by the spinlock index (parameter "int lock") or by a pointer to the hardware register ("spin_lock_t* hw", as returned from `SpinGetHw()` function). Functions in the second case are marked with "_hw". Addressing by spinlock index is easier to use, addressing by pointer can generate slightly better optimized code.

`spin_lock_t* SpinGetHw(int lock);`

lock ... lock index 0..31

Get spinlock hardware register from spinlock index.

`u8 SpinGetInx(const spin_lock_t* hw);`

hw ... pointer to spinlock hardware registers (as returned by `SpinGetHw()`)

Get spinlock index from spinlock hardware register.

Claim spinlock

Functions are not atomic safe - not recommended to be used in both cores or in IRQ at the same time. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

`u8 SpinNextStriped();`

Get next striped spinlock index in range `PICO_SPINLOCK_ID_STRIPED_FIRST` (default 16 in `config_def.h`) to `PICO_SPINLOCK_ID_STRIPED_LAST` (default 21 in `config_def.h`).

void SpinClaim(int lock);

lock ... lock index 0..31

Claim spinlock (mark it as used).

void SpinClaimMask(u32 mask);

mask ... mask of locks (bit 0..31 = lock index 0..31)

Claim spinlocks with mask (mark them as used).

void SpinUnclaim(int lock);

lock ... lock index 0..31

Unclaim spinlock (mark it as not used).

void SpinUnclaimMask(u32 mask);

mask ... mask of locks (bit 0..31 = lock index 0..31)

Unclaim spinlocks with mask (mark them as not used).

Bool SpinIsClaimed(int lock);

lock ... lock index 0..31

Check if spinlock is claimed.

s8 SpinClaimFree();

Claim free spinlock in range PICO_SPINLOCK_ID CLAIM_FREE_FIRST (default 22 in config_def.h) to PICO_SPINLOCK_ID CLAIM_FREE_LAST (default 27 in config_def.h). Returns -1 if no free spinlock.

Lock spinlock

Bool SpinTryLock(int lock);

Bool SpinTryLock_hw(spin_lock_t* hw);

lock ... lock index 0..31

hw ... pointer to spinlock hardware registers (as returned by SpinGetHw())

Try to lock spinlock - returns False if spinlock was not locked.

void SpinLock(int lock);

void SpinLock_hw(spin_lock_t* hw);

lock ... lock index 0..31

hw ... pointer to spinlock hardware registers (as returned by SpinGetHw())

Lock spinlock and wait if already locked.

Bool SpinLockTimeout(int lock, u32 us);
Bool SpinLockTimeout_hw(spin_lock_t* hw, u32 us);

lock ... lock index 0..31

hw ... pointer to spinlock hardware registers (as returned by SpinGetHw())

us ... timeout in [us]

Lock spinlock with timeout in [us]. Returns False on timeout.

void SpinUnlock(int lock);
void SpinUnlock_hw(spin_lock_t* hw);

lock ... lock index 0..31

hw ... pointer to spinlock hardware registers (as returned by SpinGetHw())

Unlock spinlock (default state).

Bool SpinIsLocked(int lock);

lock ... lock index 0..31

Check if spinlock is locked.

void SpinInit(int lock);
void SpinInit_hw(spin_lock_t* hw);

lock ... lock index 0..31

hw ... pointer to spinlock hardware registers (as returned by SpinGetHw())

Initialize spinlock (all spinlocks are at unlocked state after reset, no need to initialize).

void SpinResetAll();

Reset (initialize) all spinlocks. This function is automatically called during application startup from the internal RuntimeInit() function.

u32 SpinGetAll();

Get state of all spinlocks.

SPINLOCK_LOCK(lock);

lock ... lock index 0..31

Macro to lock spinlock and IRQ on start of function (to share resources with interrupt).

SPINLOCK_UNLOCK(lock);

lock ... lock index 0..31

Macro to unlock spinlock and IRQ on end of function.

2.29. SSI - synchronous serial interface

Files: `sdk_ssi.h`, `sdk_ssi.c`

Config: - (only RP2040)

SSI (synchronous serial interface) is used to communicate with external Flash. RP2350 uses QMI interface.

```
#define FLASHQSPI_CLKDIV_DEF      4      // default clkdiv
```

```
int SSI_FlashClkDiv();
```

Get flash speed - get clock divider.

```
void SSI_SetFlashClkDiv(int clkdiv);
```

Set flash speed - set clock divider (default 4).

```
void SSI_FlashQspi(int clkdiv);
```

clkdiv ... clock divider (must be even number)

Set flash to fast QSPI mode. Default clkdiv = `FLASHQSPI_CLKDIV_DEF` (= 4). Supported devices: Winbond W25Q080, W25Q16JV, AT25SF081, S25FL132K0. Raspberry Pico contains W25Q16JVUXIQ.

```
void SSI_InitFlash(int clkdiv);
```

clkdiv ... clock divider (must be even number)

Initialize Flash interface. It can be used when running a program from RAM when Flash memory is not initialized. Default clkdiv = `FLASHQSPI_CLKDIV_DEF` (= 4). Supported devices: Winbond W25Q080, W25Q16JV, AT25SF081, S25FL132K0. Raspberry Pico contains W25Q16JVUXIQ.

2.30. SysTick - SysTick System Timer

Files: `sdk_systick.h`, `sdk_systick.c`

Config: `USE_SYSTICK` (default 1)

SysTick is 24-bit counter of ARM CPU core that can be used as system timer. Each core has its own systick timer. SysTick in RP2040/RP2350 uses 1 us ticks as clock source. This is generated in the watchdog block as `timer_tick`.

SysTick is used in the PicoLibSDK library to measure the system time, i.e. the time elapsed since system startup. Simultaneously with the system time measurement, the current time expressed in Unix format is updated. In conjunction with a precise microsecond time counter, it is possible to set and read the current time and date with microsecond precision.

RP2350 in RISC-V mode does not support SysTick timer. Instead of it, the machine-mode timer is used in the same manner in following functions.

PicoLibSDK library runs SysTick timer with default period 5 ms. This can be changed in the `sdk_systick.h` file by editing the following values:

`#define SYSTICK_TICKS 5000`

- *SysTick period (number of 1 us ticks to interrupt every SYSTICK_MS ms)*

`#define SYSTICK_MS 5`

- *increment of system time on SysTick interrupt*

The current system time and the current Unix time can be read from the following variables:

`extern volatile u32 SysTime;`

- *system time counter, counts time from system start - incremented every SYSTICK_MS ms with overflow after 49 days (use difference, not absolute value!)*

`extern volatile u32 UnixTime; // current date and time in Unix format`

- *current date and time (incremented every SYSTICK_MS ms from CPU core 0)*

= number of seconds since 1/1/1970 (thursday) up to 12/31/2099

`extern volatile s16 TimeMs; // current time in [ms] 0..999`

void SysTickInit();

Initialize SysTick timer for this CPU core to interrupt every SYSTICK_MS ms. `AlarmInit()` function must be called before initializing SysTick timers. This function must be called from each CPU core separately. SysTick have common handler for both CPU cores, but only core 0 count system time. This function is automatically called during application startup from the internal `RuntimelInit()` function.

void SysTickTerm();

Terminate SysTick timer.

u32 GetUnixTime(s16* ms, s16* us);

ms ... pointer to get milliseconds of the time (NULL = not required)

us ... pointer to get microseconds of the time (NULL = not required)

Get current date and time in Unix format with [ms] and [us], synchronized.

void SetUnixTime(u32 time, s16 ms);

time ... time in Unix format

ms ... milliseconds 0..999

Set current date and time in Unix format with milliseconds. Should be called from CPU core 0 (from core 1, the setting may be inaccurate). The microsecond part of the time is not affected by the setting.

2.31. Ticks - Tick Generator

Files: `sdk_ticks.h`, `sdk_ticks.c`

Config: - (only RP2350)

The tick generator of the RP2350 distributes clock source to the individual clock generators. The generators should be configured to 1 us.

Tick generator index:

<code>TICK_PROC0</code>	0	clock source of SysTick of CPU core 0
<code>TICK_PROC1</code>	1	clock source of SysTick of CPU core 1
<code>TICK_TIMER0</code>	2	clock source of Timer0
<code>TICK_TIMER1</code>	3	clock source of Timer1
<code>TICK_WATCHDOG</code>	4	clock source of WatchDog
<code>TICK_RISCV</code>	5	clock source for RISC-V core

void TickStart(int tick, u32 cycles);

`tick` ... index of tick generator `TICK_*`

`cycles` ... clock division to get 1 us (source is `CLK_REF`)

Start one tick generator.

void TickStop(int tick);

`tick` ... index of tick generator `TICK_*`

stop one tick generator.

Bool TickIsRunning(int tick);

`tick` ... index of tick generator `TICK_*`

check if tick generator is running.

void TickStartAll(u32 cycles);

`cycles` ... frequency of `CLK_REF` in MHz

Start all tick generators.

2.32. Timer - Precise Timer with Alarm

Files: `sdk_timer.h`, `sdk_timer.c`

Config: `USE_TIMER` (default 1)

RP2040 has 64-bit counter, incremented once per microsecond, that can be used as precise timebase for the system. This counter can be read from a pair of latching registers, for race-free reads over a 32-bit bus. It has four alarms - match on the lower 32 bits of counter can trigger IRQ.

RP2350 has two 64-bit counters. Their source can be 1 us from Ticks generator, or system clock `CLK_SYS`. In this library the first timer is controlled by 1-us ticks by default, similar to RP2040. The second timer is by default controlled by the `CLK_SYS` clock and can be used for more accurate time measurement.

In addition to timers, the RP2350 also features 64-bit machine-mode timer for the RISC-V platform. During RISC-V mode, the machine-mode timer is used instead of the SysTick timer from ARM. In ARM mode, the machine-mode timer can be used as a regular timer.

There are 4 timer alarms. It is recommended to schedule the use of each alarm in the program configuration using `#define`. In addition, it is possible to use automatic allocation of alarms using the "claim" functions. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

[ALARM_IRQ\(alarm\)](#)

`alarm ... alarm 0..3`

Get IRQ number `IRQ_TIMER_0..IRQ_TIMER_3` for alarm 0..3.

Get time

[u32 Time\(\);](#)

Get 32-bit absolute system time LOW in [us]. The 32-bit time value is fully sufficient to measure most time intervals. The important thing is not to compare the time to a larger or smaller value, but to calculate interval from the previous value (with limitation to 32 bits) and then compare result 32-bit value.

[u16 TimeMs\(\);](#)

Get 16-bit absolute system time in [ms]. Not accurate, time in [us] is divided by 1024 - this allows the measurement of intervals using the difference of values.

[u64 Time64Fast\(\);](#)

Get 64-bit absolute system time in [us] - fast latched method, but it is not atomic safe. Do not use it simultaneously from both processor cores and from interrupts.

[u64 Time64\(\);](#)

Get 64-bit absolute system time in [us] - atomic method, concurrently safe.

void WaitUs(u32 us);

us ... required delay in microseconds

Wait for delay in [us] (max. 71 minutes, 4'294'967'295 us).

void WaitMs(int ms);

ms ... required delay in milliseconds

Wait for delay in [ms] (max. 71 minutes, 4'294'967 ms).

Claim alarm

Functions are not atomic safe - not recommended to be used in both cores or in IRQ at the same time. Claim functions do not affect the operation of other functions, it is up to the programmer's choice whether to use them or not.

void AlarmClaim(int alarm);

alarm ... alarm 0..3

Claim alarm (mark it as used).

void AlarmUnclaim(int alarm);

alarm ... alarm 0..3

Unclaim alarm (mark it as not used).

Bool AlarmIsClaimed(int alarm);

alarm ... alarm 0..3

Check if alarm is claimed.

s8 AlarmClaimFree(void);

Claim free alarm. Returns -1 on error.

Alarm interrupt

void AlarmStart(int alarm, irq_handler_t handler, u32 time);

alarm ... alarm 0..3

handler ... interrupt handler

time time interval in [us] after which to activate first alarm (max 71 minutes)

Start alarm. Alarm will be activated when lower 32 bits of the time counter match (interval max. 71 minutes). To set interrupt handler, vector table must be located in RAM. If vector table is not in RAM, use services with names isr_timer_0..isr_timer_3. Call AlarmAck() on start of interrupt, then AlarmRestart() to restart again, or AlarmStop() to deactivate. Alarm handler is shared between both processor cores. Only alarms of different numbers can be independent. Time interval 0 can be used to activate first interrupt immediately.

void AlarmForce(int alarm);

alarm ... alarm 0..3

Force alarm.

void AlarmUnforce(int alarm);

alarm ... alarm 0..3

Unforce alarm.

Bool AlarmIsForced(int alarm);

alarm ... alarm 0..3

Check if alarm is forced.

void AlarmAck(int alarm);

alarm ... alarm 0..3

Acknowledge alarm 0..3 interrupt - should be called on start of interrupt handler. Alarm will be disarmed automatically when it is triggered.

Bool AlarmIsPending(int alarm);

alarm ... alarm 0..3

Check alarm pending status.

void AlarmRestart(int alarm, u32 time);

alarm ... alarm 0..3

time ... time interval in [us] after which to activate next alarm (max 71 minutes)

Restart alarm - can be called from an interrupt for a repeated alarm. The maximum achievable interrupt frequency is about 100 kHz.

void AlarmStop(int alarm);

alarm ... alarm 0..3

Stop alarm - can be called from an interrupt if no next interrupt is required.

System Clock Timer

The following description refers to the second RP2350 timer, which is controlled by the CLK_SYS system clock by default. It can be used for more accurate time measurement.

[ALARM2_IRQ\(alarm\)](#)

alarm ... alarm 0..3

Get IRQ number [IRQ_TIMER1_0..IRQ_TIMER1_3](#) for alarm 0..3.

[void Timer2Init\(\);](#)

Initialize system clock timer. This function sets the [CLK_SYS](#) clock source and is called automatically from the [RuntimeInit\(\)](#) function when the application starts.

[u32 TimeSysClk\(\);](#)

Get 32-bit absolute system time LOW in sys_clk ticks.

[u64 TimeSysClk64Fast\(\);](#)

Get 64-bit absolute system time in sys_clk ticks. This is fast method, but it is not atomic safe (do not use simultaneously from both processor cores and from interrupts).

[u64 TimeSysClk64\(\);](#)

Get 64-bit absolute system time in sys_clk ticks - atomic method (concurrently safe).

[void WaitSysClk\(u32 sysclk\);](#)

sysclk ... delay in CLK_SYS ticks

Wait for delay in sys_clk ticks.

[void Alarm2Claim\(int alarm\);](#)

alarm ... alarm 0..3

Claim alarm (mark it as used).

[void Alarm2Unclaim\(int alarm\);](#)

alarm ... alarm 0..3

Unclaim alarm (mark it as not used).

[Bool Alarm2IsClaimed\(int alarm\);](#)

alarm ... alarm 0..3

Check if alarm is claimed.

s8 Alarm2ClaimFree();

Claim free alarm (returns -1 on error).

void Alarm2Start(int alarm, irq_handler_t handler, u32 time);

alarm ... alarm number 0..3

handler ... interrupt handler

time ... time interval in sys_clk after which to activate first alarm (min. 50)

Start alarm. Vector table must be located in RAM. If vector table is not in RAM, use services with names isr_timer1_0..isr_timer1_3. Call Alarm2Ack on start of interrupt, then Alarm2Restart to restart again, or Alarm2Stop to deactivate. Alarm handler is shared between both processor cores. Only alarms of different numbers can be independent.

void Alarm2Force(int alarm);

alarm ... alarm number 0..3

Force alarm.

void Alarm2Unforce(int alarm);

alarm ... alarm number 0..3

Unforce alarm.

Bool Alarm2IsForced(int alarm);

alarm ... alarm number 0..3

Check if alarm is forced.

void Alarm2Ack(int alarm);

alarm ... alarm number 0..3

Acknowledge alarm 0..3 interrupt - should be called at start of interrupt handler. Alarm will be disarmed automatically when it is triggered.

Bool Alarm2IsPending(int alarm);

alarm ... alarm number 0..3

Check alarm pending status.

void Alarm2Restart(int alarm, u32 time);

alarm ... alarm number 0..3

time ... time interval in sys_clk after which to activate next alarm

Restart alarm. Can be called from an interrupt for a repeated alarm. The maximum achievable interrupt frequency is about 100 kHz.

void Alarm2Stop(int alarm);

alarm ... alarm number 0..3

Stop alarm - can be called from an interrupt if no next interrupt is required.

Machine-mode timer of RP2350 RISC-V

In addition to timers, the RP2350 also features 64-bit machine-mode timer for the RISC-V platform. During RISC-V mode, the machine-mode timer is used instead of the SysTick timer from ARM. In ARM mode, the machine-mode timer can be used as a regular timer.

void MTimerDisable();

Disable RISC-V machine-mode timer.

void MTimerEnable();

Enable RISC-V machine-mode timer (default).

void MTimerSlow();

Set slow mode of RISC-V machine-mode timer - run from 1-us ticks (default).

void MTimerFast();

Set fast mode of RISC-V machine-mode timer - run directly from sys_clk.

u32 MTime();

Get 32-bit absolute time LOW from RISC-V machine-mode timer

u64 MTime64Fast();

Get 64-bit time from RISC-V machine-mode timer - fast method, but it is not atomic safe. Do not use simultaneously from both processor cores and from interrupts.

u64 MTime64();

Get 64-bit time from RISC-V machine-mode timer - atomic method (concurrently safe).

void MTimeSet(u64 time);

Set time to RISC-V machine-mode timer.

u64 MTimeGetCmp();

Get current compare value of RISC-V machine-mode timer.

void MTimeCmp(u64 timecmp);

timecmp ... compare value

Set compare value of RISC-V machine-mode timer. Each core has its own copy of compare registers. Interrupt is asserted whenever the 64-bit mtime value is greater than or equal to compare value. Raises IRQ_SIO_MTIMECMP interrupt on ARM, and mip.mtip interrupt on RISC-V.

2.33. TMDS - TMDS Encoder

Files: `sdk_tmds.h`, `sdk_tmds.c`

Config: `USE_TMDS (default 1) (only RP2350)`

RP2350 contains hardware accelerated encoder of TMDS modulation, used in DVI (HDMI) video output.

Pixel shift width

<code>TMDS_SHIFT_NO</code>	0	// do not shift colour data register
<code>TMDS_SHIFT_1</code>	1	// shift color data register by 1 bit (GRAY2)
<code>TMDS_SHIFT_2</code>	2	// shift color data register by 2 bits (GRAY4)
<code>TMDS_SHIFT_4</code>	3	// shift color data register by 4 bits (RGB121, GRAY16)
<code>TMDS_SHIFT_8</code>	4	// shift color data register by 8 bits (RGB332)
<code>TMDS_SHIFT_16</code>	5	// shift color data register by 16 bits (RGB565)

`void TMDS_ClearDC();`

Clear DC balance.

`void TMDS_Pix2NoShift();`

Enable pixel doubling - disable shift of 2nd encoder.

`void TMDS_Pix2Shift();`

Disable pixel doubling - enable shift of 2nd encoder.

`void TMDS_PixShift(int shift);`

`shift` ... pixel shift `TMDS_SHIFT_*`

Set pixel shift width.

`void TMDS_InterEnable();`

Enable interleaving. Symbols are packed into 5 chunks of 2 lanes times 2 bits, 30 bits total, each chunk 2 bits per lane.

`void TMDS_InterDisable();`

Disable interleaving. Each of 3 symbols appears as 10-bit field.

`void TMDS_Lane0Rot(int rot);`

`rot` ... 0..15

Set rotate of lane 0, blue. Right-rotate 16 LSBs of the colour accumulator by 0-15 bits.

void TMDS_Lane1Rot(int rot);

rot ... 0..15

Set rotate of lane 1, green. Right-rotate 16 LSBs of the colour accumulator by 0-15 bits.

void TMDS_Lane2Rot(int rot);

rot ... 0..15

Set rotate of lane 2, red. Right-rotate 16 LSBs of the colour accumulator by 0-15 bits.

void TMDS_Lane0Bits(int bits);

bits ... 1..8

Set bits of lane 0, blue. Set number of valid colour MSBs for lane 0. Use value 1 to 8.

void TMDS_Lane1Bits(int bits);

bits ... 1..8

Set bits of lane 1, green. Set number of valid colour MSBs for lane 1. Use value 1 to 8.

void TMDS_Lane2Bits(int bits);

bits ... 1..8

Set bits of lane 2, red. Set number of valid colour MSBs for lane 2. Use value 1 to 8.

void TMDS_Write(u32 data);

data ... data to write

Write data to colour register.

u32 TMDS_Peek();

Peek encoded data as single value 3x10 bits, no shift colour register.

u32 TMDS_Pop();

Pop encoded data as single value 3x10 bits, shift colour register.

u32 TMDS_Peek0();

Peek encoded data of lane 0 (blue) as two 10-bit symbols. No shift colour register.

u32 TMDS_Pop0();

Pop encoded data of lane 0 (blue) as two 10-bit symbols. Shift colour register.

u32 TMDS_Peek1();

Peek encoded data of lane 1 (green) as two 10-bit symbols. No shift colour register.

u32 TMDS_Pop1();

Pop encoded data of lane 1 (green) as two 10-bit symbols. Shift colour register.

u32 TMDS_Peek2();

Peek encoded data of lane 2 (red) as two 10-bit symbols. No shift colour register.

u32 TMDS_Pop2();

Pop encoded data of lane 2 (red) as two 10-bit symbols. Shift colour register.

2.34. TRNG - True Random Number Generator

Files: `sdk_trng.h, sdk_trng.c`

Config: `USE_TRNG (default 1) (only RP2350)`

RP2350 contains hardware true random number generator, suitable for data encryption.
Generation speed is 7.5 kb/s on 150 MHz.

Note: Suitable alternative is function RoscRand32(), also available on RP2040.

`void TRNG_VnErrEnable();`

Enable interrupt on von Neumann error (= 32 consecutive collected bits are identical).

`void TRNG_VnErrDisable();`

Disable interrupt on von Neumann error.

`void TRNG_VnErrClr();`

Clear von Neumann error.

`Bool TRNG_VnErr();`

Check von Neumann error.

`void TRNG_CrngtErrEnable();`

Enable interrupt on CRNGT error (= two consecutive blocks of 16 collected bits are equal).

`void TRNG_CrngtErrDisable();`

Disable interrupt on CRNGT error.

`void TRNG_CrngtClr();`

Clear CRNGT error.

`Bool TRNG_CrngtErr();`

Check CRNGT error.

`void TRNG_AutocorErrEnable();`

Enable interrupt on Autocorrelation error (= autocorrelation test failed four times in a row).

`void TRNG_AutocorErrDisable();`

Disable interrupt on Autocorrelation error.

`Bool TRNG_AutocorErr();`

Check Autocorrelation error. Cannot be cleared by software.

void TRNG_EhrValidEnable();

Enable interrupt on 192-bits collected ready.

void TRNG_EhrValidDisable();

Disable interrupt on 192-bits collected ready (default state).

void TRNG_EhrValidClr();

Clear 192-bits collected ready flag.

Bool TRNG_EhrValid();

Check if 192-bits collected ready.

void TRNG_SrcSel(int sel);

sel ... 0..3

Select number of inverters of the ROSC oscillator. 0..3, higher values select longer inverter chain lengths.

Bool TRNG_Valid();

Check if collection of bits are completed.

u32 TRNG_Data(int inx);

inx ... index of data 0..5

Get result data.

void TRNG_SrcEnable();

Enable RNG source.

void TRNG_SrcDisable();

Disable RNG source.

void TRNG_SampleCnt(int cnt);

cnt ... count clocks (min. 17)

Set count clocks between sampling of random bits.

int TRNG_AutocorrTrys();

Get count of autorocelation test starts.

void TRNG_AutocorrTrysClr();

Reset count of autorocelation test starts.

```
void TRNG_SwReset();
```

Internal reset TRNG generator.

```
Bool TRNG_Busy();
```

RNG is busy.

```
void TRNG_BitsRst();
```

Reset counter of collected bits.

```
int TRNG_BistCnt(int bist);
```

bist ... 0..2

Get RNG BIST counter.

2.35. UART - Asynchronous Serial Port

Files: `sdk_uart.h`, `sdk_uart.c`

Config: `USE_UART` (default 1)

RP2040 has 2 identical instances of a UART peripheral (Universal Asynchronous Receiver-Transmitter), based on the ARM Primecell UART (PL011). Each instance supports the following features:

- Separate 32×8 Tx and 32×12 Rx FIFOs
- Programmable baud rate generator, clocked by `clk_peri`
- Standard asynchronous communication bits (start, stop, parity)
- line break detection
- programmable serial interface (5, 6, 7, or 8 bits)
- 1 or 2 stop bits
- programmable hardware flow control

The resulting baudrate = $4 * \text{clk_peri} / (64*\text{ibrd} + \text{fbrd})$, where `ibrd` is the integer part of the divisor, `fbrd` is the fractional part of the divisor.

Most functions have two forms. Functions can be addressed either by the UART index (parameter "int uart") or by a pointer to the hardware interface ("uart_hw_t* hw", as returned from `UART_GetHw()` function). Functions in the second case are marked with "`_hw`". Addressing by alarm index is easier to use, addressing by pointer can generate slightly better optimized code.

Parity mode

<code>UART_PARITY_NONE</code>	no parity bit
<code>UART_PARITY_ODD</code>	odd parity
<code>UART_PARITY_EVEN</code>	even parity
<code>UART_PARITY_ONE</code>	one parity
<code>UART_PARITY_ZERO</code>	zero parity

FIFO trigger points (every FIFO is 32 entries deep)

<code>UART_LEVEL_VERYLOW</code>	1/8 (4 + 28 entries)
<code>UART_LEVEL_LOW</code>	1/4 (2/8) (8 + 24 entries)
<code>UART_LEVEL_MID</code>	1/2 (4/8) (16 + 16 entries) ... default
<code>UART_LEVEL_HIGH</code>	3/4 (6/8) (24 + 8 entries)
<code>UART_LEVEL_VERYHIGH</code>	7/8 (28 + 4 entries)

UART interrupt

UART_IRQ_OVERRUN	overrun error
UART_IRQ_BREAK	break error
UART_IRQ_PARITY	parity error
UART_IRQ_FRAME	framing error
UART_IRQ_TIMEOUT	receive timeout
UART_IRQ_TX	transmit
UART_IRQ_RX	receive
UART_IRQ_CTS	CTS

uart_hw_t* UART_GetHw(int uart);

uart ... UART index 0 or 1

Get UART hardware interface from UART index. Returns uart0_hw or uart1_hw.

u8 UART_GetInx(const uart_hw_t* hw);

hw ... pointer to UART hardware interface (as returned from UART_GetHw())

Get UART index from UART hardware interface.

UART data

u16 UART_RecvCharFlags(int uart);

u16 UART_RecvCharFlags_hw(uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from UART_GetHw())

Receive character with error flags. Error flags are cleared when condition disappears. Every entry in receive FIFO contains 8 bit character with 4 error bits.

Returned data with error flags:

bit 11: OE overrun error (receive data is lost)

bit 10: BE break error (long "LOW" on RX input)

bit 9: parity error (incorrect parity of received character)

bit 8: framing error (invalid stop bit)

bit 0..7: data byte

char UART_RecvChar(int uart);

char UART_RecvChar_hw(uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from UART_GetHw())

Receive character, does not wait if not ready.

```
void UART_SendChar(int uart, char data);
void UART_SendChar_hw(uart_hw_t* hw, char data);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Send character, does not wait if not ready.
```

UART errors

```
Bool UART_OverrunError(int uart);
Bool UART_OverrunError_hw(const uart_hw_t* hw);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Check overrun error. Returns True if receive FIFO is full and data is lost.
```

```
void UART_OverrunClear(int uart);
void UART_OverrunClear_hw(uart_hw_t* hw);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Clear overrun error.
```

```
Bool UART_BreakError(int uart);
Bool UART_BreakError_hw(const uart_hw_t* hw);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Check break error. Returns True if break condition 'LOW' detected.
```

```
void UART_BreakClear(int uart);
void UART_BreakClear_hw(uart_hw_t* hw);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Clear break error.
```

```
Bool UART_ParityError(int uart);
Bool UART_ParityError_hw(const uart_hw_t* hw);

    uart ... UART index 0 or 1
    hw ... pointer to UART hardware interface (as returned from UART_GetHw())
Check parity error. Returns True if incorrect parity received.
```

void UART_ParityClear(int uart);

void UART_ParityClear_hw(uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Clear parity error.

Bool UART_FrameError(int uart);

Bool UART_FrameError_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check framing error. Returns True if invalid stop bit.

void UART_FrameClear(int uart);

void UART_FrameClear_hw(uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Clear framing error.

UART FIFO

Bool UART_TxEmpty(int uart);

Bool UART_TxEmpty_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if transmit FIFO is empty.

Bool UART_RxFull(int uart);

Bool UART_RxFull_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if receive FIFO is full.

Bool UART_TxFull(int uart);

Bool UART_TxFull_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if transmit FIFO is full.

Bool UART_RxEmpty(int uart);

Bool UART_RxEmpty_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if receive FIFO is empty.

Bool UART_Busy(int uart);

Bool UART_Busy_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if UART is busy transmitting data from shift register.

Bool UART_CTS(int uart);

Bool UART_CTS_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Check if CTS (clear to send) input is LOW (active).

UART setup

void UART_IBaud(int uart, int ibaud);

void UART_IBaud_hw(uart_hw_t* hw, int ibaud);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

ibaud ... integer part of baud rate divisor

Set integer part of baud rate divisor.

u16 UART_GetIBaud(int uart);

u16 UART_GetIBaud_hw(const uart_hw_t* hw);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Get integer baud rate divisor.

void UART_FBaud(int uart, int fbaud);

void UART_FBaud_hw(uart_hw_t* hw, int fbaud);

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

fbaud ... fractional part of baud rate divisor

Set fractional baud rate divisor (= 1/64 fraction).

```
u8 UART_GetFBAud(int uart);
u8 UART_GetFBAud_hw(const uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Get fractional baud rate divisor (= 1/64 fraction).

```
void UART_Baud(int uart, u32 baudrate);
void UART_Baud_hw(uart_hw_t* hw, u32 baudrate);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

baudrate ... required baudrate

Set baudrate. Available range is 45 to 3'000'000 bps with default `clk_peri` = 48 MHz.
Hardware supports up to 921'600 bps.

```
u32 UART_GetBaud(int uart);
u32 UART_GetBaud_hw(const uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Get current baudrate.

```
void UART_Parity(int uart, int parity);
void UART_Parity_hw(uart_hw_t* hw, int parity);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

parity ... parity `UART_PARITY_*`

Set parity `UART_PARITY_*`. Default is `UART_PARITY_NONE`.

```
void UART_Word(int uart, int len);
void UART_Word_hw(uart_hw_t* hw, int len);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

len ... data length 5..8 bits

Set word length to 5..8 (default is 5 bits).

```
void UART_FifoEnable(int uart);
void UART_FifoEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Enable FIFO. Every FIFO is 32 entries deep.

```
void UART_FifoDisable(int uart);
```

```
void UART_FifoDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Disable FIFO (default state; every FIFO will be 1 entry deep).

```
void UART_Stop1(int uart);
```

```
void UART_Stop1_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Set 1 stop bit (default value).

```
void UART_Stop2(int uart);
```

```
void UART_Stop2_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Set 2 stop bits.

```
void UART_BreakOn(int uart);
```

```
void UART_BreakOn_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Start sending break signal - set TX output to LOW (min. for 2 frames; get `UART_BreakError` state).

```
void UART_BreakOff(int uart);
```

```
void UART_BreakOff_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Stop sending break signal.

```
void UART_CTSEnable(int uart);
```

```
void UART_CTSEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

CTS hardware flow control enable. If enabled, data is only transmitted when CTR signal is LOW.

```
void UART_CTSDisable(int uart);
void UART_CTSDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

CTS hardware flow control disable (default state). If disabled, data is always transmitted.

```
void UART_RTSEnable(int uart);
void UART_RTSEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

RTS hardware flow control enable. If enabled, data is requested by setting the RTS signal to LOW only if there is free space in the receive FIFO.

```
void UART_RTSDisable(int uart);
void UART_RTSDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

RTS hardware flow control disable (default state). If disabled, RTS signal is not controlled.

```
void UART_RTSDOutON(int uart);
void UART_RTSDOutON_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Set RTS output ON. Output RTS signal when RTS control is disabled: set RTS output to LOW.

```
void UART_RTSDOutOFF(int uart);
void UART_RTSDOutOFF_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Set RTS output OFF (default state). Output RTS signal when RTS control is disabled: set RTS output to HIGH.

```
void UART_RxEnable(int uart);
void UART_RxEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Enable receiver (default state).

```
void UART_RxDisable(int uart);
```

```
void UART_RxDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Disable receiver.

```
void UART_TxEnable(int uart);
```

```
void UART_TxEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Enable transmitter (default state).

```
void UART_TxDisable(int uart);
```

```
void UART_TxDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Disable transmitter.

```
void UART_LoopEnable(int uart);
```

```
void UART_LoopEnable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Enable loopback test.

```
void UART_LoopDisable(int uart);
```

```
void UART_LoopDisable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Disable loopback test (default state).

```
void UART_Enable(int uart);
```

```
void UART_Enable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

UART enable.

```
void UART_Disable(int uart);
```

```
void UART_Disable_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

UART disable (default state).

```
void UART_RxLevel(int uart, int level);
```

```
void UART_RxLevel_hw(uart_hw_t* hw, int level);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

level ... interrupt level `UART_LEVEL_*`

Set receive interrupt FIFO level select `UART_LEVEL_*` (default `UART_LEVEL_MID`).

```
void UART_TxLevel(int uart, int level);
```

```
void UART_TxLevel_hw(uart_hw_t* hw, int level);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

level ... interrupt level `UART_LEVEL_*`

Set transmit interrupt FIFO level select `UART_LEVEL_*` (default `UART_LEVEL_MID`).

UART interrupt

```
void UART_EnableIRQ(int uart, int irq);
```

```
void UART_EnableIRQ_hw(uart_hw_t* hw, int irq);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

irq ... interrupt `UART_IRQ_*`

Enable interrupt `UART_IRQ_*`.

```
void UART_DisableIRQ(int uart, int irq);
```

```
void UART_DisableIRQ_hw(uart_hw_t* hw, int irq);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

irq ... interrupt `UART_IRQ_*`

Disable interrupt `UART_IRQ_*`.

```
Bool UART_RawIRQ(int uart, int irq);
```

```
Bool UART_RawIRQ_hw(const uart_hw_t* hw, int irq);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

irq ... interrupt **UART_IRQ_***

Check if raw interrupt (before masking) **UART_IRQ_*** is raised.

```
Bool UART_IRQ(int uart, int irq);
```

```
Bool UART_IRQ_hw(const uart_hw_t* hw, int irq);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

irq ... interrupt **UART_IRQ_***

Check if masked interrupt (after enabling) **UART_IRQ_*** is raised.

```
void UART_ClearIRQ(int uart, int irq);
```

```
void UART_ClearIRQ_hw(uart_hw_t* hw, int irq);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

irq ... interrupt **UART_IRQ_***

Clear interrupt **UART_IRQ_***.

UART DMA

```
void UART_EnableErrDMA(int uart);
```

```
void UART_EnableErrDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Enable DMA on error.

```
void UART_DisableErrDMA(int uart);
```

```
void UART_DisableErrDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Disable DMA on error (default state).

```
void UART_EnableTxDMA(int uart);
```

```
void UART_EnableTxDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Enable transmit DMA.

```
void UART_DisableTxDMA(int uart);
void UART_DisableTxDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Disable transmit DMA (default state).

```
void UART_EnableRxDMA(int uart);
void UART_EnableRxDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Enable receive DMA.

```
void UART_DisableRxDMA(int uart);
void UART_DisableRxDMA_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Disable receive DMA (default state).

UART extended functions

```
void UART_SoftReset(int uart);
void UART_SoftReset_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Soft reset UART (restore to its default state).

```
void UART_HardReset(int uart);
void UART_HardReset_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Hard reset UART (send reset signal).

```
void UART_Reset(int uart);
void UART_Reset_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1
hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)
Reset UART to its default state.

```
void UART_Init(int uart, u32 baudrate, u8 word, u8 parity, u8 stop, Bool hw);  
void UART_Init_hw(uart_hw_t* hw, u32 baudrate, u8 word, u8 parity, u8 stop,  
    Bool flow);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

baudrate ... baudrate 45 to 3'000'000 Baud; hardware support up to 921'600 bps

word ... word length 5 to 8 bits

parity ... parity **UART_PARITY_***

stop ... number of stop bits 1 or 2

hw ... use hardware flow control CTS/RTS

Initialize UART and reset to default state. GPIO pins are not initialized. After init, connect UART to GPIO pins using **GPIO_Fnc(pin, GPIO_FNC_UART)**. After connecting the GPIO, wait a moment with **WaitUs(50)** and then flush any fake received characters with **UART_RecvFlush()**.

```
void UART_InitDef(int uart);  
void UART_InitDef_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Default initialize UART to 115200 Baud, 8 bits, no parity, 1 stop, no hw control. GPIO pins are not initialized. After init, connect UART to GPIO pins using **GPIO_Fnc(pin, GPIO_FNC_UART)**. After connecting the GPIO, wait a moment with **WaitUs(50)** and then flush any fake received characters with **UART_RecvFlush()**.

```
void UART_TxWait(int uart);  
void UART_TxWait_hw(const uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Waiting for end of transmission (including transmission shift register).

```
Bool UART_SendReady(int uart);  
Bool UART_SendReady_hw(const uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Check if next character can be transmitted.

```
Bool UART_RecvReady(int uart);  
Bool UART_RecvReady_hw(const uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from **UART_GetHw()**)

Check if next character can be received.

```
void UART_SendCharWait(int uart, char data);
void UART_SendCharWait_hw(uart_hw_t* hw, char data);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

data ... byte to send

Send one character with waiting.

```
char UART_RecvCharWait(int uart);
char UART_RecvCharWait_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Receive one character with waiting.

```
void UART_RecvFlush(int uart);
void UART_RecvFlush_hw(uart_hw_t* hw);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

Flush received characters.

```
void UART_Send(int uart, const u8* src, u32 len);
void UART_Send_hw(uart_hw_t* hw, const u8* src, int len);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

src ... source buffer with data to send

len ... length of data to send (number of bytes)

Send buffer with blocking (waits to be ready).

```
u32 UART_Recv(int uart, u8* dst, u32 len);
u32 UART_Recv_hw(uart_hw_t* hw, u8* dst, int len);
```

uart ... UART index 0 or 1

hw ... pointer to UART hardware interface (as returned from `UART_GetHw()`)

dst ... destination buffer to receive data

len ... length of data to receive (number of bytes)

Receive buffer without blocking, returns number of successfully read characters.

UART stream

```
void StreamWriteUartInit(sStream* str, int uart);
```

str ... data stream

uart ... UART index 0 or 1

Initialize stream to write to UART with blocking.

u32 `UART_PrintArg(int uart, const char* fmt, va_list args);`

uart ... UART index 0 or 1

fmt .. format string

args ... arguments

Formatted print with blocking, with argument list. Returns number of characters.

u32 `UART_Print(int uart, const char* fmt, ...);`

uart ... UART index 0 or 1

fmt .. format string

... ... arguments

Formatted print with blocking, with variadic arguments. Returns number of characters.

UART-stdio configuration

USE_UART_STDIO	use UART-stdio
UART_STDIO_PORT	UART stdio port 0 or 1
UART_STDIO_TX	UART stdio TX GPIO pin
UART_STDIO_RX	UART stdio RX GPIO pin
UART_STDIO_TXBUF	size of transmit ring buffer of UART stdio
UART_STDIO_RXBUF	size of receive ring buffer of UART stdio
UART_STDIO_TXSPIN	transmitter spinlock 0..31 (-1 = not used)
UART_STDIO_RXSPIN	receiver spinlock 0..31 (-1 = not used)

Bool `UartSendReady();`

Check if next character can be sent to uart-stdio (must be enabled with USE_UART_STDIO).

Bool `UartRecvReady();`

Check if next character can be received from uart-stdio (must be enabled with USE_UART_STDIO).

void `UartPrintChar(char ch);`

ch ... character to print

Print character to uart-stdio (must be enabled with USE_UART_STDIO).

u32 `UartPrintText(const char* txt);`

txt ... text to print

Print unformatted text to uart-stdio (must be enabled with USE_UART_STDIO). Returns number of characters.

u32 UartPrintArg(const char* fmt, va_list args);

fmt .. format string

args ... arguments

Formatted print string to uart-stdio (must be enabled with USE_UART_STDIO), with argument list. Returns number of characters.

u32 UartPrint(const char* fmt, ...);

fmt .. format string

... ... arguments

Formatted print string to uart-stdio (must be enabled with USE_UART_STDIO), with variadic arguments. Returns number of characters.

char UartGetChar();

Receive character from uart-stdio (must be enabled with USE_UART_STDIO). Returns NOCHAR if no character.

void UartStdioInit();

Initialize uart-stdio (must be enabled with USE_UART_STDIO). This function is automatically called during application startup from the internal RuntimeInit() function.

void UartStdioTerm();

Terminate uart-stdio (must be enabled with USE_UART_STDIO). This function is automatically called during application termination from the internal RuntimeTerm() function.

2.36. Voltage Regulator

Files: `sdk_vreg.h`, `sdk_vreg.c`

Config: - (only RP2040)

The library sets the chip voltage. On the RP2350, this feature is available in the PowMan (Power Manager) module.

Possible voltage values:

<code>VREG_VOLTAGE_0_80</code>	<code>5</code>	// 0.80V
<code>VREG_VOLTAGE_0_85</code>	<code>6</code>	// 0.85V
<code>VREG_VOLTAGE_0_90</code>	<code>7</code>	// 0.90V
<code>VREG_VOLTAGE_0_95</code>	<code>8</code>	// 0.95V
<code>VREG_VOLTAGE_1_00</code>	<code>9</code>	// 1.00V
<code>VREG_VOLTAGE_1_05</code>	<code>10</code>	// 1.05V
<code>VREG_VOLTAGE_1_10</code>	<code>11</code>	// 1.10V *default state
<code>VREG_VOLTAGE_1_15</code>	<code>12</code>	// 1.15V
<code>VREG_VOLTAGE_1_20</code>	<code>13</code>	// 1.20V
<code>VREG_VOLTAGE_1_25</code>	<code>14</code>	// 1.25V
<code>VREG_VOLTAGE_1_30</code>	<code>15</code>	// 1.30V
<code>VREG_VOLTAGE_MIN</code>	<code>VREG_VOLTAGE_0_80</code>	// minimum voltage
<code>VREG_VOLTAGE_DEF</code>	<code>VREG_VOLTAGE_1_10</code>	// default after power up
<code>VREG_VOLTAGE_MAX</code>	<code>VREG_VOLTAGE_1_30</code>	// maximum voltage

void VregSetVoltage(int vreg);
vreg ... voltage `VREG_VOLTAGE_*`

Set voltage.

u8 VregVoltage();
Get current voltage `VREG_VOLTAGE_*`.

float VregVoltageFloat();
Get current voltage in volts.

Bool VregIsOk();
Check if voltage is correctly regulated.

void VregWait();

Wait for regulated state.

2.37. Watchdog Timer

Files: `sdk_watchdog.h`, `sdk_watchdog.c`

Config: `USE_WATCHDOG` (default 1)

The watchdog is a countdown timer that can restart parts of the chip if it reaches zero. This can be used to restart the processor if software gets stuck in an infinite loop. The programmer must periodically write a value to the watchdog to stop it from reaching zero. The watchdog reference clock, `clk_tick`, is driven from `clk_ref`. The reference clock is divided internally to generate a tick (nominally 1 μ s) to use as the watchdog tick. On RP2040, the generated clock is also used to control the time counter.

Watchdog scratch registers during soft reboot

scratch register 4: magic number 0xb007c0d3 (will be cleared before transferring control)

scratch register 5: entry point XORed with magic -0xb007c0d3 (0x4ff83f2d)

scratch register 6: stack pointer

scratch register 7: entry point

void WatchdogStart(u32 xosc_mhz);

`xosc_mhz` ... frequency of XOSC in MHz (it sets divisor to generate 1 MHz)

Only RP2040. Start watchdog tick generator. Generated ticks are shared with main timer counter. RP2050 uses Ticks module as watchdog clock source.

void WatchdogUpdate();

Update watchdog (restart time counting).

u32 WatchdogGetCount();

Get current value of watchdog generator in number of us.

u8 WatchdogReason();

Get reset reason (0=hardware, 1=watchdog timer, 2=force trigger).

<code>RESET_REASON_HW</code>	0	hardware reset
<code>RESET_REASON_TIMER</code>	1	watchdog timer reset
<code>RESET_REASON_FORCE</code>	2	software force reset (<code>WatchdogTrigger()</code>)

void WatchdogTrigger();

Trigger watchdog reset.

void WatchdogSetup(u32 us, Bool pause_debug);

`us` ... delay before reset in [us], max. 0xffffffff us (8388607 us = 8.4 seconds)

`pause_debug` ... pause watchdog on debugging

Setup watchdog generator. Does not set reason into scratch registers.

void WatchdogEnable(u32 us, Bool pause_debug);

us ... delay before reset in [us], max. 0xffffffff us (8388607 us = 8.4 seconds)

pause_debug ... pause watchdog on debugging

Watchdog enable. Set reason WATCHDOG_NON_REBOOT_MAGIC.

void WatchdogSetupReset(u32 us, Bool pause_debug);

us ... delay before reset in [us], max. 0xffffffff us (8388607 us = 8.4 seconds)

pause_debug ... pause watchdog on debugging

Setup watchdog generator to do hard reset.

void WatchdogSetupReboot(u32 us, u32 pc, u32 sp);

us ... delay before reset in [us], max. 0xffffffff us (8388607 us = 8.4 seconds)

pc ... pointer to code to restart in soft reset (0 = do hard reset)

sp ... stack pointer

Setup watchdog generator to do soft reset.

void WatchdogTrigger();

Trigger watchdog reset.

u8 WatchdogReason();

Get reset reason (0=hardware, 1=watchdog timer, 2=force trigger).

Bool WatchdogReasonEnable();

Check reset reason WatchdogEnable().

void ResetToBootLoader();

Reset to boot loader.

2.38. XIP - Flash control

Files: `sdk_xip.h`, `sdk_xip.c`

Config: -

XIP module controls flash memory.

`void FlashCacheFlush();`

Only RP2040. Flush flash cache, does not wait.

`void FlashCacheFlushWait();`

Only RP2040. Flush flash cache and wait to complete.

`void FlashCacheEnable();`

Enable flash cache.

`void FlashCacheDisable();`

Disable flash cache.

2.39. XOSC - Crystal Oscillator

Files: `sdk_xosc.h`, `sdk_xosc.c`

Config: `USE_XOSC` (default 1)

The Crystal Oscillator (XOSC) uses an external crystal to produce an accurate reference clock. The RP2040 supports 1MHz to 15MHz crystals. Raspberry Pico uses a 12MHz crystal. The reference clock is distributed to the PLLs, which can be used to multiply the XOSC frequency to provide accurate high speed clocks.

void XoscEnable();

Enable crystal oscillator.

void XoscDisable();

Disable crystal oscillator. Cannot be disabled, if CPU uses it as a clock source.

void XoscSetRange(int range);

`range` ... frequency range `XOSC_RANGE_*`

Setup frequency range `XOSC_RANGE_*`.

`XOSC_RANGE_0` 0xAA0 1..15 MHz

`XOSC_RANGE_1` 0xAA1 10..30 MHz (only RP2350)

`XOSC_RANGE_2` 0xAA2 25..60 MHz (only RP2350)

`XOSC_RANGE_3` 0xAA3 40..100 MHz (only RP2350)

void XoscInit();

Initialize crystal oscillator. This function is called internally from the `ClockInit()` function during application startup.

void XoscDormant();

Start dormant mode of the crystal oscillator.

void XoscWake();

Wake up crystal oscillator from dormant mode.

Bool XoscIsStable();

Check if crystal oscillator is running and stable.

void XoscWait();

Wait for stable state of crystal oscillator (XOSC must be enabled).

void XoscSetCount(u8 count);

Set counter of crystal oscillator.

u8 XoscGetCount();

Get counter of crystal oscillator. After setting the value, it runs down to zero and stops there.

3. Tool Library

The Tool Library contains the main support resources for the SDK. The Tool Library source files are located in the `_lib` folder.

3.1. Alarm

Files: `lib_alarm.h`, `lib_alarm.c`

Config: `USE_ALARM (default 1)`

The alarm library provides time alarms triggered from the SysTick handler, with resolution 5 ms. Every CPU core has its own alarm list.

typedef void (*pAlarmCB(sAlarm* alarm);

alarm ... pointer to alarm structure

Prototype of alarm callback. Callback function can unregister alarm if it is no longer needed.

System alarm structure

typedef struct {

sListEntry	list;	<i>// list of alarms</i>
u32	time;	<i>// system time in ms to alarm</i>
u32	delta;	<i>// delta time to repeat in ms (if repetition</i>
		<i>// is not needed, the alarm must be</i>
		<i>// unregistered from the callback)</i>
pAlarmCB	callback;	<i>// callback function (called from</i>
		<i>// SysTick_Handler with IRQ locked)</i>
u32	cookie;	<i>// user data</i>

} sAlarm;

void AlarmReg(sAlarm* alarm);

alarm ... pointer to alarm structure

Register alarm. Caller must unregister alarm after waking up if it is no longer needed. Callback will be called under the same core from which the alarm was registered. Registered alarm structure must remain valid until the alarm is unregistered.

void AlarmUnreg(sAlarm* alarm);

alarm ... pointer to alarm structure

Unregister alarm. Need to be done after waking up if alarm is no longer needed. Must be called from the same CPU core as it was registered.

void AlarmUpdate();

Update alarms for this CPU core. This function is called from SysTick_Handler every SYSTICK_MS milliseconds with IRQ locked.

void AlarmlInit();

Initialize alarm lists for both cores.

3.2. Calendar - Short 32-bit Calendar

Files: lib_calendar.h, lib_calendar.c

Config: USE_CALENDAR (default 1)

Short 32-bit calendar is based on 32-bit Unix time - number of seconds since 1/1/1970, stored in one 32-bit variable. Calendar range is from 1/1/1970 to 12/31/2099.

Date and time structure (12 bytes)

```
typedef struct {
    union {
        struct {
            s16    us;           // microseconds 0..999
            s16    ms;           // milliseconds 0..999
        };
        s32    timelow;      // microseconds and milliseconds
    };
    union {
        struct {
            s8     dayofweek;   // day of week 0=Sunday..6=Saturday
            s8     sec;          // second 0..59
            s8     min;          // minute 0..59
            s8     hour;         // hour 0..23
        };
        s32    time;          // time (with day of week)
    };
    union {
        struct {
            s8     day;          // day in month 1..31
            s8     mon;          // month 1..12
            s16    year;         // year year 1970..2099
        };
        s32    date;          // date
    };
} sDateTime;
```

Limits

MINYEAR	1970 min. year
MAXYEAR	2099 max. year
MAXDAY	47481 max. day (12/31/2099)
MAXTIME	(60*60*24-1) max. time of day in seconds (=86399, 23:59:59)
MAXUNIXDT	4102444799 max. value of Unix date and time value (=0xF48656FF) Unix time 0 = 00:00:00 1/1/1970 Unix time 4102444799 (0xF48656FF) = 23:59:59 12/31/2099

Day of week

DAY_SUN	0 Sunday
DAY_MON	1 Monday
DAY_TUE	2 Tuesday
DAY_WED	3 Wednesday
DAY_THU	4 Thursday
DAY_FRI	5 Friday
DAY_SAT	6 Saturday

Bool YearIsLeap(s16 year);

year ... year

Check if year is leap using Greagorian calendar (=February has 29 days instead of 28).

void DateTimeClear(sDateTime* dt);

dt ... pointer to sDateTime structure

Clear date and time structure (set date and time to 0:0:0 1/1/1970 Thursday).

void DateTimeSet(sDateTime* dt, s16 year, s8 mon, s8 day, s8 hour, s8 min, s8 sec, s16 ms, s16 us);

dt ... pointer to sDateTime structure

year ... year

mon ... month

day ... day

hour ... hour

min ... minute

sec ... second

ms ... millisecond

us ... microsecond

Set date and time (day of week is cleared).

void DateTimeCopy(sDateTime* dst, const sDateTime* src);

dst ... pointer to destination sDateTime structure

src ... pointer to source sDateTime structure

Copy date and time from another structure,

s8 DateTimeComp(const sDateTime* dt1, const sDateTime* dt2);

dt1 ... pointer to 1st sDateTime structure

dt2 ... pointer to 2nd sDateTime structure

Compare date and time (day of week is ignored). Returns -1 if dt1<dt2, 0 if dt1==dt2, 1 if dt1>dt2.

Bool DateTimeCheck(const sDateTime* dt);

dt ... pointer to sDateTime structure

Check date and time (not day of week; returns True = ok).

Bool DateTimeValid(sDateTime* dt);

dt ... pointer to sDateTime structure

Validate date and time (not day of week). Returns True = no corrections, entries were in valid ranges. If some entry is out of valid range, it corrects it and returns False.

Bool DateTimeInc(sDateTime* dt);

dt ... pointer to sDateTime structure

Increase date by 1 day. Returns False on date overflow.

Bool DateTimeDec(sDateTime* dt);

dt ... pointer to sDateTime structure

Decrease date by 1 day. Returns False on date underflow.

u32 DateTimePack(const sDateTime* dt, s16* ms, s16* us);

dt ... pointer to source sDateTime structure

ms ... pointer to destination milliseconds (NULL = not needed)

us ... pointer to destination microseconds (NULL = not needed)

Pack date and time into Unix time, optionally with ms and us if not NULL (day of week is ignored).

void DateTimeUnpack(sDateTime* dt, u32 ut, s16 ms, s16 us);

dt ... pointer to destination sDateTime structure

ut ... Unix time

ms ... milliseconds 0..999

us ... microseconds 0..999

Unpack date and time from Unix time. Updates day of week too.

s16 DateTimeCheck();

Datetime debug check. Returns year of error or 0 if all OK. The function goes through all possible Unix time values and compares the conversions forward and backward.

3.3. Calendar 64-bit - Long 64-bit Calendar

Files: lib_calendar64.h, lib_calendar64.c

Config: USE_CALENDAR64 (default 1)

Long 64-bit calendar uses 64-bit signed number to store date and time as number of 100-nanoseconds since Saturday 1/1/1 CE 0:00:00 ("absolute time"). Its range is from year 29227 BCE to year 29227 CE. It uses Julian calendar until date 10/4/1582, and from date 10/15/1582 uses Gregorian calendar (10 days was skipped in 1582).

Absolute time is s64 number of 100-nanoseconds since Saturday 1/1/1 CE 0:00:00, year range 29227 BCE to 29227 CE.

Absolute day is s32 number of days since Saturday 1/1/1 CE (that is day 0), range -10675162 to +10674943.

Julian date is number of days since Monday 1/1/4713 BCE noon 12:00:00, expressed as a decimal number (include time) as double number. Julian date has period 7980 years, then starts again. Julian date has range 1/1/4713 BCE 12:00 to 1/1/3268 CE 12:00, e.g. 0 to 2914673 days. Julian period is 2518277472000000000 100-nanoseconds (e.g. 7980 years).

Unix time is u32 number of seconds since Thursday 1/1/1970 CE 0:00:00 (u32 number, that is absolute day 719164).

File time is number of 100-nanosecons since Monday 1/1/1601 CE 0:00:00 (that is absolute day 584390).

Julian calendar was used till Thursday 10/4/1582 (begins Julian date 2299159.5). Julian year has 365.25 days.

Gregorian calendar was used from Friday 10/15/1582 (begins Julian date 2299160.5). 10 days was skipped. Year 1582 has 355 days (it is not leap), 51 weeks. Gregorian year has 365.2425 days.

Date 1/1/1 CE was Julian Date 1721423.5, Saturday (there is no year 0!).

Date 1/1/1 BCE was Julian Date 1721057.5 Thursday.

ISO-8601: Week is 1 to 53, day of week: 1 to 7, 1=Monday. First week of year is such a week, which contains first Thursday of Gregorian year, and week containing January 4th.

Day of week

DAY_SUN	0 Sunday
DAY_MON	1 Monday
DAY_TUE	2 Tuesday
DAY_WED	3 Wednesday
DAY_THU	4 Thursday
DAY_FRI	5 Friday
DAY_SAT	6 Saturday

Date and time structure

```
typedef struct {
    union {
        struct {
            // extended entries
            u8    flags;          // flags DTFLAG_*
            s8    week;           // week in year 1..53
            s16   dayofyear;      // day of year 1..366
            // nanoseconds
            s32   nsec;           // nanoseconds 0..999'999'999
            // time (+ day of week)
            s8    dayofweek;      // day of week 0=Sunday..6=Saturday
            s8    sec;             // second 0..59
            s8    min;             // minute 0..59
            s8    hour;            // hour 0..23
            // date
            s8    day;              // day in month 1..31
            s8    mon;             // month 1..12
            s16   year;            // year -29226 .. +29227 (value 0 is year 1 BCE)
        };
        struct {
            s32   ext;             // extended entries
            s32   nsec2;           // nanoseconds
            s32   time;            // time (with day of week)
            s32   date;            // date
        };
    };
} sDateTime64;
```

Datetime flags (bit mask)

Special case: year 1582 has 355 days, not leap, 51 weeks

DTFLAG_LEAP	(B0) leap year
DTFLAG_GREG	(B1) date of gregorian calendar (or julian otherwise)
DTFLAG_W53	(B2) year has 53 weeks (or 52 weeks otherwise)
DTFLAG_LASTW	(B3) last week of previous year
DTFLAG_FIRSTW	(B4) first week of next year

Constants and ranges

MINYEAR2	-29226 min. year (= 29227 BCE)
MAXYEAR2	+29227 max. year (= 29227 CE), total 58454 years (0xE456)
MINDAY2	-10675162 min. absolute day
MAXDAY2	+10674943 max. absolute day, total 21350106 days (0x145C6DA)
JULIANBASE	1721423.5 base Julian day (1/1/1 CE, 0:00:00)
ABSTODAYCOEF	1.1574074074074074074e-12 abs. time to Julian days coefficient
DAYTOABSCOEF	864000000000.0 Julian days to absolute time coefficient
MAXTIME	(60*60*24-1) max. time of day in seconds (=86399, 23:59:59)
DAY100NS	((s64)24*60*60*10000000) one day in 100-ns (=864000000000)
MAXUNIXDT	4102444799 max. value of Unix date and time value (=0xF48656FF) min. Unix time 0 = 00:00:00 1/1/1970 max. Unix time 4102444799 (0xF48656FF) = 23:59:59 12/31/2099
SPLITYEAR	1582 split year Julian -> Gregorian
SPLITDATE	577737 split day (first day of Gregorian calendar) 10/15/1582
FILETIMEOFF	(s64)504912960000000000 Windows file time base offset
FILETIMEDAY	584390 absolute day of Windows file time base (Monday 1/1/1601 CE)
UNIXBASEOFF	(s64)621357696000000000 Unix base offset
UNIXBASEDAY	719164 absolute day of Unix base (Thursday 1/1/1970 CE)
MINTIME2	(s64)-9223339968000000000 minimal absolute time, 0x8000:1D2A:9CB2:8000 (Friday 1/1/29227 BCE 0:0:0.0)
MAXTIME2	(s64)9223151615999999999 maximal absolute time, 0x7FFF:3787:453F:FFFF (Friday 12/31/29227 CE 23:59:59.9999999) total 1844649158400000000 ticks, 0xFFFF:1A5C:A88D:8000

Bool DateTime64IsLeap(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Get flag from datetime structure - leap year DTFLAG _LEAP.

Bool DateTime64IsGreg(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Get flag from datetime structure - date belongs to the Gregorian calendar (or Julian otherwise).

Bool DateTime64IsW53(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Get flag from datetime structure - year has 53 weeks (or 52 weeks otherwise).

Bool DateTime64IsLastW(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Get flag from datetime structure - last week of previous year.

Bool DateTime64IsFirstW(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Get flag from datetime structure - first week of next year.

double AbsTimeToJulian(s64 time);

time .. absolute time

Convert absolute time to Julian date.

s64 JulianToAbsTime(double jul);

jul ... Julian date

Convert Julian date to absolute time.

s32 AbsTimeToDay(s64 time);

time .. absolute time

Convert absolute time to absolute day (MINDAY2..MAXDAY2).

s64 DayToAbsTime(s32 day);

day ... absolute day

Convert absolute day to absolute time.

s64 AbsTimeToFileTime(s64 time);

time .. absolute time

Convert absolute time to filetime (Windows).

s64 FileTimeToAbsTime(s64 time);

time .. filetime

Convert filetime (Windows) to absolute time.

u32 AbsTimeToUnixTime(s64 time, s16* ms, s16* us);

time .. absolute time

ms ... pointer to get milliseconds (NULL = not needed)

us ... pointer to get microseconds (NULL = not needed)

Convert absolute time to Unix time, milliseconds and microseconds.

s64 UnixTimeToAbsTime(u32 ut, s16 ms, s16 us);

ut .. Unix time

ms ... milliseconds

us ... microseconds

Convert Unix time with milliseconds and microseconds to absolute time.

Bool YearIsLeap64(s16 year);

year ... year -29226 to +29227

Check if long year is leap year.

u8 GetLastDayMonth64(s8 mon, s16 year);

mon ... month 1 to 12

year ... year -29226 to +29227

Get last day in month (28..31, use long year).

s16 GetDaysInYear64(s16 year);

year ... year -29226 to +29227

Get days in long year (355, 365, 366).

void DateTime64Clear(sDateTime64* dt);

dt ... pointer to destination sDateTime64 structure

Clear date and time structure (set to 0:0:0 1/1/1).

void DateTime64Set(sDateTime64* dt, s16 year, s8 mon, s8 day, s8 hour, s8 min, s8 sec, s32 nsec);

dt ... pointer to destination sDateTime64 structure

year ... year -29226 .. +29227

mon ... month 1..12

day ... day in month 1..31

hour ... hour 0..23

min ... minute 0..58

sec ... second 0..59

nsec ... nanoseconds 0.. 999'999'999

Set date and time (day of week and flags are not updated).

void DateTime64Copy(sDateTime64* dst, const sDateTime64* src);

dst ... pointer to destination sDateTime64 structure

src ... pointer to source sDateTime64 structure

Copy date and time from another structure.

s8 DateTime64Comp(const sDateTime64* dt1, const sDateTime64* dt2);

dt1 ... pointer to first sDateTime64 structure

dt2 ... pointer to second sDateTime64 structure

Compare date and time (day of week is ignored). Returns -1 if dt1<dt2, 0 if dt1==dt2, 1 if dt1>dt2.

Bool DateTime64Valid(sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Validate date and time (not day of week). Returns True = no corrections, entries were in valid ranges. If some entry is out of valid range, it corrects it and returns False.

Bool DateTime64Inc(sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Increase date structure by 1 day. Extended entries are not updated. Returns False on overflow.

Bool DateTime64Dec(sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Decrease date structure by 1 day. Extended entries are not updated. Returns False on underflow.

s32 DateTime64PackDay(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Pack date into absolute day (day of week is ignored), range MINDAY2..MAXDAY2.

s64 DateTime64PackAbsTime(const sDateTime64* dt);

dt ... pointer to sDateTime64 structure

Pack date and time into absolute time (day of week is ignored).

void DateTime64UnpackDay(sDateTime64* dt, s32 day);

dt ... pointer to destination sDateTime64 structure

day ... absolute day

Unpack date from absolute day. Updates day of week too. Time will be reset to zero.

void DateTime64UnpackUnixTime(sDateTime64* dt, u32 ut, s16 ms, s16 us);

dt ... pointer to destination sDateTime64 structure

ut .. Unix time

ms ... milliseconds

us ... microseconds

Unpack date and time from Unix time. Updates day of week too.

```
void DateTime64UnpackAbsTime(sDateTime64* dt, s64 time);
```

dt ... pointer to destination sDateTime64 structure

time .. absolute time

Unpack date and time from absolute time. Updates day of week too.

```
void DebTestCalendar64Init(int year);
```

year ... year -29226 to +29227

Initialize debug test calendar64. As first start, use value MINYEAR2.

```
int DebTestCalendar64Step();
```

Do one step of debug test calendar (complete test takes a few minutes). Return values:

0 = running OK

1 = end of test

2 = invalid structure 1

3 = invalid structure 2

4 = invalid Nsec

5 = invalid sec

6 = invalid min

7 = invalid hour

8 = invalid day

9 = invalid month

10 = invalid year

11 = invalid day of week

12 = invalid abs. time

13 = invalid packed day

```
void DebTestCalendar64()
```

Long test calendar64 (using printf output).

```
s16 DateTime64FastCheck(int loops)
```

Fast check calendar64. Returns year on error or 0 if OK).

3.4. Canvas - Drawing Surface

Files: lib_canvas.h, lib_canvas.c

Config: USE_CANVAS (default 1), DRAW_HWINTER (default 1)

The Canvas library is used in conjunction with the QVGA or PicoVGA library to draw into a graphics buffer with a format of 1 to 8 bits per pixel. It is not used for 16 bits per pixel graphics buffers (such as those used by the TFT display PicoPad).

Canvas color format

CANVAS_8 8-bits per pixels

CANVAS_4 4-bits per pixels

CANVAS_2 2-bits per pixels

CANVAS_1 1-bit per pixels

CANVAS_PLANE2 4 colors on 2 planes

CANVAS_ATTRIB8 2x4 bit color attributes per 8x8 pixel sample
attribute on draw functions:

bit 0..3 = draw color

bit 4 = draw color is background color

Canvas structure

```
typedef struct {
    u8*    img;    // image data
    u8*    img2;   // image data 2 (2nd plane of CANVAS_PLANE2,
                  // or attributes of CANVAS_ATTRIB8)
    int    w;     // width
    int    h;     // height
    int    wb;    // pitch (bytes between lines)
    u8    format; // canvas format CANVAS_*
} sCanvas;
```

void CanvasRect(sCanvas* canvas, int x, int y, int w, int h, u8 col);

canvas ... pointer to sCanvas structure

x ... left coordinate

y ... top coordinate

w ... width

h ... height

col ... color

Draw rectangle.

void CanvasFrame(sCanvas* canvas, int x, int y, int w, int h, u8 col);

canvas ... pointer to sCanvas structure

x ... left coordinate

y ... top coordinate

w ... width

h ... height

col ... color

Draw frame (line 1 pixel thick).

void CanvasClear(sCanvas* canvas);

canvas ... pointer to sCanvas structure

Clear canvas (fill with black color).

void CanvasPoint(sCanvas* canvas, int x, int y, u8 col);

canvas ... pointer to sCanvas structure

x ... X coordinate

y ... Y coordinate

col ... color

Draw point.

void CanvasLine(sCanvas* canvas, int x1, int y1, int x2, int y2, u8 col);

canvas ... pointer to sCanvas structure

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

col ... color

Draw line.

void CanvasFillCircle(sCanvas* canvas, int x0, int y0, int r, u8 col, u8 mask);

canvas ... pointer to sCanvas structure

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

col ... draw color

mask ... mask of used octants (use 0xff = 255 = draw whole circle)

. B2|B1 .

B3 . | . B0

-----O-----

B4 . | . B7

. B5|B6 .

Draw filled circle.

void CanvasCircle(sCanvas* canvas, int x0, int y0, int r, u8 col, u8 mask);

canvas ... pointer to sCanvas structure

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

col ... draw color

mask ... mask of used octants (use 0xff = 255 = draw whole circle)

. B2|B1 .

B3 . | . B0

-----O-----

B4 . | . B7

. B5|B6 .

Draw circle.

void CanvasText(sCanvas* canvas, const char* text, int x, int y, u8 col, const void* font, int fontheight, int scalex, int scaley);

canvas ... pointer to sCanvas structure

text ... ASCIIZ text to draw (with trailing 0)

x ... X coordinate

y ... Y coordinate

col ... draw color

font ... pointer to 1-bit font, font width = 8 pixels

fontheight ... font height in lines

scalex ... scale text in X direction

scaley ... scale text in Y direction

Draw text with transparent background.

void CanvasSmallText(sCanvas* canvas, const char* text, int x, int y, u8 col, const void* font, int fontheight, int fontwidth);

canvas ... pointer to sCanvas structure

text ... ASCIIZ text to draw (with trailing 0)

x ... X coordinate
y ... Y coordinate
col ... draw color
font ... pointer to 1-bit font, font width < 8 pixels
fontheight ... font height in lines
fontwidth ... font width in pixels

Draw small text with transparent background.

void CanvasTextBg(sCanvas* canvas, const char* text, int x, int y, u8 col, u8 bgcol, const void* font, int fontheight, int scalex, int scaley);

canvas ... pointer to sCanvas structure
text ... ASCIIZ text to draw (with trailing 0)
x ... X coordinate
y ... Y coordinate
col ... draw color
bgcol ... background color
font ... pointer to 1-bit font, font width = 8 pixels
fontheight ... font height in lines
scalex .. scale text in X direction
scaley ... scale text in Y direction

Draw text with background.

void CanvasSmallTextBg(sCanvas* canvas, const char* text, int x, int y, u8 col, u8 bgcol, const void* font, int fontheight, int fontwidth);

canvas ... pointer to sCanvas structure
text ... ASCIIZ text to draw (with trailing 0)
x ... X coordinate
y ... Y coordinate
col ... draw color
bgcol ... background color
font ... pointer to 1-bit font, font width < 8 pixels
fontheight ... font height in lines
fontwidth ... font width in pixels

Draw small text with background.

void CanvasImg(sCanvas* canvas, sCanvas* src, int xd, int yd, int xs, int ys, int w, int h);

canvas ... pointer to destination sCanvas structure
src ... pointer to source sCanvas structure with source image

xd ... destination X coordinate
yd ... destination Y coordinate
xs ... source X coordinate
ys ... source Y coordinate
w ... image width
h ... image height

Draw image (or its part). Source and destination must have same format.

void CanvasBlit(sCanvas* canvas, sCanvas* src, int xd, int yd, int xs, int ys, int w, int h, u8 col);

canvas ... pointer to destination sCanvas structure
src ... pointer to source sCanvas structure with source image
xd ... destination X coordinate
yd ... destination Y coordinate
xs ... source X coordinate
ys ... source Y coordinate
w ... image width
h ... image height
col ... transparency key color

Draw image with transparency. Source and destination must have same format.
CANVAS_ATTRIB8 format will be replaced by CanvasImg function.

void CanvasImgMat(sCanvas* canvas, const sCanvas* src, int x, int y, int w, int h, const sMat2D* m, u8 mode, u8 color);

canvas ... pointer to destination sCanvas structure
src ... pointer to source sCanvas structure with source image
x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
m ... transformation matrix (should be prepared using
PrepDrawImg() function)

mode ... draw mode **CANVASIMG_***

color ... key or border color

Draw 8-bit image with 2D transformation matrix. Note to wrap and perspective mode: Width and height of source image must be power of 2. If **DRAW_HWINTER** configuration flag is set, hardware interpolator will be used.

CanvasImgMat mode:

CANVASIMG_WRAP	wrap image
CANVASIMG_NOBORDER	no border (transparent border)
CANVASIMG_CLAMP	clamp image (use last pixel as border)
CANVASIMG_COLOR	color border
CANVASIMG_TRANSP	transparent image with key color
CANVASIMG_PERSP	perspective floor

void CanvasTileMap(sCanvas* canvas, const sCanvas* src, const u8* map, int mapwbits, int maphbits, int tilebits, int x, int y, int w, int h, const sMat2D* mat, u8 horizon);

canvas ... pointer to destination sCanvas structure

src ... pointer to source sCanvas structure with column of 8-bit square tiles

(width = tile size, must be power of 2)

map ... byte map of tile indices

mapwbits ... number of bits of map width (number of tiles; width must be power of 2)

maphbits ... number of bits of map height (number of tiles; height is power of 2)

tilebits ... number of bits of tile size (e.g. 5 = tile 32x32 pixel)

x ... destination X coordinate

y ... destination Y coordinate

w ... destination width

h ... destination height

mat ... transformation matrix (should be prepared with PrepDrawImg())

mode ... draw mode **CANVASIMG_***

horizon ... horizon offset (0=do not use perspective projection)

Draw tile map using perspective projection. If **DRAW_HWINTER** configuration flag is set, hardware interpolator will be used.

void CanvasImgLine(sCanvas* canvas, sCanvas* src, int xd, int yd, int xs, int ys, int wd, int ws);

canvas ... destination canvas (8-bit pixel format)

src ... source canvas (source image in 8-bit pixel format)

xd, yd ... destination coordinates

xs, ys ... source coordinates

wd ... destination width

ws ... source width

Draw image line interpolated. Overflow in X direction is not checked. If **DRAW_HWINTER** configuration flag is set, hardware interpolator will be used.

3.5. Color - RGBA Color Vector

Files: lib_color.h, lib_color.c

Config: USE_COLOR (default 1)

The RGBA color vector contains the color in RGBA format and is used for color operations (blending, color space conversion). The color vector items are float numbers by default, but can be redefined to double numbers by editing the definition at the beginning of the lib_color.h file:

```
typedef float color_float; // color element (use float or double)
```

Color vector

Color values are typically in range 0..1.

```
typedef struct {  
    color_float r;          // red (or HUE)  
    color_float g;          // green (or SAT)  
    color_float b;          // blue (or LUM or VAL)  
    color_float a;          // alpha (0 = transparent, 1 = opaque)  
} sColor;
```

u8 ColorR(const sColor* col);

col ... color vector

Get red value as u8 (clamped to 0..255).

u8 ColorG(const sColor* col);

col ... color vector

Get green value as u8 (clamped to 0..255).

u8 ColorB(const sColor* col);

col ... color vector

Get blue value as u8 (clamped to 0..255).

ColorA(const sColor* col);

col ... color vector

Get alpha value as u8 (clamped to 0..255).

void ColorSet(sColor* col, u8 r, u8 g, u8 b, u8 a);

col ... color vector

r ... red component

g ... green component
b ... blue component
a ... alpha component

Set color values from u8.

void ColorBlack(sColor* col);

col ... color vector

Set color to black.

void ColorWhite(sColor* col);

col ... color vector

Set color to white.

void ColorGrey(sColor* col, color_float val);

col ... color vector

val ... grey value 0 to 1

Set grey value.

void ColorClamp(sColor* col);

col ... color vector

Clamp color value to valid range 0..1.

void ColorAbs(sColor* col);

col ... color vector

Absolute value of color.

color_float ColorGetGrey(const sColor* col);
u8 ColorGetGreyByte(const sColor* col);

col ... color vector

Convert color to grey value.

color_float ColorGetLinGrey(const sColor* col);

col ... color vector

Get linear grey value (average value).

void ColorDecolor(sColor* col);

col ... color vector

Decolorize (set to average grey value).

```
void ColorTrans(sColor* dst, const sColor* src1, const sColor* src2, color_float k);
```

dst ... destination color vector
src1 ... source 1 color vector
src2 ... source 2 color vector
k ... transposition coefficient 0=src1 .. 1=src2

Transposition between 2 colors.

```
void ColorBlend(sColor* dst, const sColor* src);
```

dst ... destination color vector
src ... source color vector

Blend by src alpha (src->a: 0=dst,...1=src).

```
void ColorBlendInv(sColor* dst, const sColor* src);
```

dst ... destination color vector
src ... source color vector

Blend by dst alpha (dst->a: 0=src,...1=dst).

```
void ColorInv(sColor* col);
```

col ... color vector

Invert color.

```
void ColorNeg(sColor* col);
```

col ... color vector

Negate color.

```
void ColorAdd(sColor* col, color_float k);
```

col ... color vector
k ... constant

Add constant to color.

```
void ColorSum(sColor* dst, const sColor* src);
```

dst ... destination color vector
src ... source color vector

Sum two colors.

```
void ColorMul(sColor* col, color_float k);
```

col ... color vector
k ... constant

Multiply color by constant.

void ColorDiv(sColor* col, color_float k);

col ... color vector

k ... constant

Divide color by constant.

void ColorSqr(sColor* col);

col ... color vector

Square color.

void ColorSqrt(sColor* col);

col ... color vector

Square root color.

void ColorRange(sColor* col, color_float min, color_float max);

col ... color vector

min ... minimum of new interval

max ... maximum of new interval

Resize color interval from 0..1 to min..max.

void ColorAvg(sColor* col, const sColor* src);

col ... destination color vector

src ... source color vector

Average colors.

void ColorMin(sColor* col, const sColor* src);

col ... destination color vector

src ... source color vector

Minimize color.

void ColorMax(sColor* col, const sColor* src);

col ... destination color vector

src ... source color vector

Maximize color.

void ColorBright(sColor* col, color_float bright);

col ... color vector

bright ... required change of brightness

Adjust brightness (0.5 = no change, >0.5 rise, <0.5 lower brightness).

void ColorContrast(sColor* col, color_float contrast);

col ... color vector

contrast ... required change of contrast

Adjust contrast (0.5 = no change).

void ColorLevel(sColor* col, color_float bright, color_float contrast, color_float gamma);

col ... color vector

bright ... required change of brightness

contrast ... required change of contrast

gamma ... required change of gamma

Adjust brightness level (controls are in range 0..1, 0.5=linear).

u32 ColorRGB(const sColor* col);

col ... color vector

Convert color to u32 RGB.

u32 ColorRGBA(const sColor* col);

col ... color vector

Convert color to u32 RGBA.

u32 ColorBGR(const sColor* col);

col ... color vector

Convert color to u32 BGR.

u32 ColorBGRA(const sColor* col);

col ... color vector

Convert color to u32 BGRA.

u16 ColorRGB16(const sColor* col);

col ... color vector

Convert color to 16-bit RGB.

u16 ColorRGB15(const sColor* col);

col ... color vector

Convert color to 15-bit RGB.

void ColorRGBToHSL(sColor* col);

col ... color vector

Convert RGB to HSL.

void ColorHSLToRGB(sColor* col);

col ... color vector

Convert HSL to RGB.

void ColorRGBToHSV(sColor* col);

col ... color vector

Convert RGB to HSV.

void ColorHSVToRGB(sColor* col);

col ... color vector

Convert HSV to RGB.

3.6. Configuration of Device

Files: lib_config.h, lib_config.c

Config: USE_CONFIG (default 1)

The library sets the device configuration. It stores the configuration in the last 4 KB page of the Flash memory.

Configuration structure:

```
typedef struct {
    u16    crc;          // 0: checksum Crc16AFast (CRC-16 CCITT normal)
    u8     volume;       // 2: sound volume percentage 0..255
    u8     backlight;    // 3: display backlight 0..255 (default 255)
    u16    bat_full;     // 4: voltage of full battery in [mV]
    u16    bat_empty;    // 6: voltage of empty batter in [mV]
    u16    bat_diode;    // 8: voltage drop on diode in [mV]
    u16    adc_ref;      // 10: ADC reference voltage in [mV]
    float  temp_base;   // 12: temperature base voltage at 27°C
    float  temp_slope;  // 16: temperature slope - voltage per 1 degree
    u32    crystal;      // 20: crystal frequency in [Hz] (default 12000000)
    u8     stuffing[CONFIG_SIZE-24]; // 24: (8) alignment to page size
} sConfig;
extern sConfig Config;    // current configuration
```

u16 ConfigGetCRC(const sConfig* cfg);

cfg ... pointer to sConfig structure

Calculate CRC of the config structure.

void ConfigSetDef(sConfig* cfg);

cfg ... pointer to sConfig structure

Set configuration structure to default values.

void ConfigClear();

Clear configuration memory.

int ConfigLoad();

Load configuration from flash (set to default if not found; returns configuration index or -1 if not found). This function is automatically called during application startup from the internal RuntimelInit() function.

void ConfigSave();

Save configuration. If core 1 is running, lockout it or reset it! If VGA driver is running, stop it!

Volume setup

u8 ConfigGetVolume();

Get sound volume level (returns 0..CONFIG_VOLUME_MAX).

void ConfigSetVolume(int volume);

volume ... volume 0..CONFIG_VOLUME_MAX

Set sound volume level (limits range, updates playing sound, does not save configuration).

void ConfigIncVolume();

Increase sound volume level (limits range, updates playing sound, does not save configuration).

void ConfigDecVolume();

Decrease sound volume level (limits range, updates playing sound, does not save configuration).

Backlight setup

u8 ConfigGetBacklight();

Get backlight level (returns 0..CONFIG_BACKLIGHT_MAX).

void ConfigSetBacklight(int backlight);

backlight ... backlight 0..CONFIG_BACKLIGHT_MAX

Set backlight level (limits range, updated backlight, does not save configuration).

void ConfigIncBacklight();

Increase backlight (limits range, updated backlight, does not save configuration).

void ConfigDecBacklight();

Decrease backlight (limits range, updated backlight, does not save configuration).

Battery setup

u16 ConfigGetBatFullInt();

Get voltage of full battery in [mV].

float ConfigGetBatFull();

Get voltage of full battery in [V].

void ConfigSetBatFullInt(int mv);

mv ... voltage of full battery in [mV]

Set voltage of full battery in [mV] (limits range, does not save configuration).

void ConfigIncBatFull();

Increase voltage of full battery (limits range, does not save configuration).

void ConfigDecBatFull();

Decrease voltage of full battery (limits range, does not save configuration).

u16 ConfigGetBatEmptyInt();

Get voltage of empty battery in [mV].

float ConfigGetBatEmpty();

Get voltage of empty battery in [V].

void ConfigSetBatEmptyInt(int mv);

mv ... voltage of empty battery in [mV]

Set voltage of empty battery in [mV] (limits range, does not save configuration).

void ConfigIncBatEmpty();

Increase voltage of empty battery (limits range, does not save configuration).

void ConfigDecBatEmpty();

Decrease voltage of empty battery (limits range, does not save configuration).

u16 ConfigGetBatDiodeInt();

Get voltage drop on diode in [mV].

float ConfigGetBatDiode();

Get voltage drop on diode in [V].

void ConfigSetBatDiodeInt(int mv);

mv ... voltage drop on diode in [mV]

Set voltage drop on diode in [mV] (limits range, does not save configuration).

void ConfigIncBatDiode();

Increase voltage drop on diode (limits range, does not save configuration).

void ConfigDecBatDiode();

Decrease voltage drop on diode (limits range, does not save configuration).

ADC setup

u16 ConfigGetAdcRefInt();

Get ADC reference in [mV].

float ConfigGetAdcRef();

Get ADC reference in [V].

void ConfigSetAdcRefInt(int mv);

mv ... ADC reference in [mV]

Set ADC reference in [mV] (limits range, does not save configuration).

void ConfigIncAdcRef();

Increase ADC reference (limits range, does not save configuration).

void ConfigDecAdcRef();

Decrease ADC reference (limits range, does not save configuration).

Temperature setup

float ConfigGetTempBase();

Get temperature base voltage at 27°C.

void ConfigSetTempBase(float mv);

mv ... temperature base voltage at 27°C in [mV]

Set temperature base voltage at 27°C in [mV] (limits range, does not save configuration).

void ConfigIncTempBase();

Increase temperature base voltage at 27°C (limits range, does not save configuration).

void ConfigDecTempBase();

Decrease temperature base voltage at 27°C (limits range, does not save configuration).

float ConfigGetTempSlope();

Get temperature slope - voltage per 1 degree.

void ConfigSetTempSlope(float mv);

mv ... voltage per 1 degree in [mV]

Set temperature slope - voltage per 1 degree (limits range, does not save configuration).

void ConfigIncTempSlope();

Increase temperature slope - voltage per 1 degree (limits range, does not save configuration).

void ConfigDecTempSlope();

Decrease temperature slope - voltage per 1 degree (limits range, does not save configuration).

Crystal setup

u32 ConfigGetCrystal();

Get crystal in Hz.

void ConfigSetCrystal(u32 hz);

hz ... crystal frequency in [Hz]

Set crystal in [Hz] (limits range, does not save configuration).

void ConfigIncCrystal();

Increase crystal (limits range, does not save configuration).

void ConfigDecCrystal();

Decrease crystal (limits range, does not save configuration).

Screen Saver

Bool ConfigGetScreenSaver();

Get screensaver state.

void ConfigSetScreenSaver(Bool on);

Set screensaver state.

3.7. CRC - Cyclic Redundancy Check

Files: lib_crc.h, lib_crc.c

Config: USE_CRC (default 1)

Checksums are used to check the integrity of the data. In this module you can find CRC calculations for 8, 16, 32, 64, 128 or 256 bits, with or without table, with software or with DMA hardware support.

Check patterns

CRCPAT1LEN (9) length of CRC check pattern 1

CRCPAT2LEN (3) length of CRC check pattern 2 (or 4 on simple SUM)

CRCPAT3LEN (1024) length of CRC check pattern 3

const char CrcPat1[] = "123456789"; ... CRC check pattern 1

const u8 CrcPat2[] = { 0xFC, 0x05, 0x4A, 0x3E }; ... CRC check pattern 2

CrcPat3 ... Crc32BTable ... CRC check pattern 3

SHA256 hash (256-bit checksum)

SHA256 context

```
typedef struct {
    u64    datalen;           // size of input in bytes
    union {
        u32    state[8];      // current accumulation of the hash
        u8     result[32];    // hash result
    };
    u8     data[64];          // input buffer
} SHA256_Context;
```

void SHA256_Init(SHA256_Context* ctx);

ctx ... pointer to SHA256_Context structure

Initialize SHA256 context.

void SHA256_AddBuf(SHA256_Context* ctx, const void* buf, int len);

ctx ... pointer to SHA256_Context structure

buf ... buffer with data

len ... length of data in bytes

Add buffer to SHA256 accumulator.

void SHA256_AddByte(SHA256_Context* ctx, u8 data);

ctx ... pointer to SHA256_Context structure

data .. data byte

Add byte to SHA256 accumulator.

void SHA256_Final(SHA256_Context* ctx);

ctx ... pointer to SHA256_Context structure

Finalize. Input data will be aligned to 512-bit boundary, with end header. Hash result can be found in ctx->buffer.

void SHA256_Calc(u8* dst, const void* src, int len);

dst ... pointer to destination 32-byte buffer

buf ... buffer with data

len ... length of data in bytes

Calculace SHA256 hash. Input data will be aligned to 512-bit boundary, with end header. Calculation speed: 1580 us per 1 KB.

Bool SHA256_Check();

Check SHA256 hash calculations. Returns False on error.

MD5 hash (128-bit checksum)

MD5 context

typedef struct {

u64 size; *// size of input in bytes*

u32 buffer[4]; *// current accumulation of the hash, hash result*

union {

u8 input[64]; *// input buffer*

u32 input32[16];

}

} MD5_Context;

void MD5_Init(MD5_Context* ctx);

ctx ... pointer to MD5_Context structure

Initialize MD5 context.

void MD5_AddBuf(MD5_Context* ctx, const void* buf, int len);

ctx ... pointer to MD5_Context structure

buf ... buffer with data

len ... length of data in bytes

Add buffer to MD5 accumulator.

void MD5_AddByte(MD5_Context* ctx, u8 data);

ctx ... pointer to MD5_Context structure

data .. data byte

Add byte to MD5 accumulator.

void MD5_Final(MD5_Context* ctx);

ctx ... pointer to MD5_Context structure

Finalize. Input data will be aligned to 512-bit boundary, with end header. Hash result can be found in ctx->buffer.

void MD5_Calc(u8* dst, const void* src, int len);

dst ... pointer to destination 16-byte buffer

buf ... buffer with data

len ... length of data in bytes

Calculace MD5 hash. Input data will be aligned to 512-bit boundary, with end header. Calculation speed: 864 us per 1 KB.

Bool MD5_Check(void);

Check MD5 hash calculations. Returns False on error.

CRC-64 Normal (CRC64A)

Polynom: $x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^4 + x + 1$

CRC-64-ECMA, ECMA-182

CRC-64 Normal (CRC64A, polynomial 0x42F0E1EBA9EA3693),

init word 0xFFFFFFFFFFFFFFFFF:

Sample "123456789" -> 0x9D13A61C0E5B0FF5ull

Sample 0xFC 0x05 0x4A -> 0x21E4F88DB2978548ull

Sample CRC32BTable 1KB -> 0x378302BE2C61CF9Eull

const u64 Crc64ATable[256]; ... CRC-64 Normal (CRC64A) table (2 KB)

CRC64A_POLY ... 0x42F0E1EBA9EA3693 ... CRC-64 Normal (CRC64A) polynomial

CRC64A_INIT ... 0xFFFFFFFFFFFFFFFFF ... // CRC-64 Normal (CRC64A) init word

Bool Crc64ATableCheck();

Check CRC-64 Normal (CRC64A) table. Returns False on error.

u64 Crc64AByteTab(u64 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-64 Normal (CRC64A), 1 byte - tabled version (requires 2 KB of flash memory).

u64 Crc64AByteSlow(u64 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-64 Normal (CRC64A), 1 byte - slow version.

u64 Crc64ABufTab(u64 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-64 Normal (CRC64A), buffer - tabled version (requires 2 KB of flash memory).

u64 Crc64ABufSlow(u64 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-64 Normal (CRC64A), buffer - slow version.

u64 Crc64ATab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-64 Normal (CRC64A) - tabled version (requires 2 KB of flash memory). Calculation speed: 200 us per 1 KB.

u64 Crc64ASlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-64 Normal (CRC64A) - slow version. Calculation speed: 1000 us per 1 KB.

Bool Crc64ACheck(void);

Check CRC-64 Normal (CRC64A) calculations. Returns False on error.

CRC-64 Reversed (CRC64B)

CRC-64-ECMA, ECMA-182

CRC-64 Reversed (CRC64B, polynomial 0xC96C5795D7870F42), init word 0:

Sample "123456789" -> 0x995DC9BBDF1939FAull

Sample 0xFC 0x05 0x4A -> 0xF10B3B2CDEF45CFAull

Sample CRC32BTable 1KB -> 0xACB732293EDC5DC8ull

This is preferred CRC-64 method.

const u64 Crc64BTable[256]; ... CRC-64 Reversed (CRC64B) table (2 KB)

CRC64B_POLY ... 0xC96C5795D7870F42ULL ... CRC-64 Reversed (CRC64B) polynomial

CRC64B_INIT ... 0 ... CRC-64 Reversed (CRC64B) init word (not reversed, not inverted)

Bool Crc64BTableCheck();

Check CRC-64 Reversed (CRC64B) table. Returns False on error.

u64 Crc64BByteTab(u64 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-64 Reversed (CRC64B), 1 byte - tabled version (requires 2 KB of flash memory).

u64 Crc64BByteSlow(u64 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-64 Reversed (CRC64B), 1 byte - slow version.

u64 Crc64BBufTab(u64 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-64 Reversed (CRC64B), buffer - tabled version (requires 2 KB of flash memory).

u64 Crc64BBufSlow(u64 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-64 Reversed (CRC64B), buffer - slow version.

u64 Crc64BTab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-64 Reversed (CRC64B) - tabled version (requires 2 KB of flash memory). Calculation speed: 220 us per 1 KB.

u64 Crc64BSlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-64 Reversed (CRC64B) - slow version. Calculation speed: 2200 us per 1 KB.

Bool Crc64BCheck(void);

Check CRC-64 Reversed (CRC64B) calculations. Returns False on error.

CRC-64 Default

Default CRC-64 functions: CRC-64 Reversed CRC64B.

Sample "123456789" -> 0x995DC9BBDF1939FAull

Sample 0xFC 0x05 0x4A -> 0xF10B3B2CDEF45CFAull

Sample CRC32BTable 1KB -> 0xACB732293EDC5DC8ull

CRC64_INIT ... CRC64B_INIT ... CRC-64 init word

u64 Crc64Byte(u64 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-64, 1 byte.

u64 Crc64Buf(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-64, buffer.

u64 Crc64(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-64. Requires 2 KB of flash memory. Calculation speed: 220 us per 1 KB.

CRC-32 Normal (CRC32A)

Polynom: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

in BIN: 0b0000 0100 1100 0001 0001 1101 1011 0111 (x32 is omitted, CRC is 32-bit long)

in HEX: 0x04C11DB7

reversed HEX: 0xEDB88320

CRC-32 is most commonly used in 2 variants - normal (CRC32A) and reversed (CRC32B). Normal variant (CRC32A, older ITU I.363.5 algorithm) is used as checksum of boot loader stage 2 in Raspberry Pico, MPEG-2 and BZIP2 (BZIP2 with inverted result). Reversed variant (CRC32B, later ITU V.42 algorithm) us used in Ethernet, FDDI, Gzip and PKZip. It is recommended to prefer reversed variant CRC32B, which is more commonly used.

CRC-32 Normal (CRC32A, polynomial 0x04C11DB7), init word 0xFFFFFFFF:

Sample "123456789" -> 0x0376E6E7

Sample 0xFC 0x05 0xA -> 0x8E10D720

Sample CRC32BTable 1KB -> 0x5796333D

const u32 Crc32ATable[256]; ... CRC-32 Normal (CRC32A) table (1 KB)

CRC32A_POLY ... 0x04C11DB7 ... CRC-32 Normal (CRC32A) polynomial

CRC32A_INIT ... 0xFFFFFFFF ... CRC-32 Normal (CRC32A) init word

Bool Crc32ATableCheck();

Check CRC-32 Normal (CRC32A) table (returns False on error).

u32 Crc32AByteTab(u32 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-32 Normal (CRC32A), 1 byte - tabled version (requires 1 KB of memory).

u32 Crc32AByteSlow(u32 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-32 Normal (CRC32A), 1 byte - slow software version.

u32 Crc32AByteDMA(u32 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-32 Normal (CRC32A), 1 byte - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc32ABufTab(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A), buffer - tabled version (requires 1 KB of memory)

u32 Crc32ABufSlow(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A), buffer - slow software version.

u32 Crc32ABufDMA(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A), buffer - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc32ATab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A) - tabled version (requires 1 KB of flash memory). Calculation speed: 160 us per 1 KB.

u32 Crc32ASlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A) - slow software version. Calculation speed: 790 us per 1 KB.

u32 Crc32ADMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Normal (CRC32A) - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

Bool Crc32ACheck();

Check CRC-32 Normal (CRC32A) calculations (returns False on error).

CRC-32 Reversed (CRC32B)

CRC-32 Reversed (CRC32B, polynomial 0xEDB88320), init word 0xFFFFFFFF (we use non-inverted form 0):

Sample "123456789" -> 0xCB43926

Sample 0xFC 0x05 0x4A -> 0xA8E10F6D

Sample CRC32BTable 1KB -> 0x6FCF9E13

```
const u32 Crc32BTable[256]; ... CRC-32 Reversed (CRC32B) table (1 KB)
CRC32B_POLY ... 0xEDB88320 ... CRC-32 Reversed (CRC32B) polynomial
CRC32B_INIT ... 0 ... CRC-32 Reversed (CRC32B) init word
- This initialization word is commonly given in its inverted form, as 0xFFFFFFFF.
```

Bool Crc32BTableCheck();

Check CRC-32 Reversed (CRC32B) table (returns False on error).

u32 Crc32BByteTab(u32 crc, u8 data);

crc ... input CRC
data ... data byte

Calculate CRC-32 Reversed (CRC32B), 1 byte - tabled version (requires 1 KB of memory).

u32 Crc32BByteSlow(u32 crc, u8 data);

crc ... input CRC
data ... data byte

Calculate CRC-32 Reversed (CRC32B), 1 byte - slow software version.

u32 Crc32BByteDMA(u32 crc, u8 data);

crc ... input CRC
data ... data byte

Calculate CRC-32 Reversed (CRC32B), 1 byte - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc32BBufTab(u32 crc, const void* buf, int len);

crc ... input CRC
buf ... pointer to data
len ... length of data

Calculate CRC-32 Reversed (CRC32B), buffer - tabled version (requires 1 KB of memory).

u32 Crc32BBufSlow(u32 crc, const void* buf, int len);

crc ... input CRC
buf ... pointer to data
len ... length of data

Calculate CRC-32 Reversed (CRC32B), buffer - slow software version.

u32 Crc32BBufDMA(u32 crc, const void* buf, int len);

crc ... input CRC
buf ... pointer to data

len ... length of data

Calculate CRC-32 Reversed (CRC32B), buffer - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc32BTab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Reversed (CRC32B) - tabled version (requires 1 KB of flash memory). Calculation speed: 160 us per 1 KB.

u32 Crc32BSlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Reversed (CRC32B) - slow software version. Calculation speed: 900 us per 1 KB.

u32 Crc32BDMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 Reversed (CRC32B) - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

Bool Crc32BCheck();

Check CRC-32 Reversed (CRC32B) calculations (returns False on error).

CRC-32 Default

Default CRC-32 functions: CRC-32 Reversed CRC32B with DMA preferred

Sample "123456789" -> 0xCB43926

Sample 0xFC 0x05 0x4A -> 0xA8E10F6D

Sample CRC32BTable 1KB -> 0x6FCF9E13

CRC32_INIT ... 0 ... CRC-32 init word

u32 Crc32Byte(u32 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-32, 1 byte.

u32 Crc32Buf(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-32, buffer (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc32(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-32 (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

CRC-16 CCITT Normal (CRC16A)

Polynom: $x^{16} + x^{12} + x^5 + 1$

in BIN: 0b0001 0000 0010 0001 (x16 is omitted, CRC is 16-bit long)

in HEX: 0x1021

reversed HEX: 0x8408

CRC-16 CCITT is used in X.25, V.41, HDLC, XMODEM, BlueTooth, SD card and many others. It is recommended to prefer reversed variant CRC16C, which is more commonly used.

CRC-16 CCITT Normal (CRC16A, polynomial 0x1021), init word 0xFFFF:

Sample "123456789" -> 0x29B1

Sample: 0xFC 0x05 0x4A -> 0x4CD4

Sample CRC32 1KB -> 0x4EED

const u16 Crc16ATable[256]; ... CRC-16 CCITT Normal (CRC16A) table (512 B)

CRC16A_POLY ... 0x1021 ... CRC-16 CCITT Normal (CRC16A) polynomial

CRC16A_INIT ... 0xFFFF ... CRC-16 CCITT Normal (CRC16A) init word

Bool Crc16ATableCheck();

Check CRC-16 CCITT Normal (CRC16A) table (returns False on error).

u16 Crc16AByteTab(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal (CRC16A), 1 byte - tabled version (requires 512 B).

u16 Crc16AByteFast(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal (CRC16A), 1 byte - fast software version.

u16 Crc16AByteSlow(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal (CRC16A), 1 byte - slow software version.

u16 Crc16AByteDMA(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal (CRC16A), 1 byte - DMA version. Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16ABufTab(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A), buffer - tabled version (requires 512 B).

u16 Crc16ABufFast(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A), buffer - fast software version.

u16 Crc16ABufSlow(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A), buffer - slow software version.

u16 Crc16ABufDMA(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A), buffer - DMA version. Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16ATab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A) - tabled version (requires 512 B of flash memory). Calculation speed: 170 us per 1 KB.

u16 Crc16AFast(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A) - fast software version. Calculation speed: 200 us per 1 KB.

u16 Crc16ASlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal (CRC16A) - slow software version. Calculation speed: 1000 us per 1 KB.

u16 Crc16ADMA(const void* buf, int len);

Calculate CRC-16 CCITT Normal (CRC16A) - DMA version (uses DMA_TEMP_CHAN() channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

Bool Crc16ACheck();

Check CRC-16 CCITT Normal (CRC16A) calculations (returns False on error).

CRC-16 CCITT Reversed (CRC16B)

CRC-16 CCITT Reversed (CRC16B, polynomial 0x8408), init word 0xFFFF (we use non-inverted form 0):

Sample "123456789" -> 0x906E

Sample: 0xFC 0x05 0x4A -> 0x7CBD

Sample CRC32 1KB -> 0x0C42

const u16 Crc16BTable[256]; ... CRC-16 CCITT Reversed (CRC16B) table (512 B)

CRC16B_POLY ... 0x8408 ... CRC-16 CCITT Reversed (CRC16B) polynomial

CRC16B_INIT ... 0 ... CRC-16 CCITT Reversed (CRC16B) init word

- This initialization word is commonly given in its inverted form, as 0xFFFF.

Bool Crc16BTableCheck();

Check CRC-16 CCITT Reversed (CRC16B) table (returns False on error).

u16 Crc16BByteTab(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Reversed (CRC16B), 1 byte - tabled version (requires 512 B).

u16 Crc16BByteSlow(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Reversed (CRC16B), 1 byte - slow software version.

u16 Crc16BByteDMA(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Reversed (CRC16B), 1 byte - DMA version. Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16BBufTab(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B), buffer - tabled version (requires 512 B).

u16 Crc16BBufSlow(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B), buffer - slow software version.

u16 Crc16BBufDMA(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B), buffer - DMA version. Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16BTab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B) - tabled version (requires 512 B of flash memory). Calculation speed: 160 us per 1 KB.

u16 Crc16BSlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B) - slow software version. Calculation speed: 880 us per 1 KB.

u16 Crc16BDMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Reversed (CRC16B) - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously in both CPUs, but not simultaneously in an interrupt. Calculation speed: 2 us per 1 KB.

Bool Crc16BCheck();

Check CRC-16 CCITT Reversed (CRC16B) calculations (returns False on error).

CRC-16 CCITT Normal Alternative (CRC16C)

CRC-16-CCITT Normal Alternative (CRC16C, polynomial 0x1021), init word 0x1D0F:

Sample "123456789" -> 0xE5CC

Sample: 0xFC 0x05 0x4A -> 0x9144

Sample CRC32 1KB -> 0xE351

CRC16C_INIT ... 0x1D0F ... CRC-16 CCITT Normal Alternative (CRC16C) init word

u16 Crc16CByteTab(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal Alternative (CRC16C), 1 byte - tabled version (requires 512 B of flash memory).

u16 Crc16CByteFast(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal Alternative (CRC16C), 1 byte - fast software version.

u16 Crc16CByteSlow(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal Alternative (CRC16C), 1 byte - slow software version.

u16 Crc16CByteDMA(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16 CCITT Normal Alternative (CRC16C), 1 byte - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16CBufTab(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C), buffer - tabled version (requires 512 B of flash memory).

u16 Crc16CBufFast(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C), buffer - fast software version.

u16 Crc16CBufSlow(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C), buffer - slow software version.

u16 Crc16CBufDMA(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C), buffer - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u16 Crc16CTab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C) - tabled version (requires 512 B of flash memory). Calculation speed: 170 us per 1 KB.

u16 Crc16CFast(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C) - fast software version. Calculation speed: 200 us per 1 KB.

u16 Crc16CSlow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C) - slow software version. Calculation speed: 1000 us per 1 KB.

u16 Crc16CDMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 CCITT Normal Alternative (CRC16C) - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

Bool Crc16CCheck();

Check CRC-16 CCITT Normal Alternative (CRC16C) calculations (returns False on error).

CRC-16 Default

Default CRC-16: CRC-16 CCITT Normal Alternative CRC16C with DMA preferred

Sample "123456789" -> 0xE5CC

Sample: 0xFC 0x05 0x4A -> 0x9144

Sample CRC32 1KB -> 0xE351

CRC16_INIT ... 0x1D0F ... CRC-16 init word

u32 Crc16Byte(u32 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-16, 1 byte.

u32 Crc16Buf(u32 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-16, buffer (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Crc16(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-16 (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

CRC-8 Dallas

Polynom: $x^8 + x^5 + x^4 + 1$

in BIN: 0b0011 0001 (x8 is omitted, CRC is 8-bit long)

in HEX: 0x31

reversed HEX: 0x8C

Used with Dallas Maxim 1-Wire bus.

CRC-8 Dallas, init word 0:

Sample "123456789" -> 0xA1

Sample: 0xFC 0x05 0x4A -> 0xF1

Sample CRC32 1KB -> 0x1E

const u8 Crc8Table[256]; ... CRC-8 table (256 B)

CRC8_POLY ... 0x8C ... CRC-8 polynomial

CRC8_INIT ... 0 ... CRC-8 init word

Bool Crc8TableCheck();

Check CRC-8 table (returns False on error).

u8 Crc8ByteTab(u8 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-8, 1 byte - tabled version (requires 256 B of flash memory).

u8 Crc8ByteSlow(u8 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-8, 1 byte - slow software version.

u8 Crc8BufTab(u8 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-8, buffer - tabled version (requires 256 B of flash memory).

u8 Crc8BufSlow(u8 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-8, buffer - slow software version.

u8 Crc8Tab(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-8 - tabled version (requires 256 B of flash memory). Calculation speed: 115 us per 1 KB.

u8 Crc8Slow(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-8 - slow software version. Calculation speed: 820 us per 1 KB.

Bool Crc8Check();

Check CRC-8 calculations (returns False on error).

CRC-8 Default

Default uses tabled versions.

u8 Crc8Byte(u8 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-8, 1 byte.

u8 Crc8Buf(u8 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-8, buffer.

u8 Crc8(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-8. Calculation speed: 115 us per 1 KB.

Parity

Parity returns value 1 if number of bits "1" is odd. If number of bits is even, it returns 0.

Sample "123456789" -> 0x01

Sample: 0xFC 0x05 0x4A -> 0x01

Sample CRC32 1KB -> 0x00

u8 ParityByteSoft(u8 par, u8 data);

par ... input parity

data ... data byte

Calculate parity, 1 byte - software version.

u8 ParityByteDMA(u8 par, u8 data);

par ... input parity

data ... data byte

Calculate parity, 1 byte - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u8 ParityBufSoft(u8 par, const void* buf, int len);

par ... input parity

buf ... pointer to data

len ... length of data

Calculate parity, buffer - software version.

u8 ParityBufDMA(u8 par, const void* buf, int len);

par ... input parity

buf ... pointer to data

len ... length of data

Calculate parity, buffer - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u8 ParitySoft(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate parity - software version. Calculation speed: 90 us per 1 KB.

u8 ParityDMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate parity - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

Simple checksum of 8-bit data with 32-bit result

Sample "123456789" -> 0x000001DD

Sample: 0xFC 0x05 0x4A -> 0x0000014B

Sample CRC32 1KB -> 0x0001FE00

SUM8_INIT ... 0 ... 8-bit checksum init word

u32 Sum8ByteSoft(u32 sum, u8 data);

sum ... input sum

data ... data byte

Calculate 8-bit checksum, 1 byte - software version.

u32 Sum8ByteDMA(u32 sum, u8 data);

sum ... input sum

data ... data byte

Calculate 8-bit checksum, 1 byte - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum8BufSoft(u32 sum, const void* buf, int len);

sum ... input sum

buf ... pointer to data

len ... length of data

Calculate 8-bit checksum, buffer - software version.

u32 Sum8BufDMA(u32 sum, const void* buf, int len);

sum ... input sum

buf ... pointer to data

len ... length of data

Calculate 8-bit checksum, buffer - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum8Soft(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate 8-bit checksum - software version. Calculation speed: 100 us per 1 KB.

u32 Sum8DMA(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate 8-bit checksum - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 8 us per 1 KB.

Simple checksum of 16-bit data with 32-bit result

Sample "12345678" -> 0x0000D4D0

Sample 0xFC 0x05 0x4A 0x3E -> 0x00004446

Sample CRC32 1KB -> 0x00FFFF00

SUM16_INIT ... 0 ... 16-bit checksum init word

u32 Sum16WordSoft(u32 sum, u16 data);

sum ... input sum

data ... data word

Calculate 16-bit checksum, 1 word - software version.

u32 Sum16WordDMA(u32 sum, u16 data);

sum ... input sum

data ... data word

Calculate 16-bit checksum, 1 word - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum16BufSoft(u32 sum, const u16* buf, int num);

sum ... input sum

buf ... pointer to data (must be aligned to u16 entry)

num ... number of u16 entries

Calculate 16-bit checksum, buffer - software version.

u32 Sum16BufDMA(u32 sum, const u16* buf, int num);

sum ... input sum

buf ... pointer to data (must be aligned to u16 entry)

num ... number of u16 entries

Calculate 16-bit checksum, buffer - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum16Soft(const u16* buf, int num);

buf ... pointer to data (must be aligned to u16 entry)

num ... number of u16 entries

Calculate 16-bit checksum - software version. Calculation speed: 48 us per 1 KB.

u32 Sum16DMA(const u16* buf, int num);

buf ... pointer to data (must be aligned to u16 entry)

num ... number of u16 entries

Calculate 16-bit checksum - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 4 us per 1 KB.

Simple checksum of 32-bit data with 32-bit result

Sample "12345678" -> 0x6C6A6866

Sample 0xFC 0x05 0x4A 0x3E -> 0x3E4A05FC

Sample CRC32 1KB -> 0xFFFFFFF80

SUM32_INIT ... 0 ... 32-bit checksum init word

u32 Sum32DWordSoft(u32 sum, u32 data);

sum ... input sum

data ... data double word

Calculate 32-bit checksum, 1 double word - software version.

u32 Sum32DWordDMA(u32 sum, u32 data);

sum ... input sum

data ... data double word

Calculate 32-bit checksum, 1 double word - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum32BufSoft(u32 sum, const u32* buf, int num);

sum ... input sum

buf ... pointer to data (must be aligned to u32 entry)

num ... number of u32 entries

Calculate 32-bit checksum, buffer - software version.

u32 Sum32BufDMA(u32 sum, const u32* buf, int num);

sum ... input sum

buf ... pointer to data (must be aligned to u32 entry)

num ... number of u32 entries

Calculate 32-bit checksum, buffer - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt.

u32 Sum32Soft(const u32* buf, int num);

buf ... pointer to data (must be aligned to u32 entry)

num ... number of u32 entries

Calculate 32-bit checksum - software version. Calculation speed: 22 us per 1 KB.

u32 Sum32DMA(const u32* buf, int num);

buf ... pointer to data (must be aligned to u32 entry)

num ... number of u32 entries

Calculate 32-bit checksum - DMA version (uses DMA_TEMP_CHAN() temporary channel). Can be used simultaneously on both CPUs, but not simultaneously with interrupt. Calculation speed: 2 us per 1 KB.

CRC-XOR

XOR is simple fast method, but better than simple sum. It was used to check EEPROMs in vintage Robotron 8-bit computers.

Sample "123456789" -> 0x2035

Sample: 0xFC 0x05 0x4A -> 0x03B0

Sample CRC32 1KB -> 0x0000

CRCXOR_INIT ... 0 ... CRC-XOR init word

u16 CrcXorByte(u16 crc, u8 data);

crc ... input CRC

data ... data byte

Calculate CRC-XOR, 1 byte.

u16 CrcXorBuf(u16 crc, const void* buf, int len);

crc ... input CRC

buf ... pointer to data

len ... length of data

Calculate CRC-XOR, buffer.

u16 CrcXor(const void* buf, int len);

buf ... pointer to data

len ... length of data

Calculate CRC-XOR. Calculation speed: 160 us per 1 KB.

3.8. DecNum - Decode Numbers

Files: lib_decnum.h, lib_decnum.c

Config: USE_DECNUM (default 1)

DecNum is a small library to decode integer numbers into text buffer.

DECNUMBUF_SIZE ... 16 ... max. size of decode text buffer

char DecNumBuf[DECNUMBUF_SIZE]; ... temporary decode text buffer

Library does not use DecNumBuf buffer. It can be used by programmer as destination buffer to decode numbers.

int DecUNum(char* buf, u32 num, char sep);

buf ... destination buffer

num ... number to decode

sep ... character of thousand separator, or 0 if not used

Decode unsigned number into ASCIIZ text buffer. Returns number of digits.

int DecNum(char* buf, s32 num, char sep);

buf ... destination buffer

num ... number to decode

sep ... character of thousand separator, or 0 if not used

Decode signed number into ASCIIZ text buffer. Returns number of digits.

void DecHexNum(char* buf, u32 num, u8 dig);

buf ... destination buffer

num ... number to decode

dig ... count of digits

Decode hex number into ASCIIZ text buffer.

void Dec2Dig(char* buf, u8 num);

buf ... destination buffer

num ... number to decode

Decode 2 digits of number into ASCIIZ text buffer.

void Dec2DigSpc(char* buf, u8 num);

buf ... destination buffer

num ... number to decode

Decode 2 digits of number with space character into ASCIIZ text buffer.

3.9. Draw - Drawing to Display Frame Buffer

Files: lib_draw.h, lib_draw.c

Config: USE_DRAW (default 1 if use devices with screen)

The Draw library is used to output graphics to frame buffer of TFT or VGA display in 4, 8, 15 or 16-bit output. Drawing is done into the FrameBuf[] frame buffer or into BackBuf[] back buffer. During drawing, the boundaries of the modified zone are updated. After the drawing is finished, the modified area of the frame buffer is transferred to active screen by the DispUpdate() function. The library can also be used to write to the VGA buffer - to prevent image flicker, a strip back buffer is used.

When outputting to the TFT display, only drawing to the frame buffer is used, the back buffer is not used. Drawing operations update the boundaries of the changed area. When drawing is complete, the DispUpdate() function must be called to transfer the changed areas from the frame buffer to the TFT display.

When outputting to the VGA display, drawing is done either to the frame buffer or to the back buffer, depending on the setting of the USE_MINIVGA switch. If drawing is done directly to the frame buffer, the image is immediately visible on the display. With some graphics output methods, this may result in the contents of the display flickering during drawing. In the simplest case, this can be prevented by calling the VgaWaitVSync() function. If this is not enough, it is necessary to draw to the back buffer first and transfer the image to the frame buffer with the DispUpdate() function after the operations are complete. If there is not enough memory to create full back buffer, a truncated strip back buffer can be used. The image is rendered in strips, using the DispSetStripNext() and DispUpdate() functions.

Color formats are determined by the COLBITS setup:

COLBITS = 4 ... YRGB1111 (color format known from PC-CGA or EGA cards)

COLBITS = 8 ... RGB332

COLBITS = 15 ... RGB555

COLBITS = 16 ... RGB565

Configuration

COLBITS	number of color bits: 4, 8, 15 or 16
COLTYPE	type of color entry: u8 or u16
FRAMETYPE	type of frame buffer entry: u8 or u16
WIDTH	display width in number of pixels
WIDTHLEN	display width in number of frame elements
HEIGHT	display height
FRAMESIZE	frame buffer size in number of frame elements
BACKBUFSIZE	back buffer size in number of frame elements
FONT	default system font
FONTW	width of characters of default system font

FONTHEIGHT	height of characters of default system font
TEXTWIDTH	width of text buffer
TEXTHEIGHT	height of text buffer
TEXTSIZE	size of mono text buffer
FTEXTSIZE	size of text buffer with foreground color
COLOR(r,g,b)	macro to get color from color components

FRAMETYPE FrameBuf[**FRAMESIZE**]; ... frame buffer

FRAMETYPE BackBuf[**BACKBUFSIZE**]; ... back buffer

Pre-defined colors

- base colors

COL_BLACK

COL_BLUE

COL_GREEN

COL_CYAN

COL_RED

COL_MAGENTA

COL_YELLOW

COL_WHITE

COL_GRAY

- dark colors

COL_DKBLUE

COL_DKGREEN

COL_DKCYAN

COL_DKRED

COL_DKMAGENTA

COL_DKYELLOW

COL_DKWHITE

COL_DKGRAY

- light colors

COL_LTBLUE

COL_LTGREEN

COL_LTCYAN

COL_LTRED

COL_LTMAGENTA

COL_LTYELLOW

COL_LTGRAY

COL_AZURE

COL_ORANGE

- default console print color

COL_PRINT_DEF COL_GRAY

void SelFont5x8();

Select font of size 5x8 pixels.

void SelFont6x8();

Select font of size 6x8 pixels.

void SelFont8x8();

Select font of size 8x8 pixels.

void SelFont8x14();

Select font of size 8x14 pixels.

void SelFont8x16();

Select font of size 8x14 pixels.

void DrawRect(int x, int y, int w, int h, COLTYPE col);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

col ... color

Draw rectangle.

void DrawRectInv(int x, int y, int w, int h);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

Draw rectangle inverted.

void DrawFrame(int x, int y, int w, int h, COLTYPE col);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

col ... color

Draw frame 1 pixel thin.

void DrawFrameW(int x, int y, int w, int h, COLTYPE col, int width);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

col ... color

width ... line width in pixels

Draw frame with specified line width.

void DrawFrameInv(int x, int y, int w, int h);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

Draw inverted frame 1 pixel thin.

void DrawFrameInvW(int x, int y, int w, int h, int width);

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

width ... line width in pixels

Draw inverted frame with specified line width.

void DrawClearCol(COLTYPE col);

col ... color

Clear canvas with color.

void DrawClear();

Clear canvas with black color.

void DrawPoint(int x, int y, COLTYPE col);

x ... X coordinate

y ... Y coordinate

col ... color

Draw point.

void DrawPointInv(int x, int y);

x ... X coordinate

y ... Y coordinate

Draw point inverted.

COLTYPE DrawGetPoint(int x, int y);

x ... X coordinate

y ... Y coordinate

Get point from frame buffer. Returns black color if out on range. Only 8-bit or more color format.

void DrawLineClip(int x1, int y1, int x2, int y2, COLTYPE col, int xmin, int xmax, int ymin, int ymax);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

col ... color

xmin ... minimal X

xmax ... maximal X + 1

ymin ... minimal Y

ymax ... maximal Y + 1

Draw line clipped.

void DrawLine(int x1, int y1, int x2, int y2, COLTYPE col);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

col ... color

Draw line.

void DrawLineW(int x1, int y1, int x2, int y2, COLTYPE col, int w, Bool round);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

col ... color

w ... line width in pixels

round ... draw round ends

Draw line with specified width.

void DrawLineInvClip(int x1, int y1, int x2, int y2, int xmin, int xmax, int ymin, int ymax);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

xmin ... minimal X

xmax ... maximal X + 1

ymin ... minimal Y

ymax ... maximal Y + 1

Draw inverted line clipped.

void DrawLineInv(int x1, int y1, int x2, int y2);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

Draw inverted line.

void DrawLineInvW(int x1, int y1, int x2, int y2, int w, Bool round);

x1 ... X1 coordinate

y1 ... Y1 coordinate

x2 ... X2 coordinate

y2 ... Y2 coordinate

w ... line width in pixels

round ... draw round ends

Draw inverted line with specified width.

void DrawFillCircle(int x0, int y0, int r, COLTYPE col);

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

col .. color

Draw filled circle.

void DrawFillCircleInv(int x0, int y0, int r);

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

Draw inverted filled circle.

void DrawCircle(int x0, int y0, int r, COLTYPE col);

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

col .. color

Draw circle.

void DrawCircleInv(int x0, int y0, int r);

x0 ... X coordinate of circle center

y0 ... Y coordinate of circle center

r ... circle radius

Draw inverted circle.

void DrawRing(int x0, int y0, int rin, int rout, COLTYPE col);

x0 ... X coordinate of ring center

y0 ... Y coordinate of ring center

rin ... ring inner radius

rout ... ring outer radius

col .. color

Draw ring.

void DrawRingInv(int x0, int y0, int rin, int rout);

x0 ... X coordinate of ring center

y0 ... Y coordinate of ring center

rin ... ring inner radius

rout ... ring outer radius

Draw inverted ring.

void DrawFill(int x, int y, COLTYPE col)

x ... X coordinate

y ... Y coordinate

col ... color to fill

Fill area in FrameBuf with color.

void DrawChar(char ch, int x, int y, COLTYPE col);

ch ... character

x ... X coordinate

y ... Y coordinate

col ... color

Draw character with normal size and transparent background.

void DrawCharH(char ch, int x, int y, COLTYPE col);

ch ... character

x ... X coordinate

y ... Y coordinate

col ... color

Draw character with double height and transparent background.

void DrawCharW(char ch, int x, int y, COLTYPE col);

ch ... character

x ... X coordinate

y ... Y coordinate

col ... color

Draw character with double width and transparent background.

void DrawChar2(char ch, int x, int y, COLTYPE col);

ch ... character

x ... X coordinate

y ... Y coordinate

col ... color

Draw character with double size and transparent background.

void DrawCharBg(char ch, int x, int y, COLTYPE col, COLTYPE bgcol);

ch ... character

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw character with normal size and with background.

void DrawCharBgH(char ch, int x, int y, COLTYPE col, COLTYPE bgcol);

ch ... character

x ... X coordinate

y ... Y coordinate
col ... color
bgcol ... background color

Draw character with double height and with background.

void DrawCharBgW(char ch, int x, int y, COLTYPE col, COLTYPE bgcol);

ch ... character
x ... X coordinate
y ... Y coordinate
col ... color
bgcol ... background color

Draw character with double width and with background.

void DrawCharBg2(char ch, int x, int y, COLTYPE col, COLTYPE bgcol);

ch ... character
x ... X coordinate
y ... Y coordinate
col ... color
bgcol ... background color

Draw character with double size and with background.

void DrawCharBg4(char ch, int x, int y, COLTYPE col, COLTYPE bgcol);

ch ... character
x ... X coordinate
y ... Y coordinate
col ... color
bgcol ... background color

Draw character with quadruple size and with background.

void DrawText(const char* text, int x, int y, COLTYPE col);

text ... ASCIIZ text
x ... X coordinate
y ... Y coordinate
col ... color

Draw text with normal size and transparent background.

void DrawTextH(const char* text, int x, int y, COLTYPE col);

text ... ASCIIZ text
x ... X coordinate

y ... Y coordinate

col ... color

Draw text with double height and transparent background.

void DrawTextW(const char* text, int x, int y, COLTYPE col);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

Draw text with double width and transparent background.

void DrawText2(const char* text, int x, int y, COLTYPE col);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

Draw text with double size and transparent background.

void DrawTextBg(const char* text, int x, int y, COLTYPE col, COLTYPE bgcol);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw text with normal size and with background.

void DrawTextBgH(const char* text, int x, int y, COLTYPE col, COLTYPE bgcol);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw text with double height and with background.

void DrawTextBgW(const char* text, int x, int y, COLTYPE col, COLTYPE bgcol);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw text with double width and with background.

void DrawTextBg2(const char* text, int x, int y, COLTYPE col, COLTYPE bgcol);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw text with double size and with background.

void DrawTextBg4(const char* text, int x, int y, COLTYPE col, COLTYPE bgcol);

text ... ASCIIZ text

x ... X coordinate

y ... Y coordinate

col ... color

bgcol ... background color

Draw text with quadruple size and with background.

void DrawTextBuf(const char* textbuf, COLTYPE col, COLTYPE bgcol);

textbuf ... pointer to text buffer of size TEXTSIZE = TEXTWIDTH * TEXTHEIGHT

col ... color

bgcol ... background color

Draw text buffer. This function can be used if the program output is into text buffer of TEXTWIDTH character width and TEXTHEIGHT rows height, where each byte represents 1 ASCII character. The DrawTextBuf() function is used to redraw the entire text screen to the graphics screen. Function updates screen after drawing with DispUpdate() function.

void DrawFTTextBuf(const char* textbuf, COLTYPE bgcol);

textbuf ... pointer to text buffer with colors

bgcol ... background color

Draw text buffer with foreground color. This function is similar to the previous DrawTextBuf() function, except that each character byte is followed by 8-bit or 16-bit entry representing the character color. Function updates screen after drawing with DispUpdate() function.

void DrawImg(const COLTYPE* src, int xs, int ys, int xd, int yd, int w, int h, int ws);

src ... pointer to array of image data

xs ... source X coordinate

ys ... source Ycoordinate
xd ... destination X coordinate
yd ... destination Ycoordinate
w ... image width (or width of image cutout)
h ... image height (or height of image cutout)
ws ... full width of source image (in pixels)

Draw image in native default format, or draw part of image (cutout). The image format must match the current COLBITS color format.

void DrawGetImg(COLTYPE* dst, int x, int y, int w, int h)

dst ... pointer to destination buffer
x ... source X coordinate
y ... source Ycoordinate
w ... width of image cutout
h ... height of image cutout

Get image content from frame buffer. Remaining area fills black. Only 8-bit or more formats.

void DrawImgPal(const u8* src, const u16* pal, int xs, int ys, int xd, int yd, int w, int h, int ws);

src ... pointer to array of image data (1 pixel = 1 byte, 8-bit palettes color)
pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)
xs ... source X coordinate
ys ... source Y coordinate
xd ... destination X coordinate
yd ... destination Ycoordinate
w ... image width (or width of image cutout)
h ... image height (or height of image cutout)
ws ... full width of source image (in pixels)

Draw 8-bit palettes image, or draw part of image (cutout). Start of the cutout in the source image is set by calculating the start address of the source data. This function is only available when colour format COLBITS = 15 or 16.

void DrawImg4Pal(const u8* src, const u16* pal, int xs, int ys, int xd, int yd, int w, int h, int ws);

src ... pointer to array of image data (2 pixels = 1 byte, 4-bit palettes color)
pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)
xs ... source X coordinate
ys ... source Y coordinate
xd ... destination X coordinate
yd ... destination Ycoordinate

w ... image width (or width of image cutout)

h ... image height (or height of image cutout)

ws ... full width of source image (in pixels)

Draw 4-bit paletted image, or draw part of image (cutout). This function is only available when colour format COLBITS = 15 or 16.

void DrawImgRle(const u8* src, const u16* pal, int xd, int yd, int w, int h);

src ... pointer to array of image data (8-bit paletted color RLE)

pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)

xd ... destination X coordinate

yd ... destination Ycoordinate

w ... total image width

h ... total image height

Draw 8-bit paletted image with RLE compression. The image must be displayed whole, it cannot be cropped. This function is only available when colour format COLBITS = 15 or 16.

void DrawImgRle(const u8* src, int xd, int yd, int w, int h);

src ... pointer to array of image data (8-bit or 4-bit paletted color RLE)

xd ... destination X coordinate

yd ... destination Ycoordinate

w ... total image width

h ... total image height

Draw 8-bit or 4-bit paletted image with RLE compression. The image must be displayed whole, it cannot be cropped. This function is only available when colour format COLBITS = 4 or 8.

void DrawImg4Rle(const u8* src, const u16* pal, int xd, int yd, int w, int h);

src ... pointer to array of image data (4-bit paletted color RLE)

pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)

xd ... destination X coordinate

yd ... destination Ycoordinate

w ... total image width

h ... total image height

Draw 4-bit paletted image with RLE compression. The image must be displayed whole, it cannot be cropped. This function is only available when colour format COLBITS = 15 or 16.

void DrawBlit(const COLTYPE* src, int xs, int ys, int xd, int yd, int w, int h, int ws, COLTYPE col);

src ... pointer to array of image data

xs ... source X coordinate

ys ... source Y coordinate
xd ... destination X coordinate
yd ... destination Ycoordinate
w ... image width (or width of image cutout)
h ... image height (or height of image cutout)
ws ... full width of source image (in pixels)
col ... transparency key color

Draw image in default native format with transparency, or draw part of image (cutout). The image format must match the current COLBITS color format.

void DrawBlitSubst(const COLTYPE* src, int xs, int ys, int xd, int yd, int w, int h, int ws, COLTYPE col, COLTYPE fnd, COLTYPE subst)

src ... pointer to array of image data
xs ... source X
ys ... source Y
xd ... destination X
yd ... destination Y
w width
h height
ws ... source total width (in pixels)
col ... transparency key color
fnd ... color to find
subst ... replaced color

Draw image with transparency and color substitution. Only 8-bit or more color formats.

void DrawBlitPal(const u8* src, const u16* pal, int xs, int ys, int xd, int yd, int w, int h, int ws, COLTYPE col);

src ... pointer to array of image data (1 pixel = 1 byte, 8-bit palettes color)
pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)
xs ... source X coordinate
ys ... source Y coordinate
xd ... destination X coordinate
yd ... destination Ycoordinate
w ... image width (or width of image cutout)
h ... image height (or height of image cutout)
ws ... full width of source image (in pixels)
col ... transparency key color

Draw 8-bit palettes image with transparency, or draw part of image (cutout). This function is only available when colour format COLBITS = 15 or 16.

```
void DrawBlit4Pal(const u8* src, const u16* pal, int xs, int ys, int xd, int yd, int w, int h, int ws, COLTYPE col);
```

src ... pointer to array of image data (2 pixels = 1 byte, 4-bit palettes color)

pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)

xs ... source X coordinate

ys ... source Y coordinate

xd ... destination X coordinate

yd ... destination Y coordinate

w ... image width (or width of image cutout)

h ... image height (or height of image cutout)

ws ... full width of source image (in pixels)

col ... transparency key color

Draw 4-bit palettes image with transparency, or draw part of image (cutout). This function is only available when colour format COLBITS = 15 or 16.

```
void DrawRectShadow(int x, int y, int w, int h, u8 shad);
```

x ... X coordinate

y ... Y coordinate

w ... width

h ... height

shad ... shadow intensity 0=black to 15=light

Draw rectangle shadow. This function is only available when colour format COLBITS = 15 or 16.

```
void DrawBlitShadow(const u16* src, int xd, int yd, int w, int h, int ws, COLTYPE col, u8 shad);
```

src ... pointer to array of image data (1 pixel = 2 bytes, RGB565/RGB555 format)

xd ... destination X coordinate

yd ... destination Y coordinate

w ... image width (or width of image cutout)

h ... image height (or height of image cutout)

ws ... full width of source image (in pixels)

col ... transparency key color

shad ... shadow intensity 0=black to 15=light

Draw image in RGB565 format with transparency as shadow, or draw part of image (cutout). This function is only available when colour format COLBITS = 15 or 16.

```
void DrawBlit1Shadow(const u8* src, int xs, int ys, int xd, int yd, int w, int h, int ws, u8 shad);
```

src ... pointer to array of image data (8 pixels = 1 byte, 1-bit format)

xs ... source X coordinate
ys ... source Y coordinate
xd ... destination X coordinate
yd ... destination Y coordinate
w ... image width (or width of image cutout)
h ... image height (or height of image cutout)
ws ... full width of source image (in pixels)
shad ... shadow intenzity 0=black to 15=light

Draw 1-bit image with transparency as shadow. This function is only available when colour format COLBITS = 15 or 16.

```
void DrawImgMat(const COLTYPE* src, int ws, int hs, int x, int y, int w, int h,
const sMat2D* m, u8 mode, COLTYPE color);
```

src ... pointer to array of image data
ws ... source image width
hs ... source image height
x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
m ... transformation matrix
mode ... draw mode **DRAWIMG_***
color ... key or border color (**DRAWIMG_PERSP** mode: horizon offset)

Draw 16-bit image in default native format with 2D transformation matrix. Transformation matrix should be prepared using PrepDrawImg() function. Note to wrap and perspective mode: Width and height of source image should be power of 2, or it will render slower. This function is not available with colour format COLBITS = 4.

DrawImgMat mode

DRAWIMG_WRAP	wrap image
DRAWIMG_NOBORDER	no border (transparent border)
DRAWIMG_CLAMP	clamp image (use last pixel as border)
DRAWIMG_COLOR	color border
DRAWIMG_TRANSP	transparent image with key color
DRAWIMG_PERSP	perspective floor

```
void DrawImgPalMat(const u8* src, const u16* pal, int ws, int hs, int x, int y, int
w, int h, const sMat2D* m, u8 mode, COLTYPE color);
```

src ... pointer to array of image data (1 pixel = 1 byte, 8-bit palettes color)
pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)

ws ... source image width
hs ... source image height
x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
m ... transformation matrix
mode ... draw mode **DRAWIMG_***
color ... key or border color (**DRAWIMG_PERSP** mode: horizon offset)

Draw 8-bit palettes image with 2D transformation matrix. Transformation matrix should be prepared using PrepDrawImg() function. Note to wrap and perspective mode: Width and height of source image should be power of 2, or it will render slower. This function is only available when colour format COLBITS = 15 or 16.

```
void DrawImg4PalMat(const u8* src, const u16* pal, int ws, int hs, int x, int y,
int w, int h, const sMat2D* m, u8 mode, COLTYPE color);
```

src ... pointer to array of image data (2 pixels = 1 byte, 4-bit palettes color)
pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)
ws ... source image width
hs ... source image height
x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
m ... transformation matrix
mode ... draw mode **DRAWIMG_***
color ... key or border color (**DRAWIMG_PERSP** mode: horizon offset)

Draw 4-bit palettes image with 2D transformation matrix. Transformation matrix should be prepared using PrepDrawImg() function. Note to wrap and perspective mode: Width and height of source image should be power of 2, or it will render slower. This function is only available when colour format COLBITS = 15 or 16.

```
void DrawTileMap(const COLTYPE* src, const u8* map, int mapwbits, int
maphbits, int tilebits, int x, int y, int w, int h, const sMat2D* mat, u8 horizon);
```

src ... source image with column of square tiles
 (width = tile size, must be power of 2)
map ... byte map of tile indices
mapwbits ... number of bits of map width (number of tiles; width is power of 2)
maphbits ... number of bits of map height (number of tiles; height is power of 2)
tilebits ... number of bits of tile size (e.g. 5 = tile 32x32 pixel)

x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
mat ... transformation matrix
horizon ... horizon offset (0=do not use perspective projection)

Draw tile map using perspective projection. Transformation matrix should be prepared using PrepDrawImg() function. This function is not available with colour format COLBITS = 4.

```
void DrawTilePalMap(const u8* src, const u16* pal, const u8* map, int
mapwbits, int maphbits, int tilebits, int x, int y, int w, int h, const sMat2D*
mat, u8 horizon);
```

src ... source paletted image with column of 8-bit square tiles
 (width = tile size, must be power of 2)
pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)
map ... byte map of tile indices
mapwbits ... number of bits of map width (number of tiles; width is power of 2)
maphbits ... number of bits of map height (number of tiles; height is power of 2)
tilebits ... number of bits of tile size in pixels (e.g. 5 = tile 32x32 pixel)
x ... destination X coordinate
y ... destination Y coordinate
w ... destination width
h ... destination height
mat ... transformation matrix
horizon ... horizon offset (0=do not use perspective projection)

Draw tile map using perspective projection with 8-bit paletted image. Transformation matrix should be prepared using PrepDrawImg() function. This function is only available when colour format COLBITS = 15 or 16.

```
void DrawTile4PalMap(const u8* src, const u16* pal, const u8* map, int
mapwbits, int maphbits, int tilebits, int x, int y, int w, int h, const sMat2D*
mat, u8 horizon);
```

src ... source paletted image with column of 4-bit square tiles
 (width = tile size, must be power of 2)
pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)
map ... byte map of tile indices
mapwbits ... number of bits of map width (number of tiles; width is power of 2)
maphbits ... number of bits of map height (number of tiles; height is power of 2)
tilebits ... number of bits of tile size in pixels (e.g. 5 = tile 32x32 pixel)
x ... destination X coordinate

y ... destination Y coordinate
w ... destination width
h ... destination height
mat ... transformation matrix
horizon ... horizon offset (0=do not use perspective projection)

Draw tile map using perspective projection with 4-bit palettes image. Transformation matrix should be prepared using PrepDrawImg() function. This function is only available when colour format COLBITS = 15 or 16.

void DrawImgLine(const u16* src, int xd, int yd, int wd, int ws, int wbs);

src ... pointer to source image data
xd ... destination X coordinate
yd ... destination Y coordinate
wd ... destination width
ws ... source width
wbs ... source line pitch in bytes

Draw image line interpolated. This function is not available with colour format COLBITS = 4.

void DrawImgPalLine(const u8* src, const u16* pal, int xd, int yd, int wd, int ws, int wbs);

src ... pointer to array of image data (1 pixel = 1 byte, 8-bit palettes color)
pal ... pointer to palettes (array of max. 256 entries in RGB565/RGB555 format)
xd ... destination X coordinate
yd ... destination Y coordinate
wd ... destination width
ws ... source width
wbs ... source line pitch in bytes

Draw 8-bit palettes image line interpolated. This function is only available when colour format COLBITS = 15 or 16.

void DrawImg4PalLine(const u8* src, const u16* pal, int xd, int yd, int wd, int ws, int wbs);

src ... pointer to array of image data (2 pixels = 1 byte, 4-bit palettes color)
pal ... pointer to palettes (array of max. 16 entries in RGB565/RGB555 format)
xd ... destination X coordinate
yd ... destination Y coordinate
wd ... destination width
ws ... source width
wbs ... source line pitch in bytes

Draw 4-bit paletted image line interpolated. This function is only available when colour format COLBITS = 15 or 16.

void GenGrad(COLTYPE* dst, int w);

dst ... buffer to generate pixels of interpolated color gradient

w ... number of pixels to generate

Generate gradient. This function is not available with colour format COLBITS = 4.

u16 BlendCol16(u16 col1, u16 col2, u8 level);

col1 ... 1st color RGB565

col2 ... 2nd color RGB565

level ... blending level 0=col1 to 16=col2

Blend two 16-bit colors. This function is only available when colour format COLBITS = 16. Note: the method used is fast but not very accurate, the resulting color may sometimes be a bit darker due to the missing lowest bit of the color component.

void DrawScroll();

Scroll screen one row up.

void DrawScrollRect(int x, int y, int w, int h, int dy, COLTYPE col);

x ... X coordinate of rectangle

y ... Y coordinate of rectangle

w ... width of rectangle

h ... height of rectangle

dy ... number of lines to scroll up

col ... color to clear new space

Scroll rectangle up (in frame buffer, updates screen). This function is not available with colour format COLBITS = 4.

void DrawPrintCharRaw(char ch);

ch ... character to print

Console print character (without display update). Function uses control characters (table bellow) and scrolls display.

Control characters of console print functions

0x01	'\1'	^A	set not-inverted text
0x02	'\2'	^B	set inverted text (shift character code by 0x80)
0x03	'\3'	^C	use normal-sized font (default)
0x04	'\4'	^D	use double-height font
0x05	'\5'	^E	use double-width font

0x06	'\6'	^F	use double-sized font
0x07	'\a'	^G	move cursor 1 position right (no print; uses width of normal-sized font)
0x08	'\b'	^H	move cursor 1 position left (no print; uses width of normal-sized font)
0x09	'\t'	^I	(tabulator) move cursor to next 8-character position, print spaces (uses width of normal-sized font)
0x0A	'\n'	^J	(new line) move cursor to start of next row
0x0B	'\v'	^K	move cursor to start of previous row
0x0C	'\f'	^L	(form feed) clear screen, reset cursor position and set default color
0x0D	'\r'	^M	(carriage return) move cursor to start of current row
0x10	'\20'	^P	set gray text color (COL_GRAY , default)
0x11	'\21'	^Q	set blue text color (COL_AZURE)
0x12	'\22'	^R	set green text color (COL_GREEN)
0x13	'\23'	^S	set cyan text color (COL_CYAN)
0x14	'\24'	^T	set red text color (COL_RED)
0x15	'\25'	^U	set magenta text color (COL_MAGENTA)
0x16	'\26'	^V	set yellow text color (COL_YELLOW)
0x17	'\27'	^W	set white text color (COL_WHITE)

void DrawPrintChar(char ch);

ch ... character to print

Console print character, with display update (no need to call DispUpdate()). Function uses control characters (table above) and scrolls display.

void DrawPrintText(const char* txt);

txt ... ASCII text to print

Console print text, with display update (no need to call DispUpdate()). Function uses control characters (table above) and scrolls display.

u32 DrawPrintArg(const char* fmt, va_list args);

fmt ... format ASCII text string

args ... arguments

Formatted print string to drawing console, with argument list. Returns number of characters. Function updates display with DispUpdate().

u32 DrawPrint(const char* fmt, ...);

fmt ... format ASCII text string

... ... arguments

Formatted print string to drawing console, with variadic arguments. Returns number of characters. Function updates display with DispUpdate().

3.10. DrawCan - Drawing to Canvas

Files: lib_drawcan*.h, lib_drawcan*.c

Config: USE_DRAWCAN (default 0)

The DrawCan library is used to output graphics to the frame buffer in conjunction with the DispHSTX display driver. The library supports 9 frame buffer graphics formats, with color depths of 1, 2, 3, 4, 6, 8, 12, 15 and 16 bits per pixel. The library is too large for this manual to describe, so only a list of the most basic functions is given here. For a more detailed description of the library, see the DispHSTX display driver manual.

<code>void DrawSelFont8x8();</code>	Select font 8x8
<code>void DrawSelFont8x14();</code>	Select font 8x14
<code>void DrawSelFont8x16();</code>	Select font 8x16
<code>void DrawClearCol(u16 col);</code>	Clear canvas color
<code>void DrawClear();</code>	Clear canvas black
<code>void DrawPoint(int x, int y, u16 col);</code>	Draw point
<code>u16 DrawGetPoint(int x, int y);</code>	Get point
<code>void DrawRect(int x, int y, int w, int h, u16 col);</code>	Draw rectangle
<code>void DrawHLine(int x, int y, int w, u16 col);</code>	Draw horizontal line
<code>void DrawVLine(int x, int y, int h, u16 col);</code>	Draw vertical line
<code>void DrawFrame(int x, int y, int w, int h, u16 col_light, u16 col_dark);</code>	Draw 3D frame
<code>void DrawFrameW(int x, int y, int w, int h, u16 col, int thick);</code>	Draw thick frame
<code>void DrawLine(int x1, int y1, int x2, int y2, u16 col);</code>	Draw line
<code>void DrawLineW(int x1, int y1, int x2, int y2, u16 col, int thick, Bool round);</code>	Draw thick line
<code>void DrawRound(int x, int y, int d, u16 col, u8 mask);</code>	Draw round
<code>void DrawCircle(int x, int y, int d, u16 col, u8 mask);</code>	Draw circle or arc
<code>void DrawRing(int x, int y, int d, int din, u16 col, u8 mask);</code>	Draw ring
<code>void DrawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, u16 col);</code>	Draw triangle
<code>void DrawFill(int x, int y, u16 col);</code>	Draw fill area
<code>void DrawChar(char ch, int x, int y, u16 col, int scalex, int scaley);</code>	Draw character transparent
<code>void DrawCharBg(char ch, int x, int y, u16 col, u16 bgcol, int scalex, int scaley);</code>	Draw character with background
<code>void DrawText(const char* text, int len, int x, int y, u16 col, int scalex, int scaley);</code>	Draw text transparent
<code>void DrawTextBg(const char* text, int len, int x, int y, u16 col, u16 bgcol, int scalex, int scaley);</code>	Draw text with background
<code>void DrawEllipse(int x, int y, int dx, int dy, u16 col, u8 mask);</code>	Draw ellipse
<code>void DrawFillEllipse(int x, int y, int dx, int dy, u16 col, u8 mask);</code>	Draw filled ellipse
<code>void DrawImg(int xd, int yd, const void* src, int xs, int ys, int w, int h, int wbs);</code>	Draw image
<code>void DrawBlit(int xd, int yd, const void* src, int xs, int ys, int w, int h, int wbs, u16 col);</code>	Draw transparent image
<code>void DrawBlitSubs(int xd, int yd, const void* src, int xs, int ys, int w, int h, int wbs, u16 col, u16 fnd, u16 subs);</code>	Draw image with substitute color
<code>void DrawGetImg(int xs, int ys, int w, int h, void* dst, int xd, int yd, int wbd);</code>	Get image from canvas to buffer

3.11. Emu - Emulators

Files: `emu.h`, `emu.c`

Config: `USE_EMU (default 0)`

The Emu library contains emulators of processors and is used to emulate devices. To activate it, you must set the configuration parameter `USE_EMU` to 1 and activate the configuration parameter of the corresponding processor. The processor emulations take place in core 1 of the processor, allowing precise instruction timing, uninterrupted by any operations. A PWM generator is used to timing the emulation - any unused PWM generator can be used (output pins are not needed). For accurate timing, it is recommended to set the system clock before activating the emulator so that the desired frequency of the simulation clock is in integral proportion to the system clock. The `PWM_FindSysClk()` function can be used to find the appropriate system clock.

Supported processors are: Intel 4004, Intel 4040, Intel 8008, Intel 8048, Intel 8052, Intel 8080, Intel 8085, Intel 8086, Intel 8088, Intel 80186, Intel 80188, MOS-6502, MOS-65C02, Sharp X80 (LR35902), Zilog Z80.



I4004 CPU Emulator

Config: `USE_EMU_I4004 (default 0)`

Files `emu_i4004.*` contain emulator of the processor Intel 4004. To activate it, set configuration parameters `USE_EMU` and `USE_EMU_I4004` to value 1.

I4004_CLOCKMUL 4 ... clock multiplier (to achieve lower frequencies and finer timing)

I4004 CPU descriptor:

```
typedef struct {
```

```
    ... internal variables
```

```
    plI4004GetRom      readrom;           // read byte from ROM memory
    plI4004SetRom      writerom;          // write byte to ROM
    plI4004WritePort   writeramport;       // write nibble to RAM port
    plI4004WritePort   writeromport;        // write nibble to ROM port
```

```
    pl4004ReadPort    readromport;      // read nibble from ROM
    pl4004GetTest     gettest;          // get TEST signal
} sl4004;
```

void I4004_Reset(sl4004* cpu);

Reset processor.

u32 I4004_Start(sl4004* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I4004_CLOCKMUL is recommended to maintain.

void I4004_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I4004_Cont(sl4004* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I4004_IsRunning();

Check if emulation is running.

I4040 CPU Emulator

Config: USE_EMU_I4040 (default 0)

Files **emu_i4040.*** contain emulator of the processor Intel 4040. To activate it, set configuration parameters USE_EMU and USE_EMU_I4040 to value 1.

I4040_CLOCKMUL 4 ... clock multiplier (to achieve lower frequencies and finer timing)

I4040 CPU descriptor:

```
typedef struct {
```

```
    ... internal variables
```

```
    volatile u8        intreq;           // 1=interrupt request
```

```

    pl4040GetRom      readrom;           // read byte from ROM memory
    pl4040SetRom      writerom;          // write byte to ROM
    pl4040WritePort   writeramport;       // write nibble to RAM port
    pl4040WritePort   writeromport;       // write nibble to ROM port
    pl4040ReadPort    readromport;        // read nibble from ROM
    pl4040GetTest     gettest;           // get TEST signal
} sl4040;

```

void I4040_Reset(sl4040* cpu);

Reset processor.

u32 I4040_Start(sl4040* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I4040_CLOCKMUL is recommended to maintain.

void I4040_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I4040_Cont(sl4040* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I4040_IsRunning();

Check if emulation is running.

I8008 CPU Emulator

Config: USE_EMU_I8008 (default 0)

Files **emu_i8008.*** contain emulator of the processor Intel 8008. To activate it, set configuration parameters USE_EMU and USE_EMU_I8008 to value 1.

I8008_CLOCKMUL 8 ... clock multiplier (to achieve lower frequencies and finer timing)

I8008 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8          intreq; // 1=interrupt request, execute instruction RST n
    volatile u8          intins; // instruction RST n to execute during interrupt
    pEmu16Read8         readmem; // read byte from memory
    pEmu16Write8        writemem; // write byte to memory
    pEmu8Read8          readport; // read byte from port 0..7
    pEmu8Write8         writeport; // write byte to port 8..31
} sI8008;
```

void I8008_InitTab();

Initialize I8008 tables (call it on application start).

void I8008_Reset(sI8008* cpu);

Reset processor.

u32 I8008_Start(sI8008* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I8008_CLOCKMUL is recommended to maintain.

void I8008_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I8008_Cont(sI8008* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I8008_IsRunning();

Check if emulation is running.

I8048 CPU Emulator

Config: USE_EMU_I8048 (default 0)

Files `emu_i8048.*` contain emulator of the processor Intel 8048. To activate it, set configuration parameters USE_EMU and USE_EMU_I8048 to value 1.

I8048_CLOCKMUL 8 ... clock multiplier (to achieve lower frequencies and finer timing)

I8048 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8      intreq;      // 1=interrupt request
    pEmu16Read8     readrom;     // read byte from ROM memory
    pEmu8Read8      readext;     // read byte from extended data memory
    pEmu8Write8     writeext;    // write byte to extended data memory
    pEmu8Read8      readport;    // read byte from port 0..3
    pEmu8Write8     writeport;   // write byte to port 0..3
    pEmuWrite8      writeprog;   // output PROG signal (0 or 1)
    pEmu8Read8      readtest;    // read test signal 0 or 1 (returns 0 or 1)
} sl8048;
```

void I8048_Reset(sl8048* cpu);

Reset processor.

u32 I8048_Start(sl8048* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I8048_CLOCKMUL is recommended to maintain.

void I8048_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I8048_Cont(sl8048* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I8048_IsRunning();

Check if emulation is running.

I8052 CPU Emulator

Config: USE_EMU_I8052 (default 0)

Files **emu_i8052.*** contain emulator of the processor Intel 8052. To activate it, set configuration parameters **USE_EMU** and **USE_EMU_I8052** to value 1.

I8052_CLOCKMUL 1 ... clock multiplier (to achieve lower frequencies and finer timing)

I8052 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8      int0req;      // 1=external interrupt 0 request
    pEmu16Read8     readrom;     // read byte from ROM memory
    pEmu8Read8      readport;    // read byte from port 0..3
    pEmu8Write8     writeport;   // write byte to port 0..3
    pEmu16Write8    writeext;   // write byte to external memory
    pEmu16Read8     readext;    // read byte from external memory
} sl8052;
```

void I8052_InitTab();

Initialize I8052 tables (call it on application start).

void I8052_Reset(sl8052* cpu);

Reset processor.

u32 I8052_Start(sl8052* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To

achieve accurate timing, an integer ratio between system clock and clock frequency*I8052_CLOCKMUL is recommended to maintain.

void I8052_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I8052_Cont(sl8052* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I8052_IsRunning();

Check if emulation is running.

I8080 CPU Emulator

Config: USE_EMU_I8080 (default 0)

Files **emu_i8080.*** contain emulator of the processor Intel 8080. To activate it, set configuration parameters **USE_EMU** and **USE_EMU_I8080** to value 1.

I8080_CLOCKMUL 4 ... clock multiplier (to achieve lower frequencies and finer timing)

I8080 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8        intreq;      // 1=interrupt request, execute RST n
    volatile u8        intins;      // instruction RST n to execute at interrupt
    pEmu16Read8       readmem;    // read byte from memory
    pEmu16Write8      writemem;   // write byte to memory
    pEmu8Read8        readport;   // read byte from port
    pEmu8Write8       writeport;  // write byte to port
} sl8080;
```

void I8080_InitTab();

Initialize I8080 tables (call it on application start).

void I8080_Reset(sl8080* cpu);

Reset processor.

u32 I8080_Start(si8080* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I8080_CLOCKMUL is recommended to maintain.

void I8080_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I8080_Cont(si8080* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I8080_IsRunning();

Check if emulation is running.

I8085 CPU Emulator

Config: USE_EMU_I8085 (default 0)

Files **emu_i8085.*** contain emulator of the processor Intel 8085. To activate it, set configuration parameters USE_EMU and USE_EMU_I8085 to value 1.

I8085_CLOCKMUL 2 ... clock multiplier (to achieve lower frequencies and finer timing)

I8085 CPU descriptor:

typedef struct {

... internal variables

volatile u8 intreq; // 1=interrupt request

volatile u8 intins; // instruction RST n to execute at interrupt

pEmu16Read8 readmem; // read byte from memory

pEmu16Write8 writemem; // write byte to memory

pEmu8Read8 readport; // read byte from port

pEmu8Write8 writeport; // write byte to port

pl8085GetSID readsid; // read Serial Input Data Bit (returns 0 or 1)

```
    pl8085SetSOD      writesod;    // write Serial Output Data Bit (val = 0 or 1)
} sl8085;
```

void I8085_InitTab();

Initialize I8085 tables (call it on application start).

void I8085_Reset(sl8085* cpu);

Reset processor.

u32 I8085_Start(sl8085* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I8085_CLOCKMUL is recommended to maintain.

void I8085_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 I8085_Cont(sl8085* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool I8085_IsRunning();

Check if emulation is running.

I8086/I8088/I80186/I80188 CPU Emulator

Config: USE_EMU_I8086 (default 0)

Files **emu_i8086.*** contain emulator of the processors Intel 8086, Intel 8088, Intel 80186 and Intel 80188. To activate it, set configuration parameters USE_EMU and USE_EMU_I8086 to value 1.

Code modifications:

```
#define I8086_CPU_INTEL      1      // 1=use Intel vendor alternative
```

```
#define I8086_CPU_AMD      0      // 1=use AMD (and partially NEC) vendor
#define I8086_CPU_8088       0      // 1=use 8088/80188 timings limitations
#define I8086_CPU_80186       0      // 1=use 80186 extensions
```

I8086_CLOCKMUL 1 ... clock multiplier (to achieve lower frequencies and finer timing)

I8086 CPU descriptor:

```
typedef struct {
```

```
    ... internal variables
```

```
    volatile u8      test;      // TEST pin state
    pEmu32Read8    readmem;   // read byte from memory
    pEmu32Write8   writemem;  // write byte to memory
    pEmu16Read8    readport8;  // read byte from port
    pEmu16Write8   writeport8; // write byte to port
    pEmu16Read16   readport16; // read word from port
    pEmu16Write16  writeport16; // write word to port
```

```
} sI8086;
```

void I8086_InitTab();

Initialize I8086 tables (call it on application start).

void I8086_Reset(sI8086* cpu);

Reset processor.

u32 I8086_Start(sI8086* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*I8086_CLOCKMUL is recommended to maintain.

void I8086_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

```
u32 I8086_Cont(sI8086* cpu, int pwm, u32 freq);
```

Continue emulation without reset processor.

```
Bool I8086_IsRunning();
```

Check if emulation is running.

M6502/M65C02 CPU Emulator

Config: USE_EMU_M6502 (default 0)

Files **emu_M6502.*** contain emulator of the processors MOS-6502 and MOS-65C02. To activate it, set configuration parameters **USE_EMU** and **USE_EMU_M6502** to value 1.

Code modifications:

```
#define M6502_CPU_65C02      1      // 1=use modifications of 65C02 and later
```

M6502_CLOCKMUL 2 ... clock multiplier (to achieve lower frequencies)

M6502 CPU descriptor:

```
typedef struct {  
    ... internal variables  
    volatile u8      intreq;      // 1=interrupt request  
    volatile u8      nmireq;     // 1=NMI request  
    volatile u8      resreq;     // 1=RESET request  
    pEmu16Read8    readmem;    // read byte from memory  
    pEmu16Write8   writemem;   // write byte to memory  
} sM6502;
```

```
void M6502_Reset(sM6502* cpu);
```

Reset processor.

```
u32 M6502_Start(sM6502* cpu, int pwm, u32 freq);
```

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*M6502_CLOCKMUL is recommended to maintain.

void M6502_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 M6502_Cont(sM6502* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool M6502_IsRunning();

Check if emulation is running.

X80 CPU Emulator

Config: USE_EMU_X80 (default 0)

Files **emu_X80.*** contain emulator of the processor Sharp X80 (LR35902). To activate it, set configuration parameters USE_EMU and USE_EMU_X80 to value 1.

X80_CLOCKMUL 2 ... clock multiplier (to achieve lower frequencies)

X80 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8      input_req; // 1=keypad interrupt request
    pEmu16Read8     readmem;   // read byte from memory
    pEmu16Write8    writemem;  // write byte to memory
} sX80;
```

void X80_InitTab();

Initialize X80 tables (call it on application start).

void X80_Reset(sX80* cpu);

Reset processor.

u32 X80_Start(sX80* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*X80_CLOCKMUL is recommended to maintain.

void X80_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 X80_Cont(sX80* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool X80_IsRunning();

Check if emulation is running.

Z80 CPU Emulator

Config: USE_EMU_Z80 (default 0)

Files **emu_Z80.*** contain emulator of the processor Zilog Z80. To activate it, set configuration parameters **USE_EMU** and **USE_EMU_Z80** to value 1.

Z80_CLOCKMUL 2 ... clock multiplier (to achieve lower frequencies)

Z80 CPU descriptor:

```
typedef struct {
    ... internal variables
    volatile u8        intreq;      // 1=maskable interrupt INT request
    volatile u8        nmi;        // 1=nonmaskable interrupt NMI request
    volatile u8        intins;     // instruction RST n to execute interrupt
    pEmu16Read8       readmem;    // read byte from memory
    pEmu16Write8      writemem;   // write byte to memory
    pEmu16Read8       readport;   // read byte from port
    pEmu16Write8      writeport;  // write byte to port
} sZ80;
```

void Z80_InitTab();

Initialize Z80 tables (call it on application start).

void Z80_Reset(sZ80* cpu);

Reset processor.

u32 Z80_Start(sZ80* cpu, int pwm, u32 freq);

cpu ... pointer to CPU structure (fill-up callback functions before start)

pwm ... index of (unused) PWM slice (0..7)

freq ... required emulated frequency in Hz

Start execute program in core 1 - initialize synchronization, reset processor, start synchronization, execute program in core 1. Returns real emulated frequency in Hz. To achieve accurate timing, an integer ratio between system clock and clock frequency*Z80_CLOCKMUL is recommended to maintain.

void Z80_Stop(int pwm);

pwm ... index of used PWM slice (0..7)

Stop emulation.

u32 Z80_Cont(sZ80* cpu, int pwm, u32 freq);

Continue emulation without reset processor.

Bool Z80_IsRunning();

Check if emulation is running.

PC DOS Emulator

Config: USE_EMU_PC (default 0)

Files **PC\emu_pc_*** contain simple PC DOS emulator.

The emulator is very simplistic, due to the small amount of RAM available, and therefore it allows emulation of only a small number of DOS programs. In order to save RAM memory, BIOS and DOS are not interpreted, but emulated by the internal functions of the emulator.

If possible, the emulated program is loaded into Flash memory. The program memory is divided into 4 KB pages and is copied into RAM only when written, thus allowing programs requiring more memory than the RAM size to run. Video memory emulation must be added to the RAM requirements.

The emulator supports emulations of MDA, CGA, PCjr, Tandy, EGA64, EGA128, MCGA and VGA video cards, BIOS video modes number 0 to 19. Video modes with higher resolution are converted to the actual 320x240 LCD resolution. In this case, the program can switch to display part of the image in full resolution.

3.12. EscPkt - Escape Packet Protocol

Files: lib_escpkt.h, lib_escpkt.c

Config: USE_ESCPKT (default 1)

The Escape packet protocol is used to transfer packets over the serial link and to easily distinguish between control and data bytes. The following control bytes are used during transmission:

- NUL** **0x00** (Null) ignored character, can be used as idle transmission
- STX** **0x02** (Start of Text) start of packet data
- ETX** **0x03** (End of Text) end of packet data
- SYN** **0x16** (Synchronous Idle) used in synchronous transmission to synchronise signal, or as a ping character. Receiver may ignore it.
- ESC** **0x1B** (Escape) escape character before data 0x00, 0x02, 0x03, 0x16 or 0x1B. Offset 0x40 (64) is added to the data bytes. Data bytes will change to '@', 'B', 'C', 'V' or '[' characters.

Data bytes with a code corresponding to one of the control characters are transferred by adding the constant 0x40 to their code and prefixing the byte with the Esc character (code 0x1B). This prevents the receiver from synchronizing to false control bytes contained in the packet data. Data of packets are checked using the CRC-16A checksum.

Protocol descriptor

```
typedef struct {  
    sRing* tx_ring; // pointer to transmission ring buffer (NULL = not used)  
    sRing* rx_ring; // pointer to receiving ring buffer (NULL = not used)  
    int     rcv_num; // number of bytes in receive buffer (-1 = waiting for STX)  
    Bool    esc;     // flag of escape character received  
} sEscPkt;
```

void EscPktInit(sEscPkt* esc, sRing* tx_ring, sRing* rx_ring);

esc ... pointer to sEscPkt protocol descriptor

tx_ring ... pointer to transmission ring buffer (NULL = transmission not used)

rx_ring ... pointer to receiving ring buffer (NULL = receiving is not used)

Initialize protocol descriptor.

int EscPktReceive(sEscPkt* esc, u8* buf, int bufsize);

esc ... pointer to sEscPkt protocol descriptor

buf ... destination buffer (must be 2 bytes larger than longest packet,
as reserve for CRC-16A)

bufsize ... size of destination buffer (including 2-byte CRC reserve)

Receive packet, without waiting. Returns length of received packet if result is ≥ 0 , or -1 if packet not received. If packet not received (-1 is returned), packet buffer may contain part of received data. Do not change buffer contents until packet is received (buffer may contain partially received data).

void EscPktSend(sEscPkt* esc, const u8* buf, int len);

esc ... pointer to sEscPkt protocol descriptor

buf ... source buffer with packet to send (without CRC)

len ... length of packet data in source buffer (without CRC)

Send packet, with waiting.

3.13. Event - Event Ring Buffer

Files: lib_event.h, lib_event.c

Config: USE_EVENT (default 1)

Events are used to transfer messages between the interrupt handler and the main program, or between processor cores. Messages are transmitted using a ring buffer with locked access.

Event structure (16 bytes)

```
typedef struct {
    u32    data[6];      // event data (aligned to u32)
} sEvent;
```

Event ring buffer

```
typedef struct {
    sEvent*      buf;      // pointer to data buffer
    uint         size;     // size of data buffer in number of entries
    volatile uint write;   // write offset into buffer
    volatile uint read;    // read offset from buffer
    int          spinlock; // index of spinlock (-1 = not used)
} sEventRing;
```

EVENTRING(name, size, spinlock);

name ... name of the event ring buffer

size ... size of data buffer in number of entries

spinlock ... index of spinlock (-1 = not used)

Macro to create event ring buffer in *.c file with static buffer as global variable. This will create initialized global variable with given name. To declare it in *.h file, use:

```
extern sEventRing name;
```

void EventRingInit(sEventRing* ring, sEvent* buf, uint size, int spinlock);

ring ... pointer to event ring buffer

buf ... event data buffer (of size 'size' entries)

size ... event data buffer size in number of entries

spin ... index of spinlock (-1 = not used)

Initialize event ring buffer. Not needed with static global event ring buffer, created with EVENTRING() macro.

void EventRingClear(sEventRing* ring);

ring ... pointer to event ring buffer

Clear event ring buffer.

Bool EventRingWriteReady(const sEventRing* ring);

ring ... pointer to event ring buffer

Check if available space for write 1 entry into event ring buffer.

Bool EventRingWrite(sEventRing* ring, const sEvent* event);

ring ... pointer to event ring buffer

event ... pointer to event

Write event into event ring buffer, without waiting (returns False if buffer is full).

void EventRingWriteWait(sEventRing* ring, const sEvent* event);

ring ... pointer to event ring buffer

event ... pointer to event

Write event into event ring buffer, wait until ready.

Bool EventRingReadReady(const sEventRing* ring);

ring ... pointer to event ring buffer

Check if event is ready to read from event ring buffer.

Bool EventRingRead(sEventRing* ring, sEvent* event);

ring ... pointer to event ring buffer

event ... pointer to event

Read event from event ring buffer, without waiting (returns False if buffer is empty).

void EventRingReadWait(sEventRing* ring, sEvent* event);

ring ... pointer to event ring buffer

event ... pointer to event

Read event from event ring buffer, wait until ready.

3.14. FAT - FAT File System

Files: lib_fat.h, lib_fat.c

Config: USE_FAT (default 1 if USE_PICOPAD)

The FAT File System module is used to handle the SD card file system. It supports FAT12, FAT16 and FAT32 file systems. It requires an SD card interface (usually connected via SPI interface) to operate. The module only handles short DOS names of the form 8.3 (8 characters name + 3 characters extension). It does not support long names and does not support extended file systems such as exFAT or NTFS. If a larger format SD card is reformatted to an exFAT file system, it must be reformatted to FAT32.

Before starting work, the disk must first be mounted with the DiskMount() function. If it returns False, the SD card is not inserted or has the wrong format. Whether a valid SD card is inserted and just not in the correct format can be verified with the SDConnect() and DiskValid() functions.

After writing the data, it is necessary to empty the writing cache by completing the writing to the SD card - close the files FileClose() and use the DiskFlush() function to complete the writing of the buffers. It is not necessary unmount disk using DiskUnmount().

Constants

PATHCHAR	'/	default path separator ('\' can be used too)
PATH_MAX	254	max. length of path and command line
PATH_BUF	256	path buffer size
SECT_SIZE	512	sectors size (bytes)
SECT_NONE	~0UL	none sector, or invalid return value
FORMAT_MAGIC	0xBAB00CA	magic of DiskFormat() function
FS_NONE	0	invalid file system
FS_FAT12	1	FAT12
FS_FAT16	2	FAT16
FS_FAT32	3	FAT32

File attributes

ATTR_RO	B0	Read only
ATTR_HID	B1	Hidden
ATTR_SYS	B2	System
ATTR_VOL	B3	Volume label
ATTR_DIR	B4	Directory
ATTR_ARCH	B5	Archive
ATTR_NORM	0	search files with normal attributes

ATTR_FILE_MASK	search files with all attributes
ATTR_DIR_MASK	search directories
ATTR_NONE	0xFF flag - file not found

FAT directory entry (32 bytes)

```
typedef struct {
    char    name[11];   // (0x00) file name and extension
    u8      attr;       // (0x0B) attributes (ATTR_RO,...)
    u8      ntres;      // (0x0C) lower case flag (should be 0 = no EAs)
    u8      ctime10;    // (0x0D) created time sub-second in 10 ms, value 0..199,
                       // or first character of deleted file
    u16     ctime;      // (0x0E) created time in DOS format (b0-4: seconds/2,
                       // b5-10: minutes, b11-15: hours)
    u16     cdate;      // (0x10) created date in DOS format (b0-4: day,
                       // b5-8: month, b9-15: year - 1980)
    u16     adate;      // (0x12) last accessed date in DOS format (b0-4: day,
                       // b5-8: month, b9-15: year - 1980)
    u16     clustH;     // (0x14) higher 16 bits of first cluster (FAT32)
    u16     wtime;      // (0x16) last write time in DOS format (b0-4: seconds/2,
                       // b5-10: minutes, b11-15: hours)
    u16     wdate;      // (0x18) last write date in DOS format (b0-4: day,
                       // b5-8: month, b9-15: year - 1980)
    u16     clustL;     // (0x1A) lower 16 bits of first cluster
    u32     size;       // (0x1C) file size in bytes
} sDir;
```

Open file/directory structure

```
typedef struct {
    char    name[11];   // short file name (0=not open)
    u8      attr;       // attributes (ATTR_RO,..., ATTR_NONE=file not exist)
    sDir*   dir;        // pointer to directory entry in disk buffer
    u32     dirsect;    // sector with directory entry
    u16     wtime;      // last write time
    u16     wdate;      // last write date
    u32     size;       // file size
    u32     sclust;    // start cluster
    u32     off;        // current read/write offset
} sFile;
```

```

    u32      clust;      // current read/write cluster
    u32      sect;      // current read/write sector (0=end of directory)
} sFile;

```

File info structure

```

typedef struct {
    u32      size;      // file size
    u16      wtime;     // last write time
    u16      wdate;     // last write date
    char     name[12+1]; // file name (without spaces, with extension and with '.')
    u8       namelen;   // length of file name
    u8       attr;      // file attributes
    u8       res;       // ... align
} sFileInfo;

sFileInfo FileInfoTmp; // temporary file info, used internally in some functions

```

Bool Disk_ClustValid(u32 clust);

Check if cluster is valid.

u16 DosDate(u8 d, u8 m, u16 y);

- d** ... day 1..31
- m** ... month 1..12
- y** ... year 1980..2107

Compose DOS date.

u16 DosTime(u8 h, u8 m, u8 s);

- h** ... hour
- m** ... minute
- s** ... second

Compose DOS time.

u8 DosDay(u16 d);

- d** ... date in DOS format

Get day from DOS date.

u8 DosMon(u16 d);

- d** ... date in DOS format

Get month from DOS date.

u16 DosYear(u16 d);

d ... date in DOS format

Get year from DOS date.

u8 DosSec(u16 t);

t ... time in DOS format

Get second from DOS time.

u8 DosMin(u16 t);

t ... time in DOS format

Get minute from DOS time.

u8 DosHour(u16 t);

t ... time in DOS format

Get hour from DOS time.

u16 FATDATE();

Default FAT date 1/1/2000.

u16 FATTIME();

Default FAT time 12:00:00.

void DiskUnmount();

Unmount disk.

Bool DiskMount();

Mount disk (returns False on error - no SD card or invalid file system, exFAT not supported).

Bool DiskMounted();

Check if disk is mounted.

Bool DiskAutoMount();

Mount disk if not mounted.

Bool DiskFlush();

Flush disk write buffers (should be called repeatedly after some time).

char FileUpperCase(char ch);

ch ... character

Convert character to uppercase.

char FileLowerCase(char ch);

ch ... character

Convert character to lowercase.

const char* DiskFATName();

Get file system name (“none”, “FAT12”, “FAT16”, .”FAT32”)

u8 DiskFATType();

Get disk FAT type **FS_** * (0=invalid, 1=FAT12, 2=FAT16, 3=FAT32).

u8 DiskFATNum();

Get number of FATs (1 or 2).

u8 DiskClustSect();

Get number of sectors per cluster.

u16 DiskRootNum();

Get number of root directory entries (only FAT12 or FAT16; 0 on FAT32).

u32 DiskTotalSect();

Get total disk size in number of sectors (including system tables).

u32 DiskClustSize();

Get size of cluster in bytes.

u32 DiskLastClust();

Get last allocated cluster (-1=unknown).

u32 DiskFreeClust();

Get number of free clusters (returns -1 on error).

u32 DiskFreeKB();

Get size of free space in KB. Returns -1 on error.

u32 DiskTotalClust();

Get total number of clusters (only data, without system tables).

u32 DiskTotalKB();

Get total size in KB (only data, without system tables).

u32 DiskUsedClust();

Get number of used clusters. Returns -1 on error.

u32 DiskUsedKB();

Get size of used space in KB. Returns -1 on error.

u32 DiskFatEntry();

Get number of FAT entries.

u32 DiskFatSize();

Get size of FAT (number of sectors per one FAT table).

u32 DiskVolBase();

Get index of volume base sector.

u32 DiskFatBase();

Get index of FAT base sector.

u32 DiskDirBase();

Get index of root directory base sector (FAT12, FAT16) or cluster (FAT32).

u32 DiskDataBase();

Get index of data base sector.

u32 DiskDirClust();

Get start cluster of current directory (0=root).

void FileInit(sFile* file);

file ... not-initialized file descriptor

Initialize file structure sFile (set as not-open).

Bool FileIsOpen(sFile* file);

file ... initialized file descriptor

Check if file is open.

Bool FileCreate(sFile* file, const char* path);

file ... not-initialized file descriptor

path ... path to the file, including file name and extension

Create new file. Returns False on error and marks file as closed.

Bool FileOpen(sFile* file, const char* path);

file ... not-initialized file descriptor

path ... path to the file, including file name and extension

Open existing file. Returns False on error and marks file as closed.

u32 FileRead(sFile* file, void* buf, u32 num);

file ... open file descriptor

buf ... destination buffer

num ... number of bytes to read

Read from file. Returns number of bytes read, or less on error.

u32 FileWrite(sFile* file, const void* buf, u32 num);

file ... open file descriptor

buf ... source buffer

num ... number of bytes to write

Write to file. Returns number of bytes written, or less on error.

void FilePrintChar(sFile* file, char ch);

file ... open file descriptor

ch ... character to print

Print character to file.

u32 FilePrintText(sFile* file, const char* txt);

file ... open file descriptor

txt ... ASCIIZ text to print

Print unformatted text to file. Returns number of characters.

u32 FilePrintArg(sFile* file, const char* fmt, va_list args);

file ... open file descriptor

fmt ... format string

args ... arguments

Formatted print string to file, with argument list. Returns number of characters.

u32 FilePrint(sFile* file, const char* fmt, ...);

file ... open file descriptor

fmt ... format string

... ... arguments

Formatted print string to file, with variadic arguments. Returns number of characters.

Bool FileFlush(sFile* file);

file ... open file descriptor

Flush file writes and flush disk buffers. Returns False on error.

Bool FileClose(sFile* file);

file ... open file descriptor

Close file and flush disk buffers. Returns False on error. If the file was only read from, not written to, it is not necessary to close the file.

Bool SetDir(const char* path);

path ... path to directory

Change current directory. Returns False on error.

u16 GetDir(char* buf, u16 len);

buf ... destination buffer

len ... length of destination buffer, including trailing zero

Get path to current directory. Returns length of path, or 0 on error. The function loops through the directories starting from the current directory and constructs the path. If the total path length exceeds the buffer size, it returns 0 as an error flag.

Bool FileSeek(sFile* file, u32 off);

file ... open file descriptor

off ... new offset in the file

Seek file read/write pointer. Returns False on error. The pointer is limited to the file size, cannot be set beyond the end of the file.

Bool FileSeekEnd(sFile* file);

file ... open file descriptor

Seek file read/write pointer to end of the file. Returns False on error.

u32 FilePos(sFile* file);

file ... open file descriptor

Get current seek position of open file.

u32 FileSize(sFile* file);

file ... open file descriptor

Get size of open file.

u8 FileAttr(sFile* file);

file ... open file descriptor

Get attributes of open file **ATTR_***.

u16 FileWTime(sFile* file);

file ... open file descriptor

Get last write time of open file (in DOS format).

u16 FileWDate(sFile* file);

file ... open file descriptor

Get last write date of open file (in DOS format).

Bool FindOpen(sFile* find, const char* path);

find ... not-initialized find descriptor

path ... path to directory (without search pattern)

Open searching files. Returns False on error and marks searching as closed. Searching can be reopen (without close) to rewind search from begin again (for example with a different file mask).

Bool FindNext(sFile* find, sFileInfo* fileinfo, u8 attr, const char* pat);

find ... open find descriptor

fileinfo ... info structure to get info about the found file

attr ... attributes to search more to normal attributes

(use **ATTR_DIR_MASK** to search all except volumes)

pat ... ASCIIZ pattern (?=one character matching, *=rest matching)

Find next (or first) file. Returns False on error or end of directory.

Bool FindClose(sFile* find);

find ... open find descriptor

Close file searching. Returns False on error (if not open). No need to be used.

Bool FileExist(const char* path);

path ... path to the file or directory

Check if file or directory exists. Returns False on error or if not exist.

Bool FileInfo(const char* path, sFileInfo* fileinfo);

path ... path to the file or directory

fileinfo ... info structure to get info about the found file

Get file or directory info. Returns False on error.

Bool SetFileSize(sFile* file, u32 size);

file ... open file descriptor

size ... required new size

Set file size - truncate or enlarge the file. Returns False on error.

Bool FileDelete(const char* path);

path ... path to the file or directory

Delete file or directory. Directory must be empty. Returns False on error.

Bool DirCreate(const char* path);

path ... path to directory

Create directory. Returns False on error.

Bool FileMove(const char* oldpath, const char* newpath);

oldpath ... old path to the file or directory (name with extension)

newpath ... new path to the file or directory (name with extension)

Rename or move file or directory. Returns False on error.

u32 GetFileSize(const char* path);

path ... path to the file

Get size of not-open file. Returns **SECT_NONE** (= (u32)-1) on error.

u8 GetFileAttr(const char* path);

path ... path to the file or directory

Get attributes of not-open file or directory. Returns **ATTR_NONE** (= 0xFF) on error.

Bool SetFileAttr(const char* path, u8 attr);

path ... path to the file or directory

attr ... new attributes

Set attributes of not-open file or directory. Cannot change **ATTR_DIR** or **ATTR_VOL** attributes - these bits are ignored. Returns False on error.

u16 GetFileTime(const char* path);

path ... path to the file or directory

Get write time of not-open file or directory. Returns (u16)-1 on error.

u16 GetFileDate(const char* path);

path ... path to the file or directory

Get write date of not-open file or directory. Returns (u16)-1 on error.

Bool SetFileTime(const char* path, u16 time);

path ... path to the file or directory

time ... new write time in DOS format

Set write time of not-open file or directory. Returns False on error.

Bool SetFileDate(const char* path, u16 date);

path ... path to the file or directory

date ... new write date in DOS format

Set write date of not-open file or directory. Returns False on error.

u32 GetDiskSerial();

Get disk serial number (0 = invalid).

Bool GetDiskLabel();

Get disk volume label. Returns label in FileInfoTmp. Returns False on error or if label does not exists - sets empty label.

Bool SetDiskLabel(const char* label);

label ... new disk volume label (max. 11 characters)

Set disk volume label. Use empty label to delete volume label. Returns False on error.

u32 GetMediaSize();

Get media size in number of sectors. Returns 0 on error.

u32 GetMediaSizeKB();

Get media size in KB. Returns 0 on error.

Bool DiskFormat(u8 fs, u8 clust, Bool mbr, u32 magic);

fs ... required filesystem **FS_FAT12,... (FS_NONE = auto)**

clust ... cluster size in number of sectors 0,1,2,4,8,16,32,64,128 (0 = auto)

mbr ... use MBR

magic ... FORMAT_MAGIC (=0BAB00CA)

Format SD card. Returns False on error.

Bool DiskFormatDef(u32 magic);

magic ... FORMAT_MAGIC (=0BAB00CA)

Format SD card with recommended default settings.

Bool FileCheckExt(sFileInfo* fi, const char* ext);

fileinfo ... info structure with file info

ext ... file extension (0 to 3 characters ASCIIZ)

Check file extension if equal (compare with pattern). Auto uppercase.

Bool CheckApp(u32* applen, u32* proglen, u32* appcrc);

applen ... pointer to get application length, without header (NULL = not needed)

proglen ... pointer to get total program length, with boot loader
and with header (NULL = not needed)

appcrc ... pointer to get application CRC (NULL = not needed)

Check application in Flash memory. Application starts at address APPSTART = XIP_BASE + BOOTLOADER_SIZE.

Bool CheckLoaderData();

Check boot loader data. Boot loader data are at address LOADERDATA.

const sAppPath* CheckAppPath();

Check application home path. Returns pointer to sAppPath, or NULL on error.

int GetHomePath(char* path, const char* def);

path ... buffer of size APPPATH_PATHMAX+1 (= 248) to get path with terminating 0

def ... default path (used on error)

Get application home path from Flash memory. Sets home path as current directory. Returns length of the path.

3.15. FileSel - File Selection

Files: lib_filesel.h, lib_filesel.c

Config: USE_FILELIST

The FileSel library is used to select a file from the SD card. It is typically used to select a program to run for the emulator. It requires access to the SD card, keypad control, and output to the drawing functions of the graphics library.



Structure to define colors for FileSel:

```
typedef struct {
    COLTYPE titlefg;           // title foreground color (COL_BLACK)
    COLTYPE titlebg;           // title background color (COL_WHITE)
    COLTYPE filefg;            // file foreground color (COL_CYAN)
    COLTYPE filebg;            // file background color (COL_BLUE)
    COLTYPE dirfg;             // directory foreground color (COL_WHITE)
    COLTYPE dirbg;             // directory background color (COL_BLUE)
    COLTYPE curfg;             // cursor coreground color (COL_BLACK)
    COLTYPE curbg;             // cursor background color (COL_CYAN)
    COLTYPE infofg;            // info text foreground color (COL_GREEN)
    COLTYPE infobg;            // info text background color (COL_BLUE)
    COLTYPE textfg;            // text foreground color (COL_GRAY)
    COLTYPE textbg;            // text background color (COL_BLACK)
    COLTYPE biginfofg;         // big info text foreground color (COL_YELLOW)
    COLTYPE biginfobg;         // big info text background color (COL_BLACK)
    COLTYPE bigerrfg;          // big error foreground color (COL_YELLOW)
    COLTYPE bigerrbg;          // big error background color (COL_RED)
} sFileSelColors;
```

Global variables:

```
const sFileSelColors FileSelColBlue; // colors template - blue theme
const sFileSelColors FileSelColGreen; // colors template - green theme

// current path
#define FILESEL_PATHMAX      240    // max. length of path
char FileSelPath[FILESEL_PATHMAX+1]; // current path
int FileSelPathLen; // length of path
int FileSelRootLen; // length of root

// file extension
char FileSelExt[4]; // file extension in uppercase
int FileSelExtLen; // length of file extension

// temporary buffer
#define FILESEL_TEMPBUF 512 // size of temporary buffer
char FileSelTempBuf[FILESEL_TEMPBUF+1];
int FileSelTempBufNum; // number of bytes in temporary buffer

// selected last name
char FileSelLastName[9]; // last selected name (without extension)
int FileSelLastNameLen; // length of last name (without extension)
int FileSelLastNameTop; // top file of last name
u8 FileSelLastNameAttr; // attributes of last name
u32 FileSelLastNameSize; // size of last name

sFile FileSelPrevFile; // preview file (name[0] = 0 if not open)

void FileSelDispBigInfo(const char* text);
Display big info text.

void FileSelDispBigErr(const char* text);
Display big error text, wait for key press.
```

```

void FileSelInit(const char* root, const char* title, const char* ext, const
sFileSelColors* col);
void FileSelInit2(const char* root, const char* title, const char* ext, const char*
ext2, const sFileSelColors* col);
void FileSelInit3(const char* root, const char* title, const char* ext, const char*
ext2, const char* ext3, const sFileSelColors* col);

root ... path to root directory (terminated with 0; max. 240 characters)
    - use uppercase
    - use '/' character as path separator
    - start path with '/' root entry
    - do not end path with '/' character (except root path)

title ... title on file panel, used instead of root path
        (terminated with 0; max. 20 characters)

ext, ext2, ext3 ... file extension (1 to 3 characters in uppercase, terminated with 0)

col ... colors (recommended &FileSelColGreen or &FileSelColBlue)

```

Initialize file selection.

Bool FileSel();

Select file (returns True if file selected, or False to break).

If file selected with True:

- FileSelPath contains current directory (function sets this directory as active, using SetDir(FileSelPath))
- FileSelExt contains file extension
- FileSelLastName contains file name without extension
- FileSelTempBuf contains file name with extension (can be used to
- function displays info "Loading..."

Open the file as in this example:

```

if (!FileSel()) return; ... select file, or return from program
if (!FileOpen(&FileSelPrevFile, FileSelTempBuf))
    FileSelDispBigErr("Cannot open file");
else
{
    ...loading file
    FileClose(&FileSelPrevFile);
    ...processing file
}
... repeat with FileSel() again

```

3.16. List - Doubly Linked List

Files: lib_list.h, lib_list.c

Config: USE_LIST (default 1)

A doubly linked list is used to create lists of items in which pointers point to the previous and next item. The items are concatenated into a circular list. The start and end of the list is bounded by the initial list item (list carrier). The list structure is added to the data structure so that the data items can be chained into a doubly-linked list.

List entry

```
typedef struct sListEntry_ {  
    struct sListEntry_*  next; // pointer to next list entry  
    struct sListEntry_*  prev; // pointer to previous list entry  
} sListEntry;
```

void ListEntryAddAfter(sListEntry* thisentry, sListEntry* newentry);

thisentry ... current list entry

newentry ... added new entry

Add new list entry after this entry.

void ListEntryAddBefore(sListEntry* thisentry, sListEntry* newentry);

thisentry ... current list entry

newentry ... added new entry

Add new list entry before this entry.

void ListEntryRemove(sListEntry* entry);

entry ... list entry

Remove entry from the list.

base* BASEFROMLIST(entry, base, member)

entry ... pointer to list entry

base ... base structure type

member ... name of member variable of list entry in base structure

Macro to get base structure from the list entry.

Simple list

```
typedef struct {  
    sListEntry  head; // list head
```

```
} sList;
```

void ListInit(sList* list);

list ... simple list

Initialize list.

Bool ListIsEmpty(const sList* list);

list ... simple list

Check if list is empty (check if it points to itself).

sListEntry* ListGetHead(sList* list);

list ... simple list

Get head of the list - used as stop mark when walking through the list.

sListEntry* ListGetFirst(sList* list);

list ... simple list

Get first list entry.

sListEntry* ListGetLast(sList* list);

list ... simple list

Get last list entry.

void ListAddFirst(sList* list, sListEntry* entry);

list ... simple list

entry ... new list entry

Add new entry into start of list.

void ListAddLast(sList* list, sListEntry* entry);

list ... simple list

entry ... new list entry

Add new entry into end of list.

to add new entry after entry in the list - use ListEntryAddAfter

to add new entry before entry in the list - use ListEntryAddBefore

to remove entry from the list - use ListEntryRemove

LISTFOREACH(entry, list)

entry ... variable of sListEntry* type

list ... simple list sList*

Macro header of loop to walking through the simple list in forward direction.

LISTFOREACHBACK(entry, list)

entry ... variable of sListEntry* type

list ... simple list sList*

Macro header of loop to walking through the simple list in backward direction.

LISTFOREACHSAFE(entry, next, list)

entry ... variable of sListEntry* type

next ... temporary variable of next sListEntry*

list ... simple list sList*

Macro header of loop to safe walking through the simple list (entry can be deleted).

List with count

```
typedef struct {  
    sListEntry  head;    // list head  
    int         num;    // number of entries  
} sNumList;
```

void NumListInit(sNumList* list);

list ... list with count

Initialize list with count.

Bool NumListIsEmpty(const sNumList* list);

list ... list with count

Check if list with count is empty.

sListEntry* NumListGetHead(sNumList* list);

list ... list with count

Get head of the list - used as stop mark when walking through the list.

sListEntry* NumListGetFirst(sNumList* list);

list ... list with count

Get first list entry.

sListEntry* NumListGetLast(sNumList* list);

list ... list with count

Get last list entry.

void NumListAddFirst(sNumList* list, sListEntry* entry);

list ... list with count

entry ... new list entry

Add new entry into start of list with count.

void NumListAddLast(sNumList* list, sListEntry* entry);

list ... list with count

entry ... new list entry

Add new entry into end of list with count.

void NumListAddAfter(sNumList* list, sListEntry* oldentry, sListEntry* newentry);

list ... list with count

oldentry ... old list entry

newentry ... new list entry

Add new entry into list with count, after oldentry.

void NumListAddBefore(sNumList* list, sListEntry* oldentry, sListEntry* newentry);

list ... list with count

oldentry ... old list entry

newentry ... new list entry

Add new entry into list with count, before oldentry.

void NumListRemove(sNumList* list, sListEntry* entry);

list ... list with count

entry ... removed entry

Remove entry from the list with count.

NUMLISTFOREACH(entry, numlist)

entry ... variable of sListEntry* type

numlist ... list with count

Macro header of loop to walking through the list with count in forward direction.

NUMLISTFOREACHBACK(entry, numlist)

entry ... variable of sListEntry* type

numlist ... list with count

Macro header of loop to walking through the list with count in backward direction.

NUMLISTFOREACHSAFE(entry, next, numlist)

entry ... variable of sListEntry* type

next ... temporary variable of next sListEntry*

numlist ... list with count

Macro header of loop to safe walking through the list with count (entry can be deleted).

3.17. Malloc - Memory Allocator

Files: lib_malloc.h, lib_malloc.c

Config: USE_MALLOC (default 1), USE_MEMLOCK (default 1)

Malloc is a memory manager used to dynamically allocate and free memory blocks in RAM. It uses the remaining unused RAM located after the application data (pointers `_malloc_start_` to `_malloc_end_`) to allocate memory. The allocated memory blocks are always aligned to 8 bytes. The memory manager uses a chain of "memory chunk" descriptors to keep track of which portion of memory is allocated and which is free.

To enable memory sharing between processor cores, the memory manager locks access to memory descriptors during modification. To do this, it uses spinlock locks with index `SYS_SPIN` (default 31), depending on the setting of the `USE_MEMLOCK` configuration switch. When `USE_MEMLOCK = 0`, the locks are not used at all for memory access. Memory allocation can be faster, but memory can only be allocated in one core (typically core 0) and memory cannot be allocated in the interrupt handler. When `USE_MEMLOCK = 1` (the default setting), basic SpinLock locks are used without interrupt locking. In this variant, memory can be allocated in both processor cores but must not be allocated in the interrupt handler. The highest level is `USE_MEMLOCK = 2`, where the interrupt service is disabled during the access lock. Thus it is possible to allocate memory even during the interrupt service - this applies for example to the possibility of working with text variables. The disadvantage of this last option, however, is that interrupts can be disabled for long periods of time (moving memory during reallocation) and interrupt handling can be delayed or even lost.

Memory chunk header (size 8 bytes + data)

Allocable memory region starts with START empty chunk (marked as used, no data) and ends with END empty chunk (marked as used, no data).

```
typedef struct sMemChunk_
{
    // pointer to previous memory block (NULL=first block)
    struct sMemChunk_* prev;

    // size of this memory chunk (including header, > 0 free, < 0 used block)
    s32 size;

    // --- here start data of the block

    // list entry in list of free memory chunks
    // - this entry is rewritten by memory block data
    sListEntry list;

} sMemChunk;
```

void* MemGetBase();

Get base address of allocable memory region.

u32 MemGetTotal();

Get total size of allocable memory region (including headers).

int MemGetTotalNum();

Get total number of blocks (used and free).

u32 MemGetFree();

Get total size of free memory (without headers).

int MemGetFreeNum();

Get number of free memory blocks.

u32 MemGetUsed();

Get total size of used (allocated) memory.

int MemGetUsedNum();

Get number of used (allocated) memory blocks.

u32 MemGetMax();

Get largest free block (in bytes), which can be allocated.

void MemInit();

Initialize memory allocator. This function is automatically called during application startup from the internal RuntimeInit() function.

void* MemAlloc(u32 size);

void* malloc(size_t size);

size ... required size of memory block in bytes

Allocate memory block. Returns NULL on error (or if size = 0). Block address is aligned to 8 bytes.

void* calloc(size_t nitems, size_t size);

nitems ... number of elements of size 'size'

size ... size of one element

Allocate 'nitems' number of elements of size 'size' and clears memory. Returns NULL on error. Block address is aligned to 8 bytes.

u32 MemSize(void* addr);

addr ... address of memory block (can be NULL)

Get size of memory block (without chunk header). Returns 0 in case of invalid memory block. Can be used to base check block validity, but recognizing a valid block may not be 100% accurate (can mark an invalid non-zero address as a valid block).

void MemFree(void* addr);

void free(void* addr);

addr ... address of memory block (can be NULL)

Release memory block.

void* MemResize(void* addr, u32 size);

void* realloc(void* addr, size_t size);

addr ... address of memory block (can be NULL)

size ... required new size of memory block in bytes (can be 0)

Resize memory block. Returns new block or NULL on error. If address of memory block is NULL, new block will be allocated. If required size of memory block is 0, memory block will be released. The function will return the address of the new block, or may return the address of the same block if it matches the new conditions.

void* MemCopy(void* dst, const void* src, u32 size);

void* memcpy(void* dst, const void* src, size_t size);

dst ... destination address of the memory block

src ... source address of the memory block

size ... size of memory block

Copy memory block. Memory regions should not overlap. Returns destination address.

void* MemMove(void* dst, const void* src, u32 size);

void* memmove(void* dst, const void* src, size_t size);

dst ... destination address of the memory block

src ... source address of the memory block

size ... size of memory block

Move memory block, in ascending or descending direction. Memory regions can overlap. Returns destination address.

int MemComp(const void* buf1, const void* buf2, u32 size);

buf1 ... buffer 1 with 1st data

Compare memory. Returns difference of last character: <0 if buf1<buf2, 0 if buf1==buf2, >0 if buf1>buf2.

int MemCheck();

Check memory integrity. Returns 0=OK or returns error code:

0 = OK

1 = invalid START chunk

2 = invalid STOP chunk
3 = invalid chunk Prev pointer
4 = invalid chunk size
5 = invalid summary
6 = invalid free chunks size
7 = invalid number of free chunks

int MemFastCheck();

Fast check memory allocator. Returns 0 if OK, or returns error code.

void MemLongCheckInit();

Initialize long check memory manager. **MEMCHECK_NUM** is number of allocable memory blocks.

int MemLongCheck(int loop);

loop ... number of test loops

Long check memory manager. Returns 0 if OK, or returns error code.

u32 SafeInc(volatile u32* num);

num ... pointer to the variable

Atomic increment the variable. Returns resulting incremented value. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeDec(volatile u32* num);

num ... pointer to the variable

Atomic decrement the variable. Returns resulting decremented value. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeAdd(volatile u32* num, s32 val);

num ... pointer to the variable

val ... value to add

Atomic add to the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeSub(volatile u32* num, s32 val);

num ... pointer to the variable

val ... value to subtract

Atomic subtract from the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeAnd(volatile u32* num, u32 val);

num ... pointer to the variable

val ... value to AND

Atomic AND to the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeOr(volatile u32* num, u32 val);

num ... pointer to the variable

val ... value to OR

Atomic OR to the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeXor(volatile u32* num, u32 val);

num ... pointer to the variable

val ... value to XOR

Atomic XOR to the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeNot(volatile u32* num);

num ... pointer to the variable

Atomic NOT the variable. Returns result of the operation. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeExc(volatile u32* num, u32 val);

num ... pointer to the variable

val ... new value to exchange

Atomic exchange value of the variable. Returns old value. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

u32 SafeCompExc(volatile u32* num, u32 val, u32 comp);

num ... pointer to the variable

val ... new value to exchange

comp ... value to compare

Atomic compare and exchange. If old value of 'num' is equal to 'comp', the new value 'val' is stored to the 'num' variable, otherwise no operation is performed. Returns old value of the variable. Uses the same memory lock as memory manager (configure with **USE_MEMLOCK**).

3.18. Mat2D - 2D Transformation Matrix

Files: lib_mat2d.h, lib_mat2d.c

Config: USE_MAT2D (default 1)

The 2D transformation matrix is used to quickly transform images - rotate, scale, skew. When preparing the transformation matrix, initialize the matrix to the unit matrix first. Then add each transformation in reverse order from the end than they are needed in the result to do. Applying the transformation matrix to the coordinates will perform the same transformation as if it corresponded to the reverse order of the partial transformations entered.

FRACT 12 number of bits of fractional part of fractint number

FRACTMUL B12 fraction multiplier

Transformation matrix structure

```
typedef struct {  
    float m11, m12, m13;  
    float m21, m22, m23;  
} sMat2D;
```

int TOFRACT(float f);

f ... float member

Convert float number to fraction integer.

float Mat2D_GetX(sMat2D* m, float x, float y);

m ... transformation matrix

x ... source X coordinate

y ... source Y coordinate

Get destination X coordinate transformed by transformation matrix.

float Mat2D_GetY(sMat2D* m, float x, float y);

m ... transformation matrix

x ... source X coordinate

y ... source Y coordinate

Get destination Y coordinate transformed by transformation matrix.

void Mat2D_Zero(sMat2D* m);

m ... transformation matrix

Set zero matrix.

void Mat2D_Unit(sMat2D* m);

m ... transformation matrix

Set unit matrix.

void Mat2D_Copy(sMat2D* m, const sMat2D* src);

m ... transformation matrix

src ... source transformation matrix

Copy from another matrix.

void Mat2D_TransX(sMat2D* m, float tx);

m ... transformation matrix

tx ... translation in X direction

Translate matrix in X direction.

void Mat2D_TransY(sMat2D* m, float ty);

m ... transformation matrix

ty ... translation in Y direction

Translate matrix in Y direction.

void Mat2D_ScaleX(sMat2D* m, float sx);

m ... transformation matrix

sx ... scale in X direction

Scale matrix in X direction.

void Mat2D_ScaleY(sMat2D* m, float sy);

m ... transformation matrix

sy ... scale in Y direction

Scale matrix in Y direction.

void Mat2D_RotSC(sMat2D* m, float sina, float cosa);

m ... transformation matrix

sina .. sine of rotation

cosa .. cosine of rotation

Rotate matrix, using sine and cosine.

void Mat2D_Rot(sMat2D* m, float a);

m ... transformation matrix

a ... angle in radians

Rotate matrix, using angle.

void Mat2D_Rot90(sMat2D* m);

m ... transformation matrix

Rotate matrix by 90 deg.

void Mat2D_Rot180(sMat2D* m);

m ... transformation matrix

Rotate matrix by 180 deg.

void Mat2D_Rot270(sMat2D* m);

m ... transformation matrix

Rotate matrix by 270 deg.

void Mat2D_ShearX(sMat2D* m, float dx);

m ... transformation matrix

dx ... shear delta in X direction

Shear matrix in X direction.

void Mat2D_ShearY(sMat2D* m, float dy);

m ... transformation matrix

dy ... shear delta in Y direction

Shear matrix in Y direction.

void Mat2D_FlipY(sMat2D* m);

m ... transformation matrix

Flip matrix in Y direction.

void Mat2D_FlipX(sMat2D* m);

m ... transformation matrix

Flip matrix in X direction.

**void Mat2D_PrepDrawImg(sMat2D* m, int ws, int hs, int x0, int y0, int wd, int hd,
float shearx, float sheary, float r, float tx, float ty);**

m ... transformation matrix

ws ... source image width

hs ... source image height

x0 ... reference point X on source image

y0 ... reference point Y on source image

wd ... destination image width (negative = flip image in X direction)

hd ... destination image height (negative = flip image in Y direction)

shearx ... shear image in X direction

sheary ... shear image in Y direction
r ... rotate image (angle in radians)
tx ... shift in X direction (if use ws = whole image width)
ty ... shift in Y direction (if use hs = whole image height)

Prepare transformation matrix for DrawImgMat() function. Set the reference coordinates to the center of the image ($x0=IMGW/2$, $y0=IMGH/2$) so that operations such as rotation refer to the center of the image. The transformations specified in the parameters will be applied to the resulting image in the following order: setting the $x0$ and $y0$ reference point, scaling wd and hd , skewing $shearx$ and $sheary$, rotation rot , shift tx and ty . If image size is zero ($wd==0$ or $hd==0$) returns invalid zero matrix.

void Mat2D_ExportInt(const sMat2D* m, int* mat);

m ... transformation matrix
mat ... integer array of 6 integers

Export matrix to int array[6] fractional integers.

3.19. MiniRing - Mini-Ring Buffer

Files: lib_miniring.h, lib_miniring.c
Config: USE_MINIRING (default 1)

MiniRing is a minimized ring buffer. It does not use locking access to the buffer, so it is not protected against concurrent accesses from multiple cores, but it can still be used in conjunction with interrupts due to the order of access to volatile pointers.

Mini-ring buffer

```
typedef struct {  
    u8*          buf;      // pointer to data buffer  
    uint         size;     // size of data buffer in bytes  
    volatile uint write;   // write offset into buffer  
    volatile uint read;   // read offset from buffer  
} sMiniRing;
```

MINIRING(name,size);

name ... name of the mini-ring buffer

size ... size of data buffer in bytes

Macro to create mini-ring buffer in *.c file with static buffer as global variable. This will create initialized global variable with given name. To declare it in *.h file, use:

```
extern sMiniRing name;
```

void MiniRingInit(sMiniRing* ring, u8* buf, uint size);

ring ... pointer to mini-ring buffer

buf ... data buffer (of size 'size' bytes)

size ... buffer size in bytes

Initialize mini-ring buffer. Not needed with static global mini-ring buffer, created with MINIRING() macro.

void MiniRingClear(sMiniRing* ring);

ring ... pointer to mini-ring buffer

Clear mini-ring buffer.

Bool MiniRingWriteReady(sMiniRing* ring);

ring ... pointer to mini-ring buffer

Check if available space for write 1 byte into mini-ring buffer.

Bool MiniRingWrite(sMiniRing* ring, u8 data);

ring ... pointer to mini-ring buffer

data ... data byte to write

Write byte into mini-ring buffer, without waiting. Returns False if buffer is full.

uint MiniRingBufWrite(sMiniRing* ring, const u8* data, uint len);

ring ... pointer to mini-ring buffer

data ... data buffer with data to write

len ... length of data in bytes

Write data from buffer into mini-ring buffer, without waiting. Data is lost if buffer is full. Returns number of bytes successfully written to the buffer.

void MiniRingWriteWait(sMiniRing* ring, u8 data);

ring ... pointer to mini-ring buffer

data ... data byte to write

Write byte into mini-ring buffer, wait until ready.

void MiniRingBufWriteWait(sMiniRing* ring, const u8* data, uint len);

ring ... pointer to mini-ring buffer

data ... data buffer with data to write

len ... length of data in bytes

Write data from buffer into mini-ring buffer, wait until ready.

Bool MiniRingReadReady(sMiniRing* ring);

ring ... pointer to mini-ring buffer

Check if byte is ready to read from mini-ring buffer.

u8 MiniRingRead(sMiniRing* ring);

ring ... pointer to mini-ring buffer

Read byte from mini-ring buffer, without waiting. Returns 0 if buffer is empty.

uint MiniRingBufRead(sMiniRing* ring, u8* data, uint len);

ring ... pointer to mini-ring buffer

data ... data buffer to get data

len ... length of data in bytes

Read data from mini-ring buffer, without waiting. Returns number of bytes successfully read from the buffer.

u8 MiniRingReadWait(sMiniRing* ring);

ring ... pointer to mini-ring buffer

Read byte from mini-ring buffer, wait until ready.

void MiniRingBufReadWait(sMiniRing* ring, u8* data, uint len);

ring ... pointer to mini-ring buffer

data ... data buffer to get data

len ... length of data in bytes

Read data from mini-ring buffer, wait until ready.

3.20. MP3 - MP3 decoder and player

Files: lib_mp3.h, lib_mp3.c

Config: USE_MP3 (default 0)

The MP3 library is designed to decompress and play sounds in MP3 format. It contains an MP3 frame decoder and MP3 format player via PWMSnd output, either from file or from memory. The MP3Dec library of Copyright (c) 1995-2004 RealNetworks, under the RealNetworks Public Source License, is used for decompression.

MP3 MPEG audio version ID

MP3_VER_MPEG1	MPEG version 1 (ISO/IEC 11172-3)
MP3_VER_MPEG2	MPEG version 2 (ISO/IEC 13818-3)
MP3_VER_MPEG25	MPEG version 2.5 (later version of MPEG 2)

MP3 error codes

ERR_MP3_NONE	no error
ERR_MP3_INDATA_UNDERFLOW	out of data - assume last or truncated frame
ERR_MP3_MAINDATA_UNDERFLOW	not enough data in bit reservoir
ERR_MP3_FREE_BITRATE_SYNC	cannot find sync in free mode
ERR_MP3_OUT_OF_MEMORY	cannot allocate buffers
ERR_MP3_NULL_POINTER	NULL pointer passes instead of decoder handler
ERR_MP3_INVALID_FRAMEHEADER	invalid frame header
ERR_MP3_INVALID_SIDEINFO	invalid side info header
ERR_MP3_INVALID_SCALEFACT	invalid scale factors
ERR_MP3_INVALID_HUFFCODES	invalid Huffman codes
ERR_MP3_INVALID_DEQUANTIZE	invalid dequantize coefficients
ERR_MP3_INVALID_IMDCT	invalid IMDCT
ERR_MP3_INVALID_SUBBAND	invalid subband transform
ERR_MP3_INVALID_FREE	free mode not supported by MP3 player
ERR_MP3_OUTBUF_OVERFLOW	output buffer overflow
ERR_MP3_NOMOUNT	cannot mount SD disk
ERR_MP3_INVALID_FILE	invalid input file, cannot open

MP3 frame info

typedef struct {

```
    int     bitrate;           // bit rate (input bits per second, 0 = "free" mode
    int     nChans;            // number of channels (= 1 or 2)
    int     samrate;           // sample rate (samples per second per one channel)
```

```

    int bitsPerSample;           // number of bits per sample (always 16)
    int outputSamps;            // number of output samples per one frame
    int layer;                  // MPEG layer description (Layer = 1 - 3)
    int version;                // MPEG audio version ID (= MP3_VER_*)
    int size;                   // size of MP3 frame in number of bytes
} sMP3FrameInfo;

```

MP3 handler (pointer to MP3 descriptor sMP3DeclInfo*)

typedef void* HMP3Decoder;

MP3 Decoder

The MP3 decoder functions can be used to decompress individual frames of audio in MP3 format into an output buffer in 16-bit PCM format. The decoder requires working structures of about 24 KB to function. The decoder can create the structures at initialization (using memory allocation with malloc()) and discard them again at termination. Alternatively, the decoder can be passed structures statically allocated in RAM memory - this has the advantage that the structures are reserved for the entire runtime of the program and there is no risk of a memory allocation error.

MP3DECODER_SIZE Size of static structure for MP3 decoder info (24 KB)

HMP3Decoder MP3InitDecoder(void* mp3stat);

mp3stat ... pointer to static buffer MP3DECODER_SIZE, or NULL to auto-allocate

Initialize MP3 decoder. mp3stat is pointer to static buffer for MP3 decoder info structure of size **MP3DECODER_SIZE** bytes, or NULL to auto-allocate required structures. Returns MP3 handler, or NULL on error. If static buffer is used, caller must free it after finalize. If auto-allocation is used, MP3FreeDecoder() must be called after finalize, to free allocated structures.

void MP3FreeDecoder(HMP3Decoder hMP3Decoder);

hMP3Decoder ... MP3 decoder handler

Free MP3 decoder (static descriptor will not be deallocated).

int MP3CheckID3v2(const u8* buf);

buf ... pointer to input buffer (with 10 bytes of data)

Check ID3v2 tag on start of the MP3 (to skip it). Returns size of the Id3v2 tag in bytes, or 0 if ID3v2 tag not found.

int MP3FindSyncWord(const u8* buf, int nBytes);

buf ... pointer to source buffer

nBytes ... size of data in source buffer

Find start of next frame. Returns offset of start of next frame (if >= 0), or -1 if sync word not found.

```
int MP3GetNextFrameInfo(HMP3Decoder hMP3Decoder, sMP3FrameInfo*  
mp3FrameInfo, const u8* buf);
```

hMP3Decoder ... MP3 decoder handler

mp3FrameInfo ... pointer to get MP3 frame info

buf ... pointer to source buffer (with 4 or 6 bytes of data)

Get info from next frame (parse frame header, can be used at start of frame after MP3FindSyncWord() and before MP3Decode()). Returns error code **ERR_MP3_***.

```
void MP3GetLastFrameInfo(HMP3Decoder hMP3Decoder, sMP3FrameInfo*  
mp3FrameInfo);
```

hMP3Decoder ... MP3 decoder handler

mp3FrameInfo ... pointer to get MP3 frame info

Get info from last frame (can be used after MP3Decode()).

```
int MP3Decode(HMP3Decoder hMP3Decoder, const u8** inbuf, int* bytesLeft,  
s16* outbuf, int useSize);
```

hMP3Decoder ... MP3 decoder handler

inbuf ... pointer to pointer to start of frame in input buffer (will be increased, required data size sMP3FrameInfo.size)

bytesLeft ... pointer to bytes left in input buffer (will be decreased)

outbuf ... pointer to output buffer, big enough to hold one frame of decodec PCM samples (sMP3FrameInfo.outputSamps, interleave LRLRLR...)

number of output samples = nGrans * nGranSamps * nChans

useSize ... flag indicating whether MP3 data is normal MPEG format (=0) or reformed "self-contained" frames (=1)

Decode one frame of the MP3. Returns error code **ERR_MP3_***. Use MP3FindSyncWord() to find start of next frame before decode next frame.

MP3 Player

The MP3 player is used to play MP3 audio on the PWMSnd audio output. The sound can be in a file or in memory.

The player can operate in two modes. In poll mode, the MP3Poll function is called repeatedly from the main loop of the program to ensure that the sound is periodically read from the file, decoded and sent to the audio output. This mode is more RAM-intensive because it requires a larger output buffer to provide a 0.1 to 0.2 second service interval.

The second mode is the IRQ mode. In this mode, the MP3Poll function is again called repeatedly from the main program loop to ensure that the audio is periodically read from the file. The decoding does not take place in the poll function, but in the interrupt handler, which is activated whenever the audio output of PWMSnd needs to supply additional data for output. In this mode, a smaller output buffer is sufficient. However, it can only be used in ARM mode of the processor because the decode handler must operate with a low priority interrupt to allow the interrupt for the audio output to run during its operation, but it is not possible to use a lower priority interrupt in RISC-V mode.

The player requires a decoder handler structure of 28 KB, an input buffer of 12 KB, and an output buffer of 70 KB (in poll mode) or 10 KB (in IRQ mode) for its operation, for a total of 110 KB (in poll mode) or 50 KB (in IRQ mode) of RAM. No input buffer is needed when playing audio from memory (the input buffer is the audio that may be in Flash memory). The lowest RAM memory requirements (40 KB) are in IRQ mode (ARM processor) when playing audio from memory, when there is no need to call the MP3Poll function.

The player allows the use of MP3s with constant and variable bitrate. However, it is recommended to use constant bitrate, because with variable bitrate there may be inaccurate calculation of the track length and inaccurate playback position exposure. The player does not support the "free" MP3 file format (this format is supported by the decoder).

Configuration

MP3PLAYER_USEIRQ flag: 1=use IRQ mode (ARM), 0=use POLL mode (RISC-V)

MP3IRQNUM IRQ used to decode next frame (IRQ_SPAREIRQ_0 default)

MP3PLAYER_INSIZE size of input buffer in bytes (12000 default)

To hold 0.2 second of the sound in input buffer: Input bitrate can be in range 8 to 448 kbps, it means 1 to 56 KB/sec. Interval of 0.2 second requires input buffer up to 11 KB.

MP3PLAYER_OUTSIZE size of output buffer in bytes (default 10000 IRQ, 70000 poll)

To hold 0.2 second of the sound in output buffer: Output sample rate is in range 8 kHz to 48 kHz. With stereo 16-bit it is 32 to 192 KB. Interval 0.2 second requires output buffer up to 38 KB. If we limit it to 44kHz, we need buffer 35 KB. 2 buffers are 70 KB. In IRQ mode we need hold 1 frame = max. 2304 samples = 4608 bytes, we need 2 buffers = 10 KB

MP3_CHECK_LOAD 1 = check buffers and CPU loads

ID3v1.1 tag (size 128 bytes)

ID3V1_TITLE_LEN ID3v1.1 title max. length (= 30)

ID3V1_ARTIST_LEN ID3v1.1 artist max. length (= 30)

ID3V1_ALBUM_LEN ID3v1.1 album max. length (= 30)

ID3V1_YEAR_LEN ID3v1.1 year max. length (= 4)

ID3V1_COMMENT_LEN ID3v1.1 comment max. length (= 29)

ID3V1_SIZE total size of ID3v1.1 tag

typedef struct {

char tag[3]; // 0-2: tag "TAG" (3 characters)

char title[ID3V1_TITLE_LEN]; // 3-32: title (30 characters)

char artist[ID3V1_ARTIST_LEN]; // 33-62: artist (30 characters)

char album[ID3V1_ALBUM_LEN]; // 63-92: album (30 characters)

char year[ID3V1_YEAR_LEN]; // 93-96: year (4 characters)

char comment[ID3V1_COMMENT_LEN]; // 97-125: comment (29 chars)

u8 track; // 126: track number

u8 genre; // 127: genre identifier

} sID3v1;

MP3 player descriptor (size 28 KB)

```
typedef struct {

    ...
    const char* filename;           // filename (NULL = buffer mode)
    sFile file;                   // open file
    int filesize;                 // file size (without ID3v1 on the end)
    ...
    volatile int pos;             // index of current frame
    int frames;                   // total number of frames
    u32 frametime;               // time of one frame in [us]
    ...
    sMP3FrameInfo info;          // MP3 frame info
    int bitrateavg;              // average bit rate
    int framesizeavg;            // average frame input size
    int sampnumavg;              // average output samples per frame
    ...
    int timelen;                  // total length in number of seconds
    ...
    sID3v1 id3v1;                // ID3v1 tag
    u8 id3_title_len;            // ID3v1 title length
    u8 id3_artist_len;           // ID3v1 artist length
    u8 id3_album_len;            // ID3v1 album length
    u8 id3_comment_len;          // ID3v1 comment length
    u8 id3_year_len;              // ID3v1 year length
    char id3_title[ID3V1_TITLE_LEN+1]; // ID3v1 title ASCIIZ
    char id3_artist[ID3V1_ARTIST_LEN+1]; // ID3v1 artist ASCIIZ
    char id3_album[ID3V1_ALBUM_LEN+1]; // ID3v1 album ASCIIZ
    char id3_comment[ID3V1_COMMENT_LEN+1]; // ID3v1 comment
    char id3_year[ID3V1_YEAR_LEN+1]; // ID3v1 year ASCIIZ
    // check MP3 load

#ifndef MP3_CHECK_LOAD
    volatile u32 decode_time;      // sum of decode time [us]
    volatile int decode_num;       // count of decoded frames
    volatile int decode_outrem;    // remaining data in output buffer (->outsize)
    volatile int decode_inrem;     // remaining data in input buffer (->insize)
    volatile u32 poll_time;        // sum of poll time [us]
    volatile int poll_len;         // number of polled bytes
#endif
}
```

```
    volatile int    poll_rem;           // remaining data in poll buffer (->insize)
#endif
} sMP3Player;
```

MP3 handler used in IRQ mode

```
sMP3Player* MP3Handler_ID;
```

MP3 genre list

```
MP3_GENRELIST_NUM           number of entries in genre list (= 192)
```

```
const char* MP3GenreList[MP3_GENRELIST_NUM];
```

```
int MP3PlayerInit(sMP3Player* mp3, const char* filename, const u8* inbuf, int
    insize, u8* outbuf, int outsize, int scan);
```

mp3 ... pointer to sMP3Player descriptor

filename ... filename, NULL=play MP3 from memory (inbuf)

inbuf ... pointer to MP3 in memory, or pointer to input bufer for file mode
(recommended size MP3PLAYER_INSIZE)

inspace ... size of input buffer in bytes

outbuf ... pointer to output buffer (must be aligned to u16 or better to u32;
recommended size MP3PLAYER_OUTSIZE)

outsize ... size of output buffer in bytes

scan ... number of frames to scan file on open, -1=scan all file (count frames and
length; recommended value is 100)

Initialize MP3 player. Returns error code ERR_MP3_* (ERR_MP3_OK = 0 if OK). Scanning
frames on opening is used to refine the calculation of audio length and number of frames in
case of variable bitrate. To be precise, it is possible to scan the entire audio (scan = -1), but
this can take a long time. For a constant bitrate, it is sufficient to scan only the beginning
(scan = 1 to 4). The recommended optimal value is 100 frames.

'mp3' is a pointer to the sMP3Player descriptor. It is a 28 KB structure. The descriptor can be
allocated only if needed by the malloc function, but it is preferable to have it as a static
structure, to avoid the possibility of memory allocation errors. Similarly, the input and output
buffers can be allocated with malloc, but it is preferable to have them as static buffers as
well.

```
void MP3PlayerTerm(sMP3Player* mp3);
```

mp3 ... pointer to sMP3Player descriptor

Terminate MP3 player.

```
void MP3Poll(sMP3Player* mp3);
```

mp3 ... pointer to sMP3Player descriptor

Poll loading MP3 file to the player. Should be called from the program loop every 0.1 to 0.2
second.

void MP3Play(sMP3Player* mp3, int chan, Bool rep);

mp3 ... pointer to sMP3Player descriptor

chan ... PWMSnd channel to play

rep ... repeat the sound

Start/continue playing the sound.

Bool MP3Playing(sMP3Player* mp3);

mp3 ... pointer to sMP3Player descriptor

Check if MP3 Player is playing. Wait a few milliseconds after termination for the play buffers to empty.

int MP3GetTimeLen(sMP3Player* mp3);

mp3 ... pointer to sMP3Player descriptor

Get total length in seconds. The value may be inaccurate in the case of MP3 with variable bitrate.

int MP3GetFrameLen(sMP3Player* mp3);

mp3 ... pointer to sMP3Player descriptor

Get total length in frames. The value may be inaccurate in the case of MP3 with variable bitrate.

int MP3GetTimePos(sMP3Player* mp3);

mp3 ... pointer to sMP3Player descriptor

Get current position in seconds.

int MP3GetFramePos(sMP3Player* mp3);

mp3 ... pointer to sMP3Player descriptor

Get current position in frames.

void MP3SeekFrame(sMP3Player* mp3, int pos);

mp3 ... pointer to sMP3Player descriptor

pos ... required position in frames

Seek to position in frames. In the case of MP3 with variable bitrate, the player tries to find the beginning of the next frame, but the exposure may be inaccurate and may lead to interrupted playback.

void MP3SeekTime(sMP3Player* mp3, int pos);

mp3 ... pointer to sMP3Player descriptor

pos ... required position in seconds

Seek to position in seconds. In the case of MP3 with variable bitrate, the player tries to find the beginning of the next frame, but the exposure may be inaccurate and may lead to interrupted playback.

```
void MP3Stop(sMP3Player* mp3);
```

mp3 ... pointer to sMP3Player descriptor

Stop/pause playing sound.

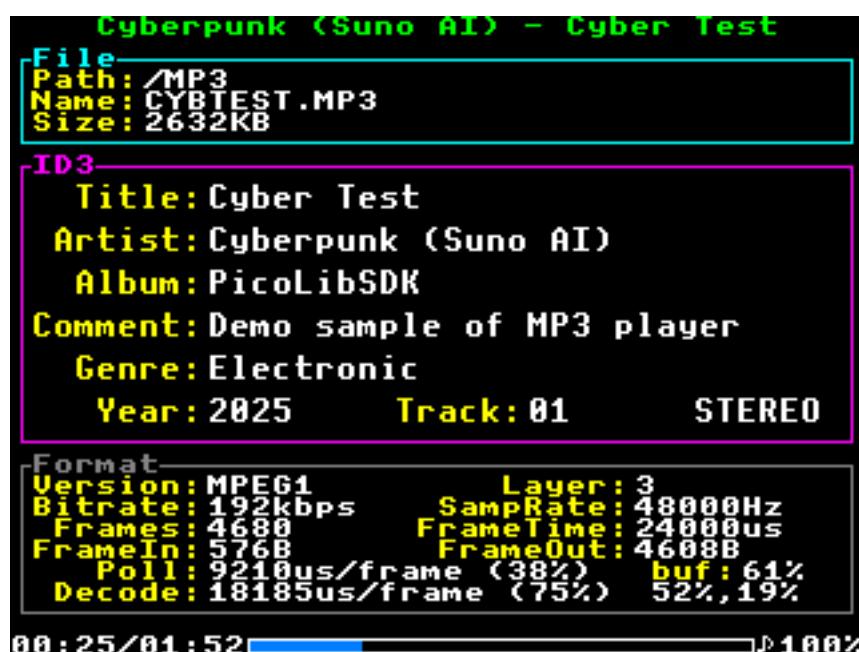
MP3 player application

The PicoLibSDK sample programs include applications for playing MP3 files. The program will offer to play a list of MP3 files from its home folder and from subfolders. When run from the loader, the loader will pass the information to the home folder path to the program. The MP3 program can be saved in any folder, but it is recommended to use the default /MP3 folder if possible.

At startup, a folder list and a list of MP3 songs are displayed. Buttons can be used in the list: **A** = start playback or nesting in a folder, **B** = start playback from the beginning, **Y** = return one folder above or end of program.

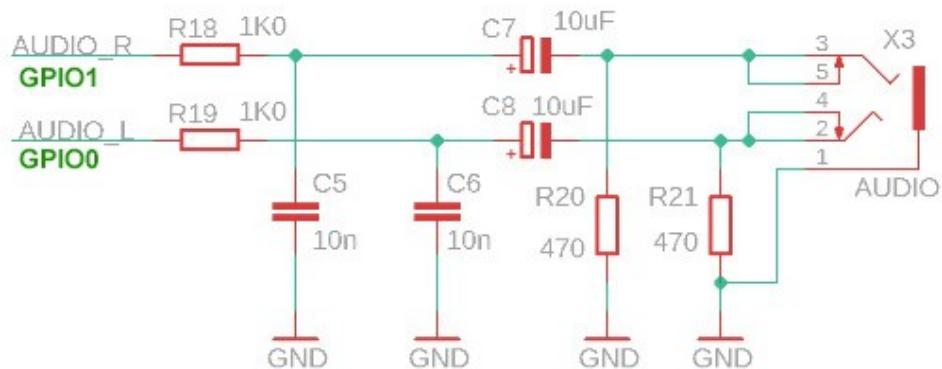
When the song is played, a BMP image with the same name as the MP3 will be displayed. The BMP file must be in RGB565 format (in Photoshop, saving in 16-bit mode cannot be used, the extended RGB565 mode must be used) and 320x240 pixels in size. If the image is not available, only the info page will be displayed.

During playback you can use the keys **Left/Right** = 10 seconds shift, **Up/Down** = volume control, **A** = previous track, **B** = next track, **X** = switch between picture and info page and pause track, **Y** = end of playback.



There are 2 lines at the bottom of the info page serving for debugging information. 'Poll' shows the time it took to load one frame from the SD card. The entry must be lower than the 'FrameTime' entry, otherwise there is a risk of audio dropout. The 'Decode' entry similarly shows the time required to decode one frame. The value must again be lower than the 'FrameTime'. The 'buf' entry displays three values - input buffer filling when reading a frame, input buffer filling when decoding a frame, and output buffer filling when decoding a frame. If the values are low, there is a risk of audio dropout.

The MP3 player application comes in two versions. The basic variant MP3.UF2 plays music in MONO on the Picopad's internal speaker. The MP3_GPO1.UF2 variant plays music in STEREO to the output pins GPIO0 and GPIO1 (Ext connector of the Picopad). The following wiring is recommended to use:



3.21. Print - Formatted Print, printf

Files: lib_print.h, lib_print.c

Config: USE_PRINT (default 1)

The Print library is used for formatted printing to the display, console and other output streams. Formatting is done using a format string and a list of arguments. A detailed description of the format string can be found at the end of this chapter.

u32 StreamPrintArg(sStream* wstr, sStream* rstr, va_list args)

wstr ... write stream (destination buffer)

rstr ... read stream (formating string)

args ... arguments

Stream printing. Returns number of characters (without trailing 0). Central formatting function. This function is not intended for user use, it is used internally within other functions.

u32 MemPrintArg(char* buf, u32 max, const char* fmt, va_list args);

int vsprintf(char* buf, const char* fmt, va_list args);

int vsnprintf(char* buf, size_t max, const char* fmt, va_list args);

buf ... destination buffer

max ... size of destination buffer, including terminating 0

fmt .. formatting string

args ... arguments

Formatted print string into memory buffer, with argument list. Returns number of characters, without terminating 0. The buffer size of the vsprintf() function is implicitly limited to SPRINTF_MAX (= 40) characters (defined in the lib_print.h file).

u32 MemPrint(char* buf, u32 max, const char* fmt, ...);

int sprintf(char* buf, const char* fmt, ...);

int snprintf(char* buf, size_t max, const char* fmt, ...);

buf ... destination buffer

max ... size of destination buffer, including terminating 0

fmt .. formatting string

... ... arguments

Formatted print string into memory buffer, with variadic arguments. Returns number of characters, without terminating 0. The buffer size of the sprintf() function is implicitly limited to SPRINTF_MAX (= 40) characters (defined in the lib_print.h file).

u32 NulPrintArg(const char* fmt, va_list args);

fmt .. formatting string

args ... arguments

Formatted print string into NUL to only get result string length, with argument list. Returns number of characters.

u32 NulPrint(const char* fmt, ...);

fmt .. formatting string

... ... arguments

Formatted print string into NUL to only get result string length, with variadic arguments. Returns number of characters.

u32 UsbPrintArg(const char* fmt, va_list args);

fmt .. formatting string

args ... arguments

Formatted print string to USB serial port, with argument list. Returns number of characters. To output print to the USB serial port it is necessary to set the configuration parameter **USE_USB_STDIO** to 1. You cannot use the USB interface for other purposes at the same time.

u32 UsbPrint(const char* fmt, ...);

fmt .. formatting string

... ... arguments

Formatted print string to USB serial port, with variadic arguments. Returns number of characters. To output print to the USB serial port it is necessary to set the configuration parameter **USE_USB_STDIO** to 1. You cannot use the USB interface for other purposes at the same time.

void UsbPrintChar(char ch);

ch ... character to print

Print character to USB serial port. To output print to the USB serial port it is necessary to set the configuration parameter **USE_USB_STDIO** to 1. You cannot use the USB interface for other purposes at the same time.

u32 UsbPrintText(const char* txt);

txt ... text to print

Print unformatted text to USB serial port. Returns number of characters. To output print to the USB serial port it is necessary to set the configuration parameter **USE_USB_STDIO** to 1. You cannot use the USB interface for other purposes at the same time.

void DrawPrintStart();

Start printing to display console using printf() or Print() function. Printing to display console must be enabled with configuration parameter **USE_DRAW_STDIO** set to 1.

void DrawPrintStop();

Stop printing to display console.

void UsbPrintStart();

Start printing to USB serial port using printf() or Print() function. To output print to the USB serial port it is necessary to set the configuration parameter **USE_USB_STDIO** to 1. You cannot use the USB interface for other purposes at the same time.

void UsbPrintStop();

Stop printing to USB serial port.

void MemPrintStart(char* buf, u32 max);

buf ... pointer to destination buffer

max ... size of destination buffer, including terminating 0

Start printing into memory buffer using printf() or Print() function.

u32 MemPrintLen();

Get current length of text in memory printing buffer.

void MemPrintStop();

Stop printing into memory buffer.

void UartPrintStart();

Start printing to uart-stdio using printf() or Print() function. Printing to uart-stdio must be enabled with configuration parameter **USE_UART_STDIO** set to 1.

void UartPrintStop();

Stop printing to uart-stdio.

void FilePrintStart(sFile* file);

file ... open file descriptor

Start printing to the open file using printf() or Print() function.

void FilePrintStop();

Stop printing to the file.

void PrintChar(char ch);

void printchar(char ch);

int putchar(int ch);

ch ... character to print

Print character to STDIO output print streams. Print streams should be enabled by DrawPrintStart(), UsbPrintStart(), MemPrintStart() etc. The output can run into several output streams at once.

```
int PrintText(const char* txt);
int printtext(const char* txt);
```

txt ... text to print

Print unformatted text to STDIO output print streams. Print streams should be enabled by DrawPrintStart(), UsbPrintStart(), MemPrintStart() etc. The output can run into several output streams at once.

```
int puts(const char* txt);
int puts_raw(const char* txt);
```

txt ... text to print

Print unformatted text with added new-line to STDIO output print streams. Print streams should be enabled by DrawPrintStart(), UsbPrintStart(), MemPrintStart() etc. The output can run into several output streams at once.

```
int PrintArg(const char* fmt, va_list args);
int vprintf(const char* fmt, va_list args);
```

fmt .. formatting string

args ... arguments

Formatted print string to STDIO output print streams, with argument list. Returns number of characters. Print streams should be enabled by DrawPrintStart(), UsbPrintStart(), MemPrintStart() etc. The output can run into several output streams at once. Do not print simultaneously from the program and from the interrupt handler. If you really need it, use SpinLock lock to access it.

```
int Print(const char* fmt, ...);
int printf(const char* fmt, ...);
int print(const char* fmt, ...);
```

fmt .. formatting string

... ... arguments

Formatted print string to STDIO output print streams, with variadic arguments. Returns number of characters. Print streams should be enabled by DrawPrintStart(), UsbPrintStart(), MemPrintStart() etc. The output can run into several output streams at once. Do not print simultaneously from the program and from the interrupt handler. If you really need it, use SpinLock lock to access it.

```
char GetChar();
```

Get character from STDIO input. Returns NOCHAR (=0) if no character. The function gets characters from different sources. Characters are converted to the uniform **CH_*** format (see Conventions). To activate the sources, set the following configuration switches to '1':

USE_USB_HOST_HID ... external USB keyboard (function UsbGetChar())

USE_USB_STDIO ... USB serial port (function UsbDevCdcReadChar())

USE_PICOPAD ... PicoPad board buttons (function KeyChar())

char GetCharWait();

char getchar();

Get character from STUDIO input, wait.

char GetCharUs(u32 us);

us ... timeout in [us]

Get character from stdio with time-out in [us] (returns NOCHAR=0 if no character).

char GetCharMs(u32 ms);

ms ... timeout in [us]

Get character from stdio with time-out in [ms] (returns NOCHAR=0 if no character).

int getchar_timeout_us(u32 us);

us ... timeout in [us]

Get character from stdio with time-out in [us] (returns PICO_ERROR_TIMEOUT=-1 if no character).

int getchar_timeout_ms(u32 ms);

ms ... timeout in [us]

Get character from stdio with time-out in [ms] (returns PICO_ERROR_TIMEOUT=-1 if no character).

void FlushChar();

void stdio_flush();

Flush input characters.

void StdioInit();

bool stdio_init_all();

Initialize Stdio interface (should be called after changing UART clock).

Syntax of Print Formatting String

Format

%[flags][width].[precision][length]specifier

Specifier

%	print character '%'.
b	unsigned integer in BIN format (1011010111).
B	print floating point as capacity in bytes, with unit prefix (3.45KB; alternative '#' with space 3.45 KB).

c	print single-byte character.
d or i	print signed integer in decimal format (1234567).
E or e	print floating point with exponent 'E' or 'e' ('exponent mode', -1.2345e+123). Trailing zeros are not reduced, or are reduced with the '#' flag.
F or f	print floating point with fixed point ('fixed mode', -123.456). Trailing zeros are not reduced, or are reduced with the '#' flag.
G or g	print floating point with automatic format selection (general mode). Trailing zeros are reduced, or are not reduced with the '#' flag.
m	print memory block pointed by argument as HEX bytes with commas. Precision . n is total number of bytes (or .* get from argument; default 8). Width n is number of bytes per row (or * from argument; default 16). Alternative '#' flag is print as decimals. Flag ' print compressed form.
M	print memory block pointed by argument as HEX 16-bit words with commas. Precision . n is total number of words (or .* get from argument; default 8). Width n is number of words per row (or * from argument; default 8). Alternative '#' flag is print as decimals. Flag ' print compressed form.
o	print unsigned integer in octal format.
P or p	print pointer (upper or lower case, with prefix 0x). (0x0200457A).
s	print ASCIIZ text (length can be limited with '.precision').
t	print floating point in technical notation, with exponent 'e' in multiples of 3. Alternative '#' reduce zeroes.
T	print floating point in technical notation, with prefix u, m, k, M, G,... Alternative '#' with space. If number is out of range 10^-30..10^+32, it changes to 't' format with exponent 'e'.
u	print unsigned integer in decimal format.
X or x	print unsigned integer in hexadecimal format, upper or lower case.

Flags

'	grouping separator, integer part of the number is grouped by 3 (or 4) digits, with a separator character '. 'Mm' - print in compressed form XX-XX-XX-...
(space)	number will be preceded by a space instead of a '+' (ignored when '+' flag is specified)
#	alternative form: 'Gg' without end-zero reduction, 'EeFft' with end-zero reduction, 'Pp' without prefix 0x, Xxbo with printing prefix 0x 0b 0, 'BT' with space, 'mM' print decimals.
-	argument will be left-aligned in the 'width' field

	(otherwise the default is right-aligned)
+	the '+' sign is printed on a positive argument (otherwise only the '-' sign is displayed)
0	number is padded with zeros instead of spaces in the 'width' field on the left

Width

width	number is minimum width of the field to display the argument (padded with spaces)
*	the width is not specified in the format string, but is read from the argument. The negative value of the width is interpreted as the character '-' (i.e. a left-aligned argument).

Precision

.precision	number after the point determines the precision: For 'bdiouXx' (integer) determines the minimum number of displayed digits (padded with leading zeros; default 1). For 'EeFf' specifies the number of decimal places (default 6). For 'BGgTt', it is the maximum number of printable significant digits. For 's' (text) it is the maximum number of characters displayed. For 'mM' (memory block)
.*	by specifying * instead of precision, the precision is read from the argument (absolute value is used).

In the case when '.*' is used, the precision is specified by an additional argument of type 'int', which appears before the argument to be converted, but after the argument supplying minimum field width if one is supplied with '*'.

Length

ll	argument is 'long long int' (u64)
-----------	-----------------------------------

Special values of decimal number

infinity ... 'inf' or '-inf' ... 'INF'

NaN ... 'nan' or '-nan' ... 'NAN'

3.22. PWM Sound Output

Files: lib_pwmsnd.h, lib_pwmsnd.c

Config: USE_PWMSND (default 4 if USE_PICOPAD)

PWM modulation provides a simple audio output requiring only minimal hardware (a simple RC filter is sufficient). The audio is in 8-bit or 16-bit PCM format, or IMA ADPCM 4-bit format, mono or stereo. Sounds can be played in multiple audio channels simultaneously - the configuration parameter **USE_PWMSND** specifies the number of audio channels used.

Configuration parameter **PWMSND_GPIO** defines GPIO pin with PWM sound output of left or single channel. Configuration parameter **PWMSND_GPIO_R** defines GPIO pin with PWM sound output of right channel. If **PWMSND_GPIO_R** is not defined, or if has value -1, only one channel is used. Both PWM sound channels must be on the same PWM slice.

The pin configuration can be changed while the program is running. The PWMSndGpio and PWMSndGpioR variables contain the GPIO number definitions for the left and right channels. Both GPIOs must belong to the same PWM slice. If PWMSndGpioR has a value of -1, the MONO output on PWMSndGpio is used. After changing the GPIO numbers, the driver must be reinitialized using PWMSndInit().

Any sound rate can be used. When playing sound, the sound playback speed is specified relative to a reference speed of 22050 Hz. The following constants can be used:

SND SPEED_8K	8000 Hz (= 0.36281f)
SND SPEED_11K	11025 Hz (= 0.50000f)
SND SPEED_12K	12000 Hz (= 0.54422f)
SND SPEED_16K	16000 Hz (= 0.72562f)
SND SPEED_22K	22050 Hz (= 1.00000f)
SND SPEED_24K	24000 Hz (= 1.08844f)
SND SPEED_32K	32000 Hz (= 1.45125f)
SND SPEED_44K	44100 Hz (= 2.00000f)
SND SPEED_48K	48000 Hz (= 2.17687f)

Supported sound formats:

SND FORM_PCM	8-bits unsigned per sample, mono
SND FORM_ADPCM	IMA ADPCM, 4-bit compression, mono
SND FORM_PCM16	no compression, 16-bits signed per sample, mono
SND FORM_PCM_S	8-bits unsigned per sample, stereo
SND FORM_ADPCM_S	IMA ADPCM, 4-bit compression, stereo
SND FORM_PCM16_S	16-bits signed per sample, stereo

void PWMSndInit();

Initialize PWM sound output. The PWMSndInit() function is automatically called during device initialization, from the DeviceInit() function. Function needs not be called again after

changing CLK_SYS system clock. But if playing any sound, SpeedSoundUpdate() should be called to update speed of current sounds. After changing the GPIO numbers, the driver must be reinitialized using PWMSndInit().

void PWMSndTerm();

Terminate PWM sound output.

void StopSoundChan(int chan);

chan ... sound channel (0..)

Stop playing sound of specified channel.

void StopSound();

Stop playing sound of channel 0.

void StopAllSound();

Stop playing sounds of all channels.

int SoundByteToLen(int size, int form);

size ... length of sound in number of bytes (use sizeof(array))

form ... sound format SNDFORM_* (4-bit, 8-bit, 16-bit, mono or stereo)

Convert length of sound in bytes to number of samples. In ADPCM format, the number of samples does not exactly match the actual number of audio samples, the value is slightly higher because the header of each ADPCM block is counted as 4 samples.

void PlaySoundChan(int chan, const void* snd, int size, Bool rep, float speed, float volume, int form, int ext);

chan ... sound channel (0..)

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

rep ... repeat mode SNDREPEAT_*

speed ... speed relative to sample rate 22050 Hz (1=normal, or use SND SPEED_*)

volume ... volume (1=normal)

form ... sound format SNDFORM_*

ext ... format extended data (ADPCM: number of samples per block)

Start playing sound to specified channel. If sound is already playing on that channel, the sound is stopped first. If want to use streaming mode with IRQ, set the 'useirq' and 'irq' entries manually, after calling PlaySoundChan() function.

SNDREPEAT_NO no repeat (or use False)

SNDREPEAT_REPEAT repeat sound (or use True)

SNDREPEAT_STREAM streaming sound

void PlaySound(const void* snd, int size);

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

Start playing sound to channel 0, no repeat sample, with normal speed 1 and normal volume 1. Sound must be PCM 8-bit mono 22050Hz. If sound is already playing on channel 0, the sound is stopped first.

PLAYSOUND(snd)

snd ... pointer to sound data

A macro that plays sound on channel 0, similar to the PlaySound() function except that it determines the length of the data from an array of sound data.

void PlaySoundRep(const void* snd, int len);

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

Start playing sound to channel 0 repeating the sample, with normal speed 1 and normal volume 1. Sound must be PCM 8-bit mono 22050Hz. If sound is already playing on channel 0, the sound is stopped first.

PLAYSOUNDREP(snd)

snd ... pointer to sound data

A macro that plays sound on channel 0 repeatedly, similar to the PlaySoundRep() function except that it determines the length of the data from an array of sound data.

void PlayADPCMChan(int chan, const void* snd, int size, int sampblock);

void PlayADPCM(const void* snd, int size, int sampblock);

chan ... sound channel (0..)

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

sampblock ... number of samples per block

Start playing sound in IMA ADPCM compression, no repeat sample, with normal speed 1 and normal volume 1. Sound must be IMA ADPCM mono 22050Hz. If sound is already playing on the channel, the sound is stopped first.

PLAYADPCM(snd, sampblock)

snd ... pointer to sound data

sampblock ... number of samples per block

A macro that plays sound on channel 0, similar to the PlayADPCM() function except that it determines the length of the data from an array of sound data.

void PlayADPCMRepChan(int chan, const void* snd, int size, int sampblock);
void PlayADPCMRep(const void* snd, int size, int sampblock);

chan ... sound channel (0..)

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

sampblock ... number of samples per block

Start playing sound in IMA ADPCM compression repeating the sample, with normal speed 1 and normal volume 1. Sound must be IMA ADPCM mono 22050Hz. If sound is already playing on the channel, the sound is stopped first.

PLAYADPCMREP(snd, sampblock)

snd ... pointer to sound data

sampblock ... number of samples per block

A macro that plays sound on channel 0 repeatedly, similar to the PlayADPCMRep() function except that it determines the length of the data from an array of sound data.

void SpeedSoundChan(int chan, float speed);

chan ... sound channel (0..)

speed ... new relative speed (1=normal)

Change relative speed of sound playing on specified channel. ADPCM must have speed 1.

void SpeedSound(float speed);

speed ... new relative speed (1=normal)

Change relative speed of sound playing on channel 0. ADPCM must have speed 1.

void VolumeSoundChan(int chan, float volume);

chan ... sound channel (0..)

volume ... new volume (1=normal)

Change volume of sound playing on specified channel.

void VolumeSound(float volume);

volume ... new volume (1=normal)

Change volume of sound playing on channel 0.

Bool PlayingSoundChan(int chan);

chan ... sound channel (0..)

Check if sound is still playing on specified channel.

Bool PlayingSound();

Check if sound is still playing on channel 0.

void SetNextSoundChan(int chan, const void* snd, int size);

chan ... sound channel (0..)

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

Set next repeated sound on specified channel. The specified sound will start playing as a continuation after one pass of the currently playing repeated sound is completed. The sound settings (volume, speed) are not changed. If no sound is playing, a new sound with the previous settings will start playing.

void SetNextSound(const void* snd, int size);

snd ... pointer to sound data

size ... length of sound data in number of bytes (use sizeof(array))

Set next repeated sound on channel 0.

Bool SoundStreamIsEmpty(int chan);

Check if streaming buffer is empty,

void SoundStreamSetNext(int chan, const void* snd, int size);

snd ... pointer to sound

size ... length of sound in number of bytes (use sizeof(array))

Set next streaming buffer with sound data.

void GlobalSoundSetOff();

Global sound set OFF. The function sets global variable GlobalSoundOff to True, mutes all sounds globally and terminates the PWM audio output PWMSndTerm(). All subsequent commands to play the sound will be ignored.

void GlobalSoundSetOn();

Global sound set ON. The function enables the sound output again, sets the GlobalSoundOff global variable to False and initializes the PWM output again with PWMSndInit().

void GlobalVolumeUpdate();

Update sound volume after changing global volume.

3.23. Rand - Random Number Generator

Files: lib_rand.h, lib_rand.c

Config: USE_RAND (default 1)

The random number generator module generates pseudo-random numbers using a 64-bit LCG generator (Linear Congruential Generator). In older versions of the C libraries, a 32-bit generator was used, which had a noticeable lack of randomness in some areas, manifested for example by periodic ripples of the terrain. This 64-bit generator does not have sufficient randomness for use e.g. for data encryption, but its randomness is already sufficient for common use in programs, including the terrain generators.

Each processor core has its own 64-bit seed, allowing fast random number generation without the need to lock access to the seed. When the application starts, the random generator is initialized by the ROSC oscillator, ensuring the randomness of the generator on repeated starts.

u64 RandSeed[2]; Seed of random number generator (for both CPU cores)

u64 RandGet();

Get seed of random number generator for current CPU core.

void RandSet(u64 seed);

seed ... new seed

Set seed of random number generator for current CPU core.

void RandInit();

Initialize seed of random number generator for current CPU core. If the ROSC oscillator is enabled (it is enabled by default), the random generator is initialized randomly according to the ROSC oscillator. Seed will not be fully randomness if ROSC is used as a CPU clock source.

u32 RandShift();

Shift random number generator and return 32-bit random number (for current CPU core).

u8 RandU8();

Generate 8-bit unsigned integer random number.

u16 RandU16();

Generate 16-bit unsigned integer random number.

u32 RandU32();

Generate 32-bit unsigned integer random number.

u64 RandU64();

Generate 64-bit unsigned integer random number.

s8 RandS8();

Generate 8-bit signed integer random number.

s16 RandS16();

Generate 16-bit signed integer random number.

s32 RandS32();

Generate 32-bit signed integer random number.

s64 RandS64();

Generate 64-bit signed integer random number.

float RandFloat();

Generate float random number in range 0 (including) to 1 (excluding).

double RandDouble();

Generate double random number in range 0 (including) to 1 (excluding).

u8 RandU8Max(u8 max);

max ... max. value of generated number

Generate 8-bit unsigned integer random number in range 0 to MAX (including).

u16 RandU16Max(u16 max);

max ... max. value of generated number

Generate 16-bit unsigned integer random number in range 0 to MAX (including).

u32 RandU32Max(u32 max);

max ... max. value of generated number

Generate 32-bit unsigned integer random number in range 0 to MAX (including).

u64 RandU64Max(u64 max);

max ... max. value of generated number

Generate 64-bit unsigned integer random number in range 0 to MAX (including).

s8 RandS8Max(s8 max);

max ... max. value of generated number, can be negative

Generate 8-bit signed integer random number in range 0 to MAX (including).

s16 RandS16Max(s16 max);

max ... max. value of generated number, can be negative

Generate 16-bit signed integer random number in range 0 to MAX (including).

s32 RandS32Max(s32 max);

max ... max. value of generated number, can be negative

Generate 32-bit signed integer random number in range 0 to MAX (including).

s64 RandS64Max(s64 max);

max ... max. value of generated number, can be negative

Generate 64-bit signed integer random number in range 0 to MAX (including).

float RandFloatMax(float max);

max ... max. value of generated number (excluding)

Generate float random number in range 0 (including) to MAX (excluding).

double RandDoubleMax(double max);

max ... max. value of generated number (excluding)

Generate double random number in range 0 (including) to MAX (excluding).

u8 RandU8MinMax(u8 min, u8 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 8-bit unsigned integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

u16 RandU16MinMax(u16 min, u16 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 16-bit unsigned integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

u32 RandU32MinMax(u32 min, u32 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 32-bit unsigned integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

u64 RandU64MinMax(u64 min, u64 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 64-bit unsigned integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

s8 RandS8MinMax(s8 min, s8 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 8-bit signed integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

s16 RandS16MinMax(s16 min, s16 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 16-bit signed integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

s32 RandS32MinMax(s32 min, s32 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 32-bit signed integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

s64 RandS64MinMax(s64 min, s64 max);

min ... min. value of generated number

max ... max. value of generated number

Generate 64-bit signed integer random number in range MIN to MAX (including). If MIN > MAX, then number is generated out of interval.

float RandFloatMinMax(float min, float max);

min ... min. value of generated number

max ... max. value of generated number

Generate float random number in range MIN (including) to MAX (excluding). The order of MIN and MAX does not matter.

double RandDoubleMinMax(double min, double max);

min ... min. value of generated number

max ... max. value of generated number

Generate double random number in range MIN (including) to MAX (excluding). The order of MIN and MAX does not matter.

float RandFloatGauss(float mean, float sigma);

mean ... center if interval (default use 0)

sigma ... width of interval (default use 1)

Generate Gaussian float random number.

double RandDoubleGauss(double mean, double sigma);

mean ... center of interval (default use 0)

sigma ... width of interval (default use 1)

Generate Gaussian double random number

u8 RandTestU8();

u16 RandTestU16();

u32 RandTestU32();

u64 RandTestU64();

s8 RandTestS8();

s16 RandTestS16();

s32 RandTestS32();

s64 RandTestS64();

float RandTestFloat();

float RandTestFloatMinMax(u8 expmin, u8 expmax);

double RandTestDouble();

double RandTestDoubleMinMax(u16 expmin, u16 expmax);

Generate random integer number for testing purposes. The generated number has a random number of valid bits, which makes it preferable to generate small numbers.

float Noise1D(int x, int seed);

x ... X coordinate

seed ... seed of generator

1D coordinate Perlin noise generator (output -1..+1).

float Noise2D(int x, int y, int seed);

x ... X coordinate

y ... Y coordinate

seed ... seed of generator

2D coordinate Perlin noise generator (output -1..+1).

float Noise3D(int x, int y, int z, int seed);

x ... X coordinate

y ... Y coordinate

z ... Z coordinate

seed ... seed of generator

3D coordinate Perlin noise generator (output -1..+1).

float SmoothNoise1D(float x, int scale, int seed);

x ... X coordinate

scale ... scale of coordinate (1...)

seed ... seed of generator

Interpolated 1D Perlin noise (output -1..+1).

float SmoothNoise2D(float x, float y, int scale, int seed);

x ... X coordinate

y ... Y coordinate

scale ... scale of coordinates (1...)

seed ... seed of generator

Interpolated 2D Perlin noise generator (output -1..+1).

3.24. Rect - Rectangle

Files: lib_rect.h, lib_rect.c
Config: USE_RECT (default 1)

The Rect library allows manipulation with rectangular regions - moving, resizing, merging, etc.

Rectangle structure

```
typedef struct {  
    int     x;      // left  
    int     y;      // top  
    int     w;      // width  
    int     h;      // height  
} sRect;
```

void RectSet(sRect* r, int x, int y, int w, int h);

r ... pointer to sRect structure
x ... left X coordinate of rectangle
y ... top Y coordinate of rectangle
w ... width of rectangle
h ... height of rectangle

Set rectangle, using size.

void RectSet2(sRect* r, int x, int y, int x2, int y2);

r ... pointer to sRect structure
x ... left X coordinate of rectangle
y ... top Y coordinate of rectangle
x2 ... right X coordinate of rectangle
y2 ... bottom Y coordinate of rectangle

Set rectangle, using coordinates.

void RectClear(sRect* r);

r ... pointer to sRect structure

Clear rectangle.

void RectMove(sRect* r, int x, int y);

r ... pointer to sRect structure
x ... new left X coordinate of rectangle

y ... new top Y coordinate of rectangle

Move left top corner with preserving size.

void RectMove2(sRect* r, int x2, int y2);

r ... pointer to sRect structure

x2 ... new right X coordinate of rectangle

y2 ... new bottom Y coordinate of rectangle

Move right bottom corner with changing size.

void RectShift(sRect* r, int dx, int dy);

r ... pointer to sRect structure

dx ... relative change left X coordinate of rectangle

dy ... relative change top Y coordinate of rectangle

Relative shift rectangle. Move left top corner with preserving size.

void RectResize(sRect* r, int w, int h);

r ... pointer to sRect structure

w ... new width of rectangle

h ... new height of rectangle

Set size of rectangle. Shift right bottom corner, preserve left top corner.

void RectCopy(sRect* dst, const sRect* src);

dst ... pointer to destination sRect structure

src ... pointer to source sRect structure

Copy rectangle.

void RectCopyShift(sRect* dst, const sRect* src, int dx, int dy);

dst ... pointer to destination sRect structure

src ... pointer to source sRect structure

dx ... relative change left X coordinate of rectangle

dy ... relative change top Y coordinate of rectangle

Copy rectangle and shift relative.

Bool RectIsValid(const sRect* r);

r ... pointer to sRect structure

Check if rectangle is valid (size must be > 0).

Bool RectHit(const sRect* r, int x, int y);

r ... pointer to sRect structure

x ... point X coordinate

y ... point Y coordinate

Check if point hits the rectangle.

Bool RectOverlap(const sRect* r, const sRect* r2);

r ... pointer to 1st sRect structure

r2 ... pointer to 2nd sRect structure

Check if rectangle partially overlaps another rectangle.

Bool RectInside(const sRect* rin, const sRect* rout);

rin ... pointer to 1st sRect structure (inner rectangle)

rout ... pointer to 2nd sRect structure (outer rectangle)

Check if rectangle 'rin' is whole inside another rectangle 'rout'.

Bool RectEqu(const sRect* r, const sRect* r2);

r ... pointer to 1st sRect structure

r2 ... pointer to 2nd sRect structure

Check if rectangles are equal.

void RectClip(sRect* rin, const sRect* rout);

rin ... pointer to 1st sRect structure (inner rectangle)

rout ... pointer to 2nd sRect structure (outer rectangle)

Clip rectangle 'rin' by another rectangle 'rout'.

void RectMerge(sRect* dst, const sRect* src);

dst ... pointer to destination sRect structure

src ... pointer to source sRect structure

Merge rectangle 'dst' with another rectangle 'src'.

3.25. Ring Buffer

Files: lib_ring.h, lib_ring.c

Config: USE_RING (default 1)

The ring buffer serves as a storage space while transferring data between the interrupt handler and the main application, or between processor cores. It allows access locking and transfer activation.

typedef void (*pRingForce(sRing*);

Prototype of wake-up function to force handler to send or receive data.

Ring buffer (size 32 bytes)

typedef struct {

```
    u8*          buf;           // pointer to data buffer
    u32          size;          // size of buffer (number of bytes)
    volatile u32  write;         // write offset into buffer
    volatile u32  read;          // read offset from buffer
    int           spinlock;       // index of spinlock (-1 = not used)
    pRingForce   force_send;     // wake-up function to send data (NULL = none)
    pRingForce   force_recv;     // wake-up function to receive data (NULL = none)
    u32          cookie;         // user data
} sRing;
```

RING(name,size,spinlock, force_send,force_recv,cookie);

name ... name of the ring buffer

size ... size of data buffer in bytes

spinlock ... index of spinlock to lock access (-1 if not used)

force_send ... wake-up function to force handler to send data (NULL = none)

force_recv ... wake-up function to force handler to receive data (NULL = none)

cookie ... user data

Macro to create ring buffer in *.c file with static buffer as global variable. This will create initialized global variable with given name. To declare it in *.h file, use:

extern sRing name;

**void RingInit(sRing* ring, u8* buf, u32 size, int spinlock, pRingForce
force_send, pRingForce force_recv, u32 cookie);**

ring ... pointer to ring buffer

buf ... data buffer (of size 'size' bytes)

size ... size of data buffer in bytes

spinlock ... index of spinlock to lock access (-1 if not used)

force_send ... wake-up function to force handler to send data (NULL = none)

force_recv ... wake-up function to force handler to receive data (NULL = none)

cookie ... user data

Initialize ring buffer. Not needed with static global ring buffer, created with RING() macro.

void RingWriteUpdate(sRing* ring);

ring ... pointer to ring buffer

Update ring buffer send transmission - force handler to send data.

void RingReadUpdate(sRing* ring);

ring ... pointer to ring buffer

Update ring buffer receive transmission - force handlers to receive data.

void RingClear(sRing* ring);

ring ... pointer to ring buffer

Clear ring buffer. Function can be called from inside the locked section.

u32 RingFree(sRing* ring);

ring ... pointer to ring buffer

Get free space for writing. Function can be called from inside the locked section.

u32 RingNum(sRing* ring);

ring ... pointer to ring buffer

Get bytes available for reading. Function can be called from inside the locked section.

Bool RingWriteReady(sRing* ring, u32 len);

ring ... pointer to ring buffer

len ... required number of bytes

Check if ring buffer is ready to write required number of bytes. Function can be called from inside the locked section.

Bool RingReadReady(sRing* ring, u32 len);

ring ... pointer to ring buffer

len ... required number of bytes

Check if ring buffer is ready to read required number of bytes. Function can be called from inside the locked section.

Bool RingWrite(sRing* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to the data

len ... length of the data

Write whole data into ring buffer, without waiting. Returns False if buffer has not free space. Data is written only if there is enough space in the ring buffer. Cannot write only part of the data.

u32 RingWriteData(sRing* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to the data

len ... length of the data

Write part of data into ring buffer, without waiting. Returns number of written bytes.

Bool RingWrite8(sRing* ring, u8 data);

ring ... pointer to ring buffer

data ... 8-bit data to write

Write 8-bit data into ring buffer, without waiting. Returns False on buffer overflow.

Bool RingWrite16(sRing* ring, u16 data);

ring ... pointer to ring buffer

data ... 16-bit data to write

Write 16-bit data into ring buffer, without waiting. Returns False on buffer overflow. Only the full 16 bits of data are written, not a partial write.

Bool RingWrite32(sRing* ring, u32 data);

ring ... pointer to ring buffer

data ... 32-bit data to write

Write 32-bit data into ring buffer, without waiting. Returns False on buffer overflow. Only the full 32 bits of data are written, not a partial write.

void RingWriteWait(sRing* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to the data

len ... length of the data

Write data into ring buffer, wait until ready.

void RingWrite8Wait(sRing* ring, u8 data);

ring ... pointer to ring buffer

data ... 8-bit data to write

Write 8-bit data into ring buffer, wait until ready.

void RingWrite16Wait(sRing* ring, u16 data);

ring ... pointer to ring buffer

data ... 16-bit data to write

Write 16-bit data into ring buffer, wait until ready.

void RingWrite32Wait(sRing* ring, u32 data);

ring ... pointer to ring buffer

data ... 32-bit data to write

Write 32-bit data into ring buffer, wait until ready.

Bool RingRead(sRing* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read whole data from ring buffer, without waiting. Return False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

u32 RingReadData(sRing* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read part of data from ring buffer, without waiting. Returns number of read bytes.

char RingReadChar(sRing* ring);

ring ... pointer to ring buffer

Read 1 character from ring buffer, without waiting. Returns 0 if buffer has not enough data.

Bool RingRead8(sRing* ring, u8* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 8-bit data from ring buffer, without waiting. Returns False if buffer has not enough data.

Bool RingRead16(sRing* ring, u16* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 16-bit data from ring buffer, without waiting. Returns False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

Bool RingRead32(sRing* ring, u32* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 32-bit data from ring buffer, without waiting. Returns False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

void RingReadWait(sRing* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read data from ring buffer, wait until ready.

u8 RingRead8Wait(sRing* ring);

ring ... pointer to ring buffer

Read 8-bit data from ring buffer and wait until ready.

u16 RingRead16Wait(sRing* ring);

ring ... pointer to ring buffer

Read 16-bit data from ring buffer and wait until ready.

u32 RingRead32Wait(sRing* ring);

ring ... pointer to ring buffer

Read 32-bit data from ring buffer and wait until ready.

void StreamWriteRingInit(sStream* str, sRing* ring);

str ... data stream

ring ... pointer to ring buffer

Initialize stream to write to ring buffer.

u32 RingPrintArg(sRing* ring, const char* fmt, va_list args);

ring ... pointer to ring buffer

fmt .. format string

args ... arguments

Formatted print string into ring buffer, with argument list.

u32 RingPrint(sRing* ring, const char* fmt, ...);

ring ... pointer to ring buffer

fmt .. format string

... ... arguments

Formatted print string into ring buffer, with variadic arguments.

3.26. RingRX - Ring Buffer with DMA in Receiver Mode

Files: lib_ringrx.h, lib_ringrx.c

Config: USE_RINGRX (default 0)

RingRX is a special variant of the ring buffer, using DMA transfer during data receiving. DMA transfer is used in ring buffer mode - the buffer for DMA must be a power of 2 (max. 32 KB) and must start at an address that is a multiple of the buffer size. The buffer can be allocated either by defining static buffer with the 'aligned...' attribute (this can create an inefficient unused area of data), or at a fixed address in the *.ld linker script, or allocating a 2x larger memory block using malloc() and aligning the DMA buffer address inside it (unused data area will remain).

Usage:

- define required resources in the config_def.h file - spinlock index and DMA channel. Spinlock is not required (can be "-1") if buffer is not used in the second CPU core.
- create ring buffer in *.c file using RIBGRX. Buffer must be aligned to its size.
- declare ring buffer in *.h file
- start DMA transfer with function RingRxStart()
- extract data from the buffer using functions RingRxRead*(). Data must be extracted from the buffer fast enough to prevent buffer overflows.

DMA RingRX buffer

```
typedef struct {
    u8*          buf;        // pointer to data buffer (address and size
                           // must be aligned to '1 << order' bytes)
    u32          size;       // size of buffer in bytes (= 1 << order; max. 32 KB)
    volatile u32  read;      // read offset from the buffer
    volatile u32* port;     // pointer to periphery data register (data port)
    int          spinlock;   // index of spinlock (-1 = not used)
    u8           order;      // order of buffer size 1..15 (size = 1 << order)
    u8           dma;        // DMA channel
    u8           dreq;       // DMA channel data request DREQ_*
    u8           entry;      // entry size DMA_SIZE_*
} sRingRx;
```

RINGRX(name, port, spinlock, order, dma, dreq, entry);

name ... name of the ring buffer

port ... pointer to periphery data register (data port)

spinlock ... index of spinlock (-1 = not used)

order ... order of buffer size 1..15 (size = 1 << order)

dma ... DMA channel
dreq ... DMA channel data request **DREQ_***
entry ... entry size **DMA_SIZE_***

Macro to create DMA ring buffer in *.c file with static buffer as global variable. This will create initialized global variable with given name. Buffers start address must be aligned to '1 << order' bytes (use attribute aligned...). To declare it in *.h file, use:

extern sRingRx name;

void RingRxInit(sRingRx* ring, u8* buf, volatile u32* port, int spin, u8 order, u8 dma, u8 dreq, u8 entry);

ring ... pointer to the ring buffer
buf ... data buffer (size '1 << order' bytes; start address aligned to '1 << order')
port ... pointer to periphery data register (data port)
spinlock ... index of spinlock (-1 = not used)
order ... order of buffer size 1..15 (size = 1 << order)
dma ... DMA channel
dreq ... DMA channel data request **DREQ_***
entry ... entry size **DMA_SIZE_***

Initialize DMA ring buffer. Not needed with static global ring buffer, created with RINGRX() macro.

void RingRxStart(sRingRx* ring);

ring ... pointer to the ring buffer

Start DMA read transfer. Data will be read from the port permanently, without checking for buffer overflow.

The RingRxStart() function must be started first before using any of the following functions.

void RingRxPause(sRingRx* ring);

ring ... pointer to the ring buffer

Pause DMA read transfer.

void RingRxUnpause(sRingRx* ring);

ring ... pointer to the ring buffer

Unpause DMA read transfer.

void RingRxStop(sRingRx* ring);

ring ... pointer to the ring buffer

Stop DMA read transfer.

u32 RingRxFree(sRingRx* ring);

ring ... pointer to the ring buffer

Get free space in buffer.

u32 RingRxNum(sRingRx* ring);

ring ... pointer to the ring buffer

Get bytes available for reading.

Bool RingRxReady(sRingRx* ring, u32 len);

ring ... pointer to the ring buffer

len ... required number of bytes

Check if ring buffer is ready to read required number of bytes.

Bool RingRxRead(sRingRx* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read whole data from ring buffer, without waiting. Returns False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

u32 RingRxReadData(sRingRx* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read part of data from ring buffer, without waiting. Returns number of read bytes.

Bool RingRxRead8(sRingRx* ring, u8* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 8-bit data from ring buffer, without waiting. Returns False if buffer has not enough data.

Bool RingRxRead16(sRingRx* ring, u16* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 16-bit data from ring buffer, without waiting. Returns False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

Bool RingRxRead32(sRingRx* ring, u32* dst);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

Read 32-bit data from ring buffer, without waiting. Returns False if buffer has not enough data. Data is read only if there is enough data in the ring buffer. Cannot read only part of the data.

void RingRxReadWait(sRingRx* ring, void* dst, u32 len);

ring ... pointer to ring buffer

dst ... pointer to buffer to receive data

len ... length of the data

Read data from ring buffer, wait until ready.

u8 RingRxRead8Wait(sRingRx* ring);

ring ... pointer to ring buffer

Read 8-bit data from ring buffer and wait until ready.

u16 RingRxRead16Wait(sRingRx* ring);

ring ... pointer to ring buffer

Read 16-bit data from ring buffer and wait until ready.

u32 RingRxRead32Wait(sRingRx* ring);

ring ... pointer to ring buffer

Read 32-bit data from ring buffer and wait until ready.

3.27. RingTX - Ring Buffer with DMA in Transmitt Mode

Files: lib_ringtx.h, lib_ringtx.c
Config: USE_RINGTX (default 0)

RingTX is a special variant of the ring buffer, using DMA transfer during data transmitting. DMA transfer is used in ring buffer mode - the buffer for DMA must be a power of 2 (max. 32 KB) and must start at an address that is a multiple of the buffer size. The buffer can be allocated either by defining static buffer with the 'aligned...' attribute (this can create an inefficient unused area of data), or at a fixed address in the *.ld linker script, or allocating a 2x larger memory block using malloc() and aligning the DMA buffer address inside it (unused data area will remain).

Usage:

- define required resources in the config_def.h file - spinlock index and DMA channel. Spinlock is not required (can be "-1") if buffer is not used in the second CPU core.
- create ring buffer in *.c file using RIBGTX. Buffer must be aligned to its size.
- declare ring buffer in *.h file
- start DMA transfer with function RingRxStart()
- run RingTxPrep() to prepare DMA channel (not needed if you use the RingTxStart() function immediately).

You can now proceed in "stream mode", which is a continuous flow of data, or "single mode", where only written data is sent.

To use fast stream mode (continuous transmission):

- fill the buffer with some data to be sent first
- start DMA stream transfer with function RingTxStart()
- repeatedly write more data to the buffer. The buffer must not become empty, the transmit function does not wait for data.

To use single mode (only written data is sent):

- Send data using RingTxWrite*() functions.
- After sending block of data, call the RingTxSend() function to ensure that the remaining data in the buffer is sent (not necessary after RingTxWriteSend()).

DMA RingTX buffer

```
typedef struct {
    u8*          buf;           // pointer to data buffer (address and size
                                // must be aligned to '1 << order' bytes)
    u32          size;          // size of buffer in bytes (= 1 << order; max. 32 KB)
    volatile u32  write;         // write offset in the buffer
    volatile u32* port;         // pointer to periphery data register (data port)
```

```

int      spinlock;    // index of spinlock (-1 = not used)
u8       order;       // order of buffer size 1..15 (size = 1 << order)
u8       dma;         // DMA channel
u8       dreq;        // DMA channel data request DREQ_*
u8       entry;       // entry size DMA_SIZE_*
} sRingTx;

```

RINGTX(name, port, spinlock, order, dma, dreq, entry);

name ... name of the ring buffer
port ... pointer to periphery data register (data port)
spinlock ... index of spinlock (-1 = not used)
order ... order of buffer size 1..15 (size = 1 << order)
dma ... DMA channel
dreq ... DMA channel data request **DREQ_***
entry ... entry size **DMA_SIZE_***

Macro to create DMA ring buffer in *.c file with static buffer as global variable. This will create initialized global variable with given name. Buffers start address must be aligned to '1 << order' bytes (use attribute aligned...). To declare it in *.h file, use:

```
extern sRingTx name;
```

void RingTxInit(sRingTx* ring, u8* buf, volatile u32* port, int spinlock, u8 order, u8 dma, u8 dreq, u8 entry);

ring ... pointer to the ring buffer
buf ... data buffer (size '1 << order' bytes; start address aligned to '1 << order')
port ... pointer to periphery data register (data port)
spinlock ... index of spinlock (-1 = not used)
order ... order of buffer size 1..15 (size = 1 << order)
dma ... DMA channel
dreq ... DMA channel data request **DREQ_***
entry ... entry size **DMA_SIZE_***

Initialize DMA ring buffer. Not needed with static global ring buffer, created with RINGTX() macro.

void RingTxStart(sRingTx* ring);

ring ... pointer to the ring buffer

Start DMA write transfer in fast stream mode (continuous transmission).

void RingTxPrep(sRingTx* ring);

ring ... pointer to the ring buffer

Prepare DMA ring buffer if created with RINGTX() macro and not with RingTxInit() function.
Must be called before any other function (not needed before RingTxStart() function).

**The RingTxStart() or RingTxPrep() function must be started first
before using any of the following functions.**

void RingTxPause(sRingTx* ring);

ring ... pointer to the ring buffer

Pause DMA write transfer.

void RingTxUnpause(sRingTx* ring);

ring ... pointer to the ring buffer

Unpause DMA write transfer.

void RingTxStop(sRingTx* ring);

ring ... pointer to the ring buffer

Stop DMA write transfer.

void RingTxSend(sRingTx* ring);

ring ... pointer to the ring buffer

Send remaining data in the buffer (use in single-run mode, do not use in stream mode).

u32 RingTxFree(sRingTx* ring);

ring ... pointer to the ring buffer

Get free space for writing.

u32 RingTxNum(sRingTx* ring);

ring ... pointer to the ring buffer

Get bytes in buffer.

Bool RingTxReady(sRingTx* ring, u32 len);

ring ... pointer to the ring buffer

len ... required number of bytes

Check if ring buffer is ready to write required number of bytes.

Bool RingTxWrite(sRingTx* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to buffer with data to send

len ... length of the data

Write whole data into ring buffer, without waiting. Returns False if buffer has not free space. Data is written only if there is enough space in the ring buffer. Cannot write only part of the data.

Bool RingTxWriteData(sRingTx* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to buffer with data to send

len ... length of the data

Write part of data into ring buffer, without waiting. Returns number of written bytes.

Bool RingTxWrite8(sRingTx* ring, u8 data);

ring ... pointer to ring buffer

data ... data to send

Write 8-bit data into ring buffer, without waiting. Data is lost if buffer is full - returns False on buffer overflow.

Bool RingTxWrite16(sRingTx* ring, u16 data);

ring ... pointer to ring buffer

data ... data to send

Write 16-bit data into ring buffer, without waiting. Data is lost if buffer is full - returns False on buffer overflow. Only the full 16 bits of data are written, not a partial write.

Bool RingTxWrite32(sRingTx* ring, u32 data);

ring ... pointer to ring buffer

data ... data to send

Write 32-bit data into ring buffer, without waiting. Data is lost if buffer is full - returns False on buffer overflow. Only the full 32 bits of data are written, not a partial write.

void RingTxWriteWait(sRingTx* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to buffer with data to send

len ... length of the data

Write data into ring buffer, wait until ready.

void RingTxWrite8Wait(sRingTx* ring, u8 data);

ring ... pointer to ring buffer

data ... data to send

Write 8-bit data into ring buffer, wait until ready.

void RingTxWrite16Wait(sRingTx* ring, u16 data);

ring ... pointer to ring buffer

data ... data to send

Write 16-bit data into ring buffer, wait until ready.

void RingTxWrite32Wait(sRingTx* ring, u32 data);

ring ... pointer to ring buffer

data ... data to send

Write 32-bit data into ring buffer, wait until ready.

void RingTxWriteSend(sRingTx* ring, const void* src, u32 len);

ring ... pointer to ring buffer

src ... pointer to buffer with data to send

len ... length of the data

Write data into ring buffer and send remaining data.

void StreamWriteRingTxInit(sStream* str, sRingTx* ring);

str ... data stream

ring ... pointer to ring buffer

Initialize stream to write to ring buffer.

u32 RingTxPrintArg(sRingTx* ring, const char* fmt, va_list args);

ring ... pointer to ring buffer

fmt .. format string

args ... arguments

Formatted print string into ring buffer, with argument list.

u32 RingTxPrint(sRingTx* ring, const char* fmt, ...);

ring ... pointer to ring buffer

fmt .. format string

... ... arguments

Formatted print string into ring buffer, with variadic arguments.

3.28. SD Card

Files: lib_sd.h, lib_sd.c

Config: USE_SD (default 1 if USE_PICOPAD)

SD Card is an interface for accessing SD card or microSD card via the SPI interface. It provides SD card access for the FAT file system.

Configuration

SD_SPI	(=1)	SD card SPI interface
SD_RX	(=12)	SD card RX (MISO input), to DATA_OUT pin 7
SD_CS	(=13)	SD card CS, to CS pin 1
SD_SCK	(=10)	SD card SCK, to SCLK pin 5
SD_TX	(=11)	SD card TX (MOSI output), to DATA_IN pin 2
SD_BAUDLOW	(=250000)	SD card low baud speed (to initialize connection)
SD_BAUD	(=2000000)	SD card baud speed (max. 7-12 Mbit/s)
SECT_SIZE	(=512)	sector size

SD card type

SD_NONE	(=0)	unknown type
SD_MMC	(=1)	MMC
SD_SD1	(=2)	SD v1
SD_SD2	(=3)	SD v2
SD_SDHC	(=4)	SDHC, block device

`u8 SDType;`

Current SD card type `SD_NONE`,...

const char* GetSDName();

Get SD card type name (“MMC”, “SDv1”,...).

Bool SDConnect();

Connect to SD card after inserting. Returns False on error.

void SDDisconnect();

Disconnect SD card.

Bool SDReadSect(u32 sector, u8* buffer);

`sector` ... sector number

buffer ... data buffer

Read one 512-byte sector from SD card. Returns False on error.

Bool SDWriteSect(u32 sector, const u8* buffer);

sector ... sector number

buffer ... data buffer

Write one 512-byte sector to SD card. Returns False on error.

u32 SDMediaSize();

Get media size, in number of 512-byte sectors. Returns 0 on error.

Bool DiskValid();

Check if disk is valid (**SDType** != **SD_NONE**).

void SDInit();

Initialize SD card interface. Must be re-initialized after changing system clock.

void SDTerm();

Terminate SD card interface.

3.29. Stream - Data Stream

Files: lib_stream.h, lib_stream.c

Config: USE_STREAM (default 1)

A stream library is an interface for data streams, allowing different data modules to be connected to each other. It is used, for example, to output texts to different devices from the printf() function.

Data stream

```
typedef struct {  
    // variables are used by callback functions, should not be used by caller  
    u32          pos;      // object seek position  
    u32          size;     // object size  
    u32          cookie;   // object cookie (base address, file handle)  
    // caller interacts with the stream through callback functions  
    pStreamOpen   open;     // open stream (start transactions; NULL=none)  
    pStreamClose  close;    // close stream (stop transactions; NULL=none)  
    pStreamWrite  write;    // write data (returns number of bytes written)  
    pStreamRead   read;    // read data (returns number of bytes read)  
} sStream;
```

typedef void (*pStreamOpen(sStream* str);

str ... pointer to the stream

Prototype of callback function to open stream (start transactions).

typedef void (*pStreamClose(sStream* str);

str ... pointer to the stream

Prototype of callback function to close stream (stop transactions and flush buffers).

typedef u32 (*pStreamWrite(sStream* str, const void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Prototype of callback function to write data to the stream. Returns number of bytes written.

typedef u32 (*pStreamRead(sStream* str, void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Prototype of callback function to read data from the stream. Returns number of bytes read.

u32 StreamWrite0(sStream* str, const void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Default callback function - nul write stream function.

u32 StreamRead0(sStream* str, void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Default callback function - nul read stream function.

void Stream0Init(sStream* str);

str ... pointer to the stream

Initialize nul stream. Used to only get number of characters.

u32 StreamWriteBuf(sStream* str, const void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Callback function - write data to memory buffer.

u32 StreamReadBuf(sStream* str, void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Callback function - read data from memory buffer.

void StreamReadBuflInit(sStream* str, const void* buf, u32 num);

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Initialize stream to read from memory buffer.

```
void StreamWriteBufInit(sStream* str, void* buf, u32 num);
```

str ... pointer to the stream

buf ... pointer to data buffer

num ... number of bytes

Initialize stream to write to memory buffer.

3.30. Text Strings

Files: lib_text.h, lib_text.c, lib_textnum.c

Config: USE_TEXT (default 1)

Text module allows working with text strings - merging, splitting, searching, converting texts. It requires a memory allocator (MemAlloc(), MemResize(), MemFree()) and system locks (SafeInc(), SafeDec()). A text object is represented by a pText pointer pointing to sTextData text data. The text data can be shared between multiple text objects and contain a reference counter of the object's owners. When passing text to another function, the entire text is not passed, but only the pointer to the text data, and the reference counter of the owners is updated. The text data is discarded only when all owners release it and the reference counter reaches zero.

Text string data

```
typedef struct {
    volatile u32    ref;      // reference counter (number of owners, 0=nobody)
    s32             len;     // length of text string
    char            data[1];  // ASCIIZ text string (including terminating zero)
} sTextData;

typedef sTextData* pText;    // Text object (pointer to text data)

sTextData EmptyTextData;    // Empty text string data.
pText EmptyText;           // Empty text
```

ASCIIZ string support functions

int StrLen(const char* text);

text ... ASCII text string (can be NULL)

Get length of ASCIIZ text string.

int StrComp(const char* text1, const char* text2);

text1 ... 1st ASCII text string (cannot be NULL)

text2 ... 2nd ASCII text string (cannot be NULL)

Compare ASCIIZ text strings. Returns 0=equal, or returns character difference.

s32 StrToInt(const char* text);

text ... ASCII text string (can be NULL)

Convert ASCIIZ text to signed integer number. Number can be in HEX format “0xN” or “\$N”. No overflow check. Big number can overflow to negative number.

u32 StrToInt(const char* text);

text ... ASCII text string (can be NULL)

Convert ASCIIZ text to unsigned integer number. Number can be in HEX format “0xN” or “\$N”. No overflow check.

s64 StrToInt64(const char* text);

text ... ASCII text string (can be NULL)

Convert ASCIIZ text to 64-bit signed integer number. Number can be in HEX format “0xN” or “\$N”. No overflow check. Big number can overflow to negative number.

u64 StrToInt64(const char* text);

text ... ASCII text string (can be NULL)

Convert ASCIIZ text to 64-bit unsigned integer number. Number can be in HEX format “0xN” or “\$N”. No overflow check.

float StrToFloat(const char* text);

text ... ASCII text string (cannot be NULL)

Convert ASCIIZ text to float number.

Bool CheckInfF(float num);

Bool CheckSInfF(float num);

Bool CheckNanF(float num);

Bool CheckSNanF(float num);

num ... number

Check special float numbers - INF and NAN.

double StrtoDouble(const char* text);

text ... ASCII text string (cannot be NULL)

Convert ASCIIZ text to double number.

Bool CheckInfD(double num);

Bool CheckSInfD(double num);

Bool CheckNanD(double num);

Bool CheckSNanD(double num);

num ... number

Check special double numbers - INF and NAN.

Internal functions (not for public use)

sTextData* TextDataNew(s32 len);

len ... length of data (without trailing 0)

Create new text data. Returns NULL on memory error.

sTextData* TextDataResize(sTextData* data, s32 len);

data ... pointer to text data

len ... new length of data (without trailing 0)

Resize text data (ref must be = 1). Returns new pointer or NULL on memory error.

void TextAttach(pText* text, sTextData* data);

text ... pointer to text object

data ... pointer to text data

Attach text data to text object. Object must not be attached.

void TextDetach(pText* text);

text ... pointer to text object

Detach (and destroy if needed) text data from text object.

Bool TextCopyWrite(pText* text);

text ... pointer to text object

Appropriate text string data before write. Returns False on memory error.

Base control functions

void TextInit(pText* text);

text ... pointer to text object

Initialize text object - attach empty text. Object must not be attached).

void TextTerm(pText* text);

text ... pointer to text object

Uninitialize text object - detach text data.

u32 TextRef(const pText* text);

text ... pointer to text object

Get text data reference counter.

s32 TextLen(const pText* text);

text ... pointer to text object

Get text length (without trailing zero).

const char* TextPtr(const pText* text);

text ... pointer to text object

Get pointer to ASCIIZ text.

Bool TextSetLen(pText* text, s32 len);

text ... pointer to text object

len ... new length of data (without trailing 0)

Set new length of the text (appropriates text). Returns False on memory error.

void TextEmpty(pText* text);

text ... pointer to text object

Empty (clear) text string.

Bool TextIsEmpty(const pText* text);

text ... pointer to text object

Check if text string is empty.

Bool TextIsNotEmpty(const pText* text);

text ... pointer to text object

Check if text string is not empty.

int TextWrite(const pText* text, char* buf, int maxlen);

text ... pointer to text object

buf ... destination buffer

maxlen ... max. length of text

Write text into buffer without terminating zero. Returns text length.

int TextWriteZ(const pText* text, char* buf, int maxlen);

text ... pointer to text object

buf ... destination buffer

maxlen ... max. length of text without terminating zero

Write text into buffer with terminating zero. Returns text length.

void TextCheckNul(pText* text);

text ... pointer to text object

NUL correction - check and truncate text if contains NUL character.

int TextComp(const pText* text1, const pText* text2);

text1 ... pointer to 1st text object

text2 ... pointer to 2nd text object

Compare text stings. Returns 0=equal, or returns character difference.

Character in text string

Bool TextIsValid(const pText* text, s32 inx);

text ... pointer to text object

inx .. index of the character (0..)

Check if character index is valid.

Bool TextIsNotValid(const pText* text, s32 inx);

text ... pointer to text object

inx .. index of the character (0..)

Check if character index is not valid.

char TextAt(const pText* text, s32 inx);

text ... pointer to text object

inx .. index of the character (0..)

Get character at given position. Returns 0 if index is invalid.

char TextFirst(const pText* text);

text ... pointer to text object

Get first character of text string. Returns 0 if text string is empty.

char TextLast(const pText* text);

text ... pointer to text object

Get last character of text string. Returns 0 if text string is empty.

Bool TextSetAt(pText* text, s32 inx, char ch);

text ... pointer to text object

inx .. index of the character (0..)

ch ... character to set

Set character at given position. Checks valid position. Returns False on memory error.

Bool TextSetFirst(pText* text, char ch);

text ... pointer to text object

ch ... character to set

Set first character of text string. Checks valid position. Returns False on memory error.

Bool TextSetLast(pText* text, char ch);

text ... pointer to text object

ch ... character to set

Set last character of text string. Checks valid position. Returns False on memory error.

Set text

Bool TextSetChar(pText* text, char ch);

text ... pointer to text object

ch ... character to set

Set text to single-char string. Returns False on memory error.

Bool TextSetStrLen(pText* text, const char* str, s32 len);

text ... pointer to text object

str ... ASCII text string

len ... length of ASCII string

Set text from ASCII text with length. Returns False on memory error.

Bool TextSetStr(pText* text, const char* str);

text ... pointer to text object

str ... ASCII text string

Set text from ASCII text. Returns False on memory error.

void TextSet(pText* dst, const pText* src);

dst ... pointer to destination text object

src ... pointer to source text object

Set text from another text object.

void TextExc(pText* text1, pText* text2);

text1 ... pointer to 1st text object

text2 ... pointer to 2nd text object

Exchange text objects.

Add text

Bool TextAddRep(pText* text, const pText* src, int rep);

text ... pointer to text object

src .. pointer to source text object

rep ... number of repeats

Add repeated text. Returns False on memory error.

Bool TextAddRepStrLen(pText* text, const char* str, int rep, s32 len);

text ... pointer to text object

src .. pointer to ASCII source string

rep ... number of repeats

len ... length of ASCII source string

Add repeated ASCII text string with length. Returns False on memory error.

Bool TextAddRepStr(pText* text, const char* str, int rep);

text ... pointer to text object

src .. pointer to ASCIIZ source string

rep ... number of repeats

Add repeated ASCIIZ text string. Returns False on memory error.

Bool TextAddRepCh(pText* text, char ch, int rep);

text ... pointer to text object

ch .. character to add to the text

rep ... number of repeats

Add repeated character. Returns False on memory error.

Bool TextAddRepSpc(pText* text, int rep);

text ... pointer to text object

rep ... number of repeats

Add repeated space character. Returns False on memory error.

Bool TextAdd(pText* text, const pText* src);

text ... pointer to text object

src .. pointer to source text object

Add text. Returns False on memory error.

Bool TextAddStrLen(pText* text, const char* str, s32 len);

text ... pointer to text object

src .. pointer to ASCII source string

len ... length of ASCII source string

Add ASCII text string with length. Returns False on memory error.

Bool TextAddStr(pText* text, const char* str);

text ... pointer to text object

src .. pointer to ASCIIZ source string

Add ASCIIZ text string. Returns False on memory error.

Bool TextAddCh(pText* text, char ch)

text ... pointer to text object

ch .. character to add to the text

Add character. Returns False on memory error.

Bool TextAddSpc(pText* text);

text ... pointer to text object

Add space character. Returns False on memory error.

Set decadic integer

Bool TextSetAddInt(pText* text, s32 num, char sep, Bool add, Bool unsign);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

add ... True = add text, False = set text

unsign ... number is unsigned

Set/add signed/unsigned integer number to text. Returns False on memory error.

Bool TextSetUInt(pText* text, u32 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Convert unsigned integer number to text. Returns False on memory error.

Bool TextAddUInt(pText* text, u32 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Add unsigned integer number to text. Returns False on memory error.

Bool TextSetInt(pText* text, s32 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Convert signed integer number to text. Returns False on memory error.

Bool TextAddInt(pText* text, s32 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Add signed integer number to text. Returns False on memory error.

Bool TextSetAddInt64(pText* text, s64 num, char sep, Bool add, Bool unsign);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

add ... True = add text, False = set text

unsign ... number is unsigned

Set/add signed/unsigned 64-bit integer number to text. Returns False on memory error.

Bool TextSetU64(pText* text, u64 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Convert unsigned integer number to text. Returns False on memory error.

Bool TextAddU64(pText* text, u64 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Add unsigned integer number to text. Returns False on memory error.

Bool TextSetS64(pText* text, s64 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Convert signed integer number to text. Returns False on memory error.

Bool TextAddS64(pText* text, s64 num, char sep);

text ... pointer to text object

num ... number

sep ... thousand separator (0 = not used)

Add signed integer number to text. Returns False on memory error.

s32 TextToInt(const pText* text);

text ... pointer to text object

Convert text to signed integer number. Text can be in HEX format “0xN” or “\$N”.

u32 TextToInt(const pText* text);

text ... pointer to text object

Convert text to unsigned integer number. Text can be in HEX format “0xN” or “\$N”.

s64 TextToS64(const pText* text);

text ... pointer to text object

Convert text to 64-bit signed integer number. Text can be in HEX format “0xN” or “\$N”.

u64 TextToU64(const pText* text);

text ... pointer to text object

Convert text to 64-bit unsigned integer number. Text can be in HEX format “0xN” or “\$N”.

Set digits

Bool TextSetAdd2Dig(pText* text, s8 num, Bool add);

text ... pointer to text object

num ... number

add ... True = add text, False = set text

Set/add 2 digits. Returns False on memory error.

Bool TextSet2Dig(pText* text, s8 num);

text ... pointer to text object

num ... number

Set 2 digits. Returns False on memory error.

Bool TextAdd2Dig(pText* text, s8 num);

text ... pointer to text object

num ... number

Add 2 digits. Returns False on memory error.

Bool TextSetAdd2DigSpc(pText* text, s8 num, Bool add);

text ... pointer to text object

num ... number

add ... True = add text, False = set text

Set/add 2 digits with space padding. Returns False on memory error.

Bool TextSet2DigSpc(pText* text, s8 num);

text ... pointer to text object

num ... number

Set 2 digits with space padding. Returns False on memory error.

Bool TextAdd2DigSpc(pText* text, s8 num);

text ... pointer to text object

num ... number

Add 2 digits with space padding. Returns False on memory error.

Bool TextSetAdd4Dig(pText* text, s16 num, Bool add);

text ... pointer to text object

num ... number

add ... True = add text, False = set text

Set/add 4 digits. Returns False on memory error.

Bool TextSet4Dig(pText* text, s16 num);

text ... pointer to text object

num ... number

Set 4 digits. Returns False on memory error.

Bool TextAdd4Dig(pText* text, s16 num);

text ... pointer to text object

num ... number

Add 4 digits. Returns False on memory error.

Set HEX integer

Bool TextSetAddHex(pText* text, u32 num, s8 digits, char sep, Bool add);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

add ... True = add text, False = set text

Set/add integer to text as HEX. Returns False on memory error.

Bool TextSetHex(pText* text, u32 num, s8 digits, char sep);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

Convert integer to text as HEX. Returns False on memory error.

Bool TextAddHex(pText* text, u32 num, s8 digits, char sep);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

Add integer to text as HEX. Returns False on memory error.

Bool TextSetAddHex64(pText* text, u64 num, s8 digits, char sep, Bool add);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

add ... True = add text, False = set text

Set/add 64-bit integer to text as HEX. Returns False on memory error.

Bool TextSetHex64(pText* text, u64 num, s8 digits, char sep);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

Convert 64-bit integer to text as HEX. Returns False on memory error.

Bool TextAddHex64(pText* text, u64 num, s8 digits, char sep);

text ... pointer to text object

num ... number

digits ... number of digits (0 = auto digits)

sep ... 4-digit separator (0 = none)

Add 64-bit integer to text as HEX. Returns False on memory error.

Set decimal number

Bool TextSetAddFloat(pText* text, float num, s8 digits, Bool add);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 6)

add ... True = add text, False = set text

Set/add float number to text. Returns False on error.

Bool TextSetFloat(pText* text, float num, s8 digits);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 6)

Convert float number to text. Returns False on error.

Bool TextAddFloat(pText* text, float num, s8 digits);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 6)

Add float number to text. Returns False on error.

float TextToFloat(const pText* text);

text ... pointer to text object

Convert text to float number.

Bool TextSetAddDouble(pText* text, double num, s8 digits, Bool add);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 14)

add ... True = add text, False = set text

Set/add double number to text. Returns False on error.

Bool TextSetDouble(pText* text, double num, s8 digits);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 14)

Convert double number to text. Returns False on error.

Bool TextAddDouble(pText* text, double num, s8 digits);

text ... pointer to text object

num ... number

digits ... number of digits (recommended digits = 14)

Add double number to text. Returns False on error.

double Text.ToDouble(const pText* text);

text ... pointer to text object

Convert text to double number.

Date and time

Bool TextSetAddTime(pText* text, u32 ut, s16 ms, s8 hour, s8 sec, s8 sep, Bool add);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

hour ... hour format 0=1:23, 1= 1:23, 2=01:23, 3=1.23p

sec ... second format 0=1:23, 1=1:23:45, 2=1:23:45.678

sep ... separator 0=none, 1=":" and ".", 2=":" and ",", 3=".:" and ","

add ... True = add text, False = set text

Set/add time in Unix format. Returns False on memory error.

Bool TextSetTime(pText* text, u32 ut, s16 ms, s8 hour, s8 sec, s8 sep);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

hour ... hour format 0=1:23, 1= 1:23, 2=01:23, 3=1.23p

sec ... second format 0=1:23, 1=1:23:45, 2=1:23:45.678

sep ... separator 0=none, 1=":" and ".", 2=":" and ",", 3=".:" and ","

Set time in Unix format. Returns False on memory error.

Bool TextAddTime(pText* text, u32 ut, s16 ms, s8 hour, s8 sec, s8 sep);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

hour ... hour format 0=1:23, 1= 1:23, 2=01:23, 3=1.23p

sec ... second format 0=1:23, 1=1:23:45, 2=1:23:45.678

sep ... separator 0=none, 1=":" and ".", 2=":" and ",", 3=".:" and ","

Add time in Unix format. Returns False on memory error.

Bool TextSetAddDow(pText* text, u32 ut, Bool add);

text ... pointer to text object

ut ... time in Unix format

add ... True = add text, False = set text

Set/add day of week 2-character text from Unix time ("Su", "Mo", "Tu", ...).

Bool TextSetDow(pText* text, u32 ut);

text ... pointer to text object

ut ... time in Unix format

Set day of week 2-character text from Unix time (“Su”, “Mo”, “Tu”, ...).

Bool TextAddDow(pText* text, u32 ut);

text ... pointer to text object

ut ... time in Unix format

Add day of week 2-character text from Unix time (“Su”, “Mo”, “Tu”, ...).

Bool TextSetAddDateCustom(pText* text, s16 year, s8 mon, s8 day, const char* form, Bool add);

text ... pointer to text object

year ... year

mon ... month

day ... day

form ... formatting string

add ... True = add text, False = set text

Set/add date in custom format. Returns False on memory error. Characters in formatting string: “d”=day 1 or 2 digits, “D”=day 2 digits, “m”=month 1 or 2 digits, “M”=month 2 digits, “N”=month name 3 letters (“Jan”, “Feb”,...), “y”=year 2 digits, “Y”=year 4 digits.

Bool TextSetDateCustom(pText* text, s16 year, s8 mon, s8 day, const char* form);

text ... pointer to text object

year ... year

mon ... month

day ... day

form ... formatting string

Set date in custom format. Returns False on memory error. Characters in formatting string: “d”=day 1 or 2 digits, “D”=day 2 digits, “m”=month 1 or 2 digits, “M”=month 2 digits, “N”=month name 3 letters (“Jan”, “Feb”,...), “y”=year 2 digits, “Y”=year 4 digits.

Bool TextAddDateCustom(pText* text, s16 year, s8 mon, s8 day, const char* form);

text ... pointer to text object

year ... year

mon ... month

day ... day

form ... formatting string

Add date in custom format. Returns False on memory error. Characters in formatting string: “d”=day 1 or 2 digits, “D”=day 2 digits, “m”=month 1 or 2 digits, “M”=month 2 digits, “N”=month name 3 letters (“Jan”, “Feb”,...), “y”=year 2 digits, “Y”=year 4 digits.

Bool TextSetDate(pText* text, s16 year, s8 mon, s8 day, u8 form);

text ... pointer to text object

year ... year

mon ... month

day ... day

form ... format 0..10

Set date in pre-select format. Returns False on memory error. Date format:

- USA formats

0 ... M/D/y (01/25/22)

1 ... m/d/Y (1/25/2022)

2 ... M/D/Y (01/25/2022)

3 ... N d,Y (Jan 25,2022)

- European formats

4 ... D.M.y (25.01.22)

5 ... d.m.Y (25.1.2022)

6 ... D.M.Y (25.01.2022)

7 ... d. N Y (25. Jan 2022)

- ISO format

8 ... y-M-D (22-01-25)

9 ... Y-M-D (2022-01-25)

- Technical format

10 ... yMD (220125)

Bool TextAddDate(pText* text, s16 year, s8 mon, s8 day, u8 form);

text ... pointer to text object

year ... year

mon ... month

day ... day

form ... format 0..10

Add date in pre-select format. Returns False on memory error.

Bool TextSetDateUnix(pText* text, u32 ut, u8 form);

text ... pointer to text object

ut ... time in Unix format

form ... format 0..10

Set Unix date in pre-select format. Returns False on memory error.

Bool TextAddDateUnix(pText* text, u32 ut, u8 form);

text ... pointer to text object

ut ... time in Unix format

form ... format 0..10

Add Unix date in pre-select format. Returns False on memory error.

Bool TextSetAddDateTech(pText* text, u32 ut, u16 ms, Bool add);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

add ... True = add text, False = set text

Set/add Unix date and time in technical format yyyyymmddhhmmssmmm (17 digits). Returns False on memory error.

Bool TextSetDateTech(pText* text, u32 ut, u16 ms);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

Set Unix date and time in technical format yyyyymmddhhmmssmmm (17 digits). Returns False on memory error.

Bool TextAddDateTech(pText* text, u32 ut, u16 ms);

text ... pointer to text object

ut ... time in Unix format

ms ... number of milliseconds

Add Unix date and time in technical format yyyyymmddhhmmssmmm (17 digits). Returns False on memory error.

Find and replace

int TextFindChar(const pText* text, char ch, int pos);

text ... pointer to text object

ch ... character to search

pos ... starting position

Find character from starting position. Returns character position or -1=not found.

int TextFindCharRev(const pText* text, char ch, int pos);

text ... pointer to text object

ch ... character to search

pos ... starting position

Find character reversed from starting position. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindSpace(const pText* text, int pos);

text ... pointer to text object

pos ... starting position

Find white space character from starting position (space/tab/newline 1..32). Returns position or -1=not found.

int TextFindSpaceRev(const pText* text, int pos);

text ... pointer to text object

pos ... starting position

Find white space character reverse from starting position (space/tab/newline 1..32). Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindNoSpace(const pText* text, int pos);

text ... pointer to text object

pos ... starting position

Find non-white space character from starting position (no space/tab/newline, > 32). Returns position or -1=not found.

int TextFindNoSpaceRev(const pText* text, int pos);

text ... pointer to text object

pos ... starting position

Find non-white space character reverse from starting position (no space/tab/newline, > 32). Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFind(const pText* text, const pText* fnd, int pos);

text ... pointer to text object

fnd ... pointer to text object to search

pos ... starting position

Find text from starting position. Returns position or -1=not found.

int TextFindStrLen(const pText* text, const char* fnd, s32 len, int pos);

text ... pointer to text object

fnd ... pointer to ASCII string to search

len ... length of ASCII string

pos ... starting position

Find ASCII string with length from starting position. Returns position or -1=not found.

int TextFindStr(const pText* text, const char* fnd, int pos);

text ... pointer to text object

fnd ... pointer to ASCII string to search

pos ... starting position

Find ASCII string from starting position. Returns position or -1=not found.

int TextFindRev(const pText* text, const pText* fnd, int pos);

text ... pointer to text object

fnd ... pointer to text object to search

pos ... starting position

Find text reverse from starting position. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindStrLenRev(const pText* text, const char* fnd, s32 len, int pos);

text ... pointer to text object

fnd ... pointer to ASCII string to search

len ... length of ASCII string

pos ... starting position

Find ASCII string with length reverse from starting position. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindStrRev(const pText* text, const char* fnd, int pos);

text ... pointer to text object

fnd ... pointer to ASCII string to search

pos ... starting position

Find ASCII string reverse from starting position. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindListStrLen(const pText* text, const char* list, int len, int pos);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to search

len ... number of characters in the ASCII list (= length of ASCII string)

pos ... starting position

Find characters from ASCII string list with length. Returns position or -1=not found.

int TextFindListStr(const pText* text, const char* list, int pos);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to search

pos ... starting position

Find characters from ASCII string list. Returns position or -1=not found.

int TextFindList(const pText* text, const pText* list, int pos);

text ... pointer to text object

list ... pointer to text object with list of characters to search

pos ... starting position

Find characters from text list. Returns position or -1=not found.

int TextFindListStrLenRev(const pText* text, const char* list, int len, int pos);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to search

len ... number of characters in the ASCII list (= length of ASCII string)

pos ... starting position

Find characters from ASCII string list with length. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindListStrRev(const pText* text, const char* list, int pos);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to search

pos ... starting position

Find characters from ASCIIZ string list. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindListRev(const pText* text, const pText* list, int pos);

text ... pointer to text object

list ... pointer to text object with list of characters to search

pos ... starting position

Find characters from text list. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindExceptListStrLen(const pText* text, const char* list, int len, int pos);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to NOT-search

len ... number of characters in the ASCII list (= length of ASCII string)

pos ... starting position

Find characters except ASCII string list with length. Returns position or -1=not found.

int TextFindExceptListStr(const pText* text, const char* list, int pos);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to NOT-search

pos ... starting position

Find characters except ASCIIZ string list. Returns position or -1=not found.

int TextFindExceptList(const pText* text, const pText* list, int pos);

text ... pointer to text object

list ... pointer to text object with list of characters to NOT-search

pos ... starting position

Find characters except string list. Returns position or -1=not found.

int TextFindExceptListStrLenRev(const pText* text, const char* list, int len, int pos);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to NOT-search

len ... number of characters in the ASCII list (= length of ASCII string)

pos ... starting position

Find characters reverse except ASCII string list with length. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindExceptListStrRev(const pText* text, const char* list, int pos);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to NOT-search

pos ... starting position

Find characters reverse except ASCIIZ string list. Returns position or -1=not found. Use pos=BIGINT to search from the end.

int TextFindExceptListRev(const pText* text, const pText* list, int pos);

text ... pointer to text object

list ... pointer to text object with list of characters to NOT-search

pos ... starting position

Find characters reverse except string list. Returns position or -1=not found. Use pos=BIGINT to search from the end.

Bool TextSubst(pText* text, const pText* find, const pText* subst);

text ... pointer to text object

find ... pointer to text object to search

subst ... pointer to text object with string that will replace the found text

Substitute text. Returns False on memory error.

Bool TextSubstListStrLen(pText* text, const char* list, const char* subst, int len);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to search

subst ... pointer to ASCII string with list of characters that will replace

len ... number of characters in the ASCII list (= length of ASCII string)

Substitute characters from ASCII string list with length. Strings ‘list’ and ‘subst’ must be of the same length - each found character from ‘list’ is matched by one replacement character from ‘subst’. Returns False on memory error.

Bool TextSubstListStr(pText* text, const char* list, const char* subst);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to search

subst ... pointer to ASCIIZ string with list of characters that will replace

Substitute characters from ASCIIZ string list. Strings ‘list’ and ‘subst’ must be of the same length - each found character from ‘list’ is matched by one replacement character from ‘subst’. Returns False on memory error.

Bool TextSubstList(pText* text, const pText* list, const pText* subst);

text ... pointer to text object

list ... pointer to text object with list of characters to search

subst ... pointer to text object with list of characters that will replace

Substitute characters from text list. Strings ‘list’ and ‘subst’ must be of the same length - each found character from ‘list’ is matched by one replacement character from ‘subst’. Returns False on memory error.

Conversion

Bool TextUpperCase(pText* text);

text ... pointer to text object

Convert text to uppercase. Returns False on memory error.

Bool TextLowerCase(pText* text);

text ... pointer to text object

Convert text to lowercase. Returns False on memory error.

Bool TextFlipCase(pText* text);

text ... pointer to text object

Convert text to opposite case. Returns False on memory error.

Bool TextWordCase(pText* text);

text ... pointer to text object

Convert text to wordcase - first letter upper. Returns False on memory error.

Bool TextReverse(pText* text);

text ... pointer to text object

Convert text to reverse order. Returns False on memory error.

Part of text

Bool TextLeft(pText* dst, const pText* src, int len);

dst ... pointer to destination text object

src ... pointer to source text object

len ... length of text

Get left part of the text. Returns False on memory error.

Bool TextRight(pText* dst, const pText* src, int len);

dst ... pointer to destination text object

src ... pointer to source text object

len ... length of text

Get right part of the text. Returns False on memory error.

Bool TextRightFrom(pText* dst, const pText* src, int pos);

dst ... pointer to destination text object

src ... pointer to source text object

pos ... start position of the right part

Get right part of the text from position. Returns False on memory error.

Bool TextMid(pText* dst, const pText* src, int pos, int len);

dst ... pointer to destination text object

src ... pointer to source text object

pos ... start position of the middle part

len ... length of text

Get middle part of the text. Returns False on memory error.

Insert/delete

Bool TextInsertStrLen(pText* dst, int pos, const char* src, int len);

dst ... pointer to destination text object

pos ... start position where to insert new text

src ... pointer to ASCII source text

len ... length of ASCII source text

Insert ASCII string with length. Returns False on memory error.

Bool TextInsertStr(pText* dst, int pos, const char* src);

dst ... pointer to destination text object

pos ... start position where to insert new text

src ... pointer to ASCIIZ source text

Insert ASCIIZ string. Returns False on memory error.

Bool TextInsert(pText* dst, int pos, const pText* src);

dst ... pointer to destination text object

pos ... start position where to insert new text

src ... pointer to source text object

Insert text. Returns False on memory error.

Bool TextInsertChar(pText* dst, int pos, char ch);

dst ... pointer to destination text object

pos ... start position where to insert the character

ch ... character to insert

Insert character. Returns False on memory error.

Bool TextDelete(pText* text, int pos, int num);

text ... pointer to text object

pos ... position of the part to delete

num ... number of characters to delete

Delete characters. Returns False on memory error.

Bool TextDelFirst(pText* text);

text ... pointer to text object

Delete first character of the text. Returns False on memory error.

Bool TextDelLast(pText* text);

text ... pointer to text object

Delete last character of the text. Returns False on memory error.

Bool TextDelToEnd(pText* text, int pos);

text ... pointer to text object

pos ... start position of right part to delete

Delete text to end. Returns False on memory error.

Bool TextDelListStrLen(pText* text, const char* list, int len);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to delete

len ... number of characters in the ASCII list (= length of ASCII string)

Delete characters from ASCII string list with length. Returns False on memory error.

Bool TextDelListStr(pText* text, const char* list);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to delete

Delete characters from ASCIIZ string list. Returns False on memory error.

Bool TextDelList(pText* text, const pText* list);

text ... pointer to text object

list ... pointer to text object with list of characters to delete

Delete characters from text list. Returns False on memory error.

Bool TextDelExceptListStrLen(pText* text, const char* list, int len);

text ... pointer to text object

list ... pointer to ASCII string with list of characters to NOT-delete

len ... number of characters in the ASCII list (= length of ASCII string)

Delete characters except ASCII string list with length. Returns False on memory error.

Bool TextDelExceptListStr(pText* text, const char* list);

text ... pointer to text object

list ... pointer to ASCIIZ string with list of characters to NOT-delete

Delete characters except ASCIIZ string list. Returns False on memory error.

Bool TextDelExceptList(pText* text, const pText* list);

text ... pointer to text object

list ... pointer to text object with list of characters to NOT-delete

Delete characters except text list. Returns False on memory error.

Bool TextDelWordStrLen(pText* text, const char* word, int len);

text ... pointer to text object

word ... pointer to ASCII string with word to delete

len ... length of the ASCII word

Find and delete all ASCII string words with length. Returns False on memory error.

Bool TextDelWordStr(pText* text, const char* word);

text ... pointer to text object

word ... pointer to ASCIIZ string with word to delete

Find and delete all ASCIIZ string words. Returns False on memory error.

Bool TextDelWord(pText* text, pText* word);

text ... pointer to text object

word ... pointer to text object with word to delete

Find and delete all text words. Returns False on memory error.

Bool TextTrimLeft(pText* text);

text ... pointer to text object

Trim white spaces from begin of the string. Returns False on memory error.

Bool TextTrimRight(pText* text);

text ... pointer to text object

Trim white spaces from end of the string. Returns False on memory error.

Bool TextTrim(pText* text);

text ... pointer to text object

Trim white spaces from begin and end of the string. Returns False on memory error.

Multi-line text

int TextRowsNum(const pText* text);

text ... pointer to text object

Get number of rows of multi-line text (search LF separator).

Bool TextRow(pText* dst, const pText* src, int row);

dst ... pointer to destination text object (to get row)

src ... pointer to source text object (with multi-line text)

row ... required row 0..

Get one row from multi-line text (search LF, trim CR). Returns False on memory error.

Bool TextUnformat(pText* text);

text ... pointer to text object

Unformat text - substitute white characters with one space character.

Bool TextReformat(pText* text, int width);

text ... pointer to text object

width ... new width of paragraph (in number of characters)

Reformat text to different width of paragraph (inserts LF).

Formatted print

u32 TextPrintArg(pText* text, const char* fmt, va_list args);

text ... pointer to text object, or NULL to get only text length without printing

fmt ... formatting string (cannot be identical to destination ‘text’ object)

args ... arguments

Formatted print string into text, with argument list. Returns number of characters.

u32 TextPrint(pText* text, const char* fmt, ...);

text ... pointer to text object, or NULL to get only text length without printing

fmt ... formatting string (cannot be identical to destination ‘text’ object)

... ... arguments

Formatted print string into text, with variadic arguments. Returns number of characters.

u32 TextAddPrintArg(pText* text, const char* fmt, va_list args);

text ... pointer to text object, or NULL to get only text length without printing

fmt ... formatting string (cannot be identical to destination ‘text’ object)

args ... arguments

Add formatted print string to text, with argument list. Returns number of added characters.

u32 TextAddPrint(pText* text, const char* fmt, ...);

text ... pointer to text object, or NULL to get only text length without printing

fmt ... formatting string (cannot be identical to destination ‘text’ object)

... ... arguments

Add formatted print string to text, with variadic arguments. Returns number of added characters.

3.31. TextList - List of Text Strings

Files: lib_textlist.h, lib_textlist.c
Config: USE_TEXTLIST (default 1)

TextList is a list of text strings with variable array size. Library requires memory allocator (MemAlloc(), MemResize(), MemFree()).

List of text strings

```
typedef struct {  
    int      num;    // number of entries in the list  
    int      max;    // array size (current max. number of entries)  
    pText*   data;   // data array  
} sTextList;
```

void TextListInit(sTextList* list);

list ... pointer to list of text strings

Initialize list of text strings (constructor).

void TextListDelAll(sTextList* list);

list ... pointer to list of text strings

Delete all entries in the list.

Bool TextListSetNum(sTextList* list, int num, Bool init);

list ... pointer to list of text strings

num ... new number of entries in the list

init ... initialize and terminate entries

Set number of entries. Returns False on memory error.

void TextListTerm(sTextList* list);

list ... pointer to list of text strings

Terminate list of text strings (destructor).

int TextListNum(const sTextList* list);

list ... pointer to list of text strings

Get number of entries in the list.

pText* TestListData(sTextList* list);

list ... pointer to list of text strings

Get array of entries.

Bool TextListIsValid(const sTextList* list, int inx);

list ... pointer to list of text strings

inx ... entry index 0..

Check if entry index is valid.

Bool TextListIsNotValid(const sTextList* list, int inx);

list ... pointer to list of text strings

inx ... entry index 0..

Check if entry index is not valid.

pText* TextListEntry(sTextList* list, int inx);

list ... pointer to list of text strings

inx ... entry index 0..

Get pointer to text entry (without index checking).

int TextListNew(sTextList* list);

list ... pointer to list of text strings

Add new empty text entry to end of the list. Returns index of new entry, or -1 on memory error.

int TextListAddStrLen(sTextList* list, const char* text, int len);

list ... pointer to list of text strings

text ... pointer to ASCII string with text entry

len ... length of ASCII string

Add new ASCII string entry with length to end of the list. Returns index of new entry, or -1 on memory error.

int TextListAddStr(sTextList* list, const char* text);

list ... pointer to list of text strings

text ... pointer to ASCIIZ string with text entry

Add new ASCIIZ string entry to end of list. Returns index of new entry, or -1 on memory error.

int TextListAdd(sTextList* list, const pText* text);

list ... pointer to list of text strings

text ... pointer to text object with text entry

Add new text entry. Returns index of new entry, or -1 on memory error.

int TextListAddList(sTextList* list, const sTextList* list2);

list ... pointer to destination list of text strings

list2 ... pointer to source list of text strings to add

Add another list to the string list. Returns index of first entry, or -1 on error.

Bool TextListIns(sTextList* list, int inx, const pText* text);

list ... pointer to list of text strings

inx ... index of position (0..) where to insert the new entry

text ... pointer to text object with text entry

Insert new entry into list. Returns False on memory error.

Bool TextListInsList(sTextList* list, int inx, const sTextList* list2);

list ... pointer to destination list of text strings

inx ... index of position (0..) where to insert source list

list2 ... pointer to source list of text strings to insert

Insert another list into list. Returns False on memory error. Source list and destination list cannot be the same.

Bool TextListCopy(sTextList* list, const sTextList* list2);

list ... pointer to destination list of text strings

list2 ... pointer to source list of text strings

Copy list from another list (destroys old content). Returns False on memory error.

void TextListDel(sTextList* list, int inx, int num);

list ... pointer to list of text strings

inx ... index of position (0..) where to start delete the entries

num ... number of entries to delete

Delete entries from text list.

void TextListDelFirst(sTextList* list);

list ... pointer to list of text strings

Delete first entry of the text list.

void TextListDelLast(sTextList* list);

list ... pointer to list of text strings

Delete last entry of the text list.

void TextListDelToEnd(sTextList* list, int inx);

list ... pointer to list of text strings

inx ... index of position (0..) where to start delete the entries

Delete entries from text list to end of list.

int TextListDup(sTextList* list, int inx);

list ... pointer to list of text strings

inx ... index of entry to duplicate (0..)

Duplicate text entry (add new entry to end of list).. Returns index of new entry, or -1 on error.

void TextListGet(const sTextList* list, pText* text, int inx);

list ... pointer to list of text strings

text ... pointer to destination text object

inx ... index of the entry

Get text from text list.

void TextListSet(sTextList* list, const pText* text, int inx);

list ... pointer to list of text strings

text ... pointer to source text object

inx ... index of the entry

Set text entry in the text list.

void TextListReverse(sTextList* list);

list ... pointer to list of text strings

Reverse order of entries of text list.

Bool TextListSplitStrLen(sTextList* list, const pText* text, const char* delim, int len, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

delim ... ASCII text of delimiter

len ... length of ASCII text of delimiter (or -1 to auto-length,
or 0 to split source text to single characters)

limit ... max. number of entries (use BIGINT to unlimited)

Split text string into text list by ASCII string delimiter with length (destroys old content od text list). Returns False on memory error.

Bool TextListSplitStr(sTextList* list, const pText* text, const char* delim, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

delim ... ASCII text of delimiter

(use empty text to split source text to single characters)

limit ... max. number of entries (use BIGINT to unlimited)

Split text string into text list by ASCIIZ string delimiter (destroys old content od text list). Returns False on memory error.

Bool TextListSplit(sTextList* list, const pText* text, const pText* delim, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

delim ... text of delimiter (empty text = split source text to single characters)

limit ... max. number of entries (use BIGINT to unlimited)

Split text string into text list by text delimiter (destroys old content od text list). Returns False on memory error.

Bool TextListSplitChar(sTextList* list, const pText* text, char delim, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

delim ... 1-character delimiter

limit ... max. number of entries (use BIGINT to unlimited)

Split text string into text list by 1-character delimiter (destroys old content od text list). Returns False on memory error.

Bool TextListSplitWords(sTextList* list, const pText* text, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

limit ... max. number of entries (use BIGINT to unlimited)

Split text to words, limited by white space characters. Returns False on memory error.

Bool TextListSplitLines(sTextList* list, const pText* text, int limit);

list ... pointer to destination list (content will be destroyed)

text ... source text to split

limit ... max. number of entries (use BIGINT to unlimited)

Split text to lines (delimiter is LF, reduce CR).

Bool TextListJoinStrLen(const sTextList* list, pText* text, const char* delim, int len);

list ... pointer to source list

text ... pointer to destination text object to join

delim ... ASCII text of delimiter

len ... length of ASCII text of delimiter

Join text strings from the list with ASCII string delimiter with length. Returns False on memory error.

Bool TextListJoinStr(const sTextList* list, pText* text, const char* delim);

list ... pointer to source list

text ... pointer to destination text object to join

delim ... ASCIIZ text of delimiter

Join text strings from the list with ASCIIZ string delimiter. Returns False on memory error.

Bool TextListJoin(const sTextList* list, pText* text, pText* delim);

list ... pointer to source list

text ... pointer to destination text object to join

delim ... pointer to text object with delimiter

Join text strings from the list with text delimiter. Returns False on memory error.

Bool TextListJoinChar(const sTextList* list, pText* text, char delim);

list ... pointer to source list

text ... pointer to destination text object to join

delim ... 1-character delimiter

Join text strings from the list with character delimiter. Returns False on memory error.

Bool TextListJoinLines(const sTextList* list, pText* text);

list ... pointer to source list

text ... pointer to destination text object to join

Join lines with LF as delimiter. Returns False on memory error.

3.32. TPrint - Print to Attribute Text Buffer

Files: lib_tprint.h, lib_tprint.c

Config: USE_TPRINT (default 1)

The TPrint library is intended for printing texts into the PicoVGA text buffer with the GF_ATEXT or GC_MTEXT format. It can be used to work with the text mode also for graphic display types, as the contents of the text buffer can be transferred to the graphic display by the DrawTextBuf() function.

Text PC-colors (if using PC CGA colors)

PC_BLACK	0
PC_BLUE	1
PC_GREEN	2
PC_CYAN	3
PC_RED	4
PC_MAGENTA	5
PC_BROWN	6
PC_LTGRAY	7
PC_GRAY	8
PC_LTBLUE	9
PC_LTGREEN	10
PC_LTCYAN	11
PC_LTRED	12
PC廖MAGENTA	13
PC_YELLOW	14
PC_WHITE	15

```
u8* TPrintBuf;           // current print buffer
int TPrintBufW;          // width of print buffer (in number of characters)
int TPrintBufH;          // height of print buffer (in number of rows)
int TPrintBufWB;         // width of print buffer in bytes
int TPrintX;              // current print position X
int TPrintY;              // current print position Y
u8 TPrintCol;            // current print color
```

PC_COLOR(bg,fg)

Macro to compose PC color from bg=background and fc=foreground color.

void TPrintSetup(u8* buf, int bufw, int bufh, int bufwb);

buf ... pointer to print buffer

bufw ... width of print buffer (in number of characters)

bufh ... height of print buffer (in number of rows)

bufwb ... width of print buffer in bytes

Setup print service (if bufwb < 2*bufw, use mono text).

u8* TPrintAddr(int x, int y);

x ... X coordinate (character position on row)

y ... Y coordinate (row)

Get address in the buffer.

void TPrintClear();

Clear text screen, using current color.

void TPrintHome();

Print home.

void TPrintSetPos(int x, int y);

x ... X coordinate (character position on row)

y ... Y coordinate (row)

Set print position.

void TPrintAddPos(int x, int y);

x ... shift X coordinate (character position on row)

y ... shift Y coordinate (row)

Shift relative print position.

void TPrintSetCol(u8 col);

col ... new print color

Set print color (2x4 bits of colors, use PC_COLOR macro).

void TPrintChar0(char ch);

ch ... character to print

Print character, not using control characters.

void TPrintShadow();

Print shadow (darkening the colours of the current character).

void TPrintChar(char ch);

ch ... character to print

Print character, using control characters CR, LF, TAB.

void TPrintSpc();

Print space character.

void TPrintSpcTo(int pos);

pos ... position on row

Print space to position.

void TPrintCharRep(char ch, int num);

ch ... character to print

num ... number of characters

Print repeated character.

void TPrintShadowRep(int num);

num ... number of shadow characters

Print repeated shadow (darkening colours).

void TPrintSpcRep(int num);

num ... number of spaces

Print repeated space.

void TPrintText(const char* text, int len);

text ... text to print

len ... length of text, or -1 = auto

Print text.

void TPrintHLine(int x, int y, int w);

x ... start X coordinate

y ... start Y coordinate

w ... line width

Print horizontal line into screen, using current color (must not stretch outside valid range).

void TPrintDHLine(int x, int y, int w);

x ... start X coordinate

y ... start Y coordinate

w ... line width

Print horizontal double line into screen, using current color (must not stretch out of valid range).

void TPrintVLine(int x, int y, int h);

x ... start X coordinate

y ... start Y coordinate

h ... line height

Print vertical line into screen, using current color (must not stretch outside valid range).

void TPrintDLine(int x, int y, int h);

x ... start X coordinate

y ... start Y coordinate

h ... line height

Print vertical double line into screen, using current color (must not stretch out of valid range).

void TPrintFrame(int x, int y, int w, int h);

x ... start X coordinate

y ... start Y coordinate

w ... frame width

h ... frame height

Print frame, using current color.

void TPrintDFrame(int x, int y, int w, int h);

x ... start X coordinate

y ... start Y coordinate

w ... frame width

h ... frame height

Print double frame, using current color.

void TPrintFill(int x, int y, int w, int h);

x ... start X coordinate

y ... start Y coordinate

w ... box width

h ... box height

Print filled box, using current color.

void TPrintScroll();

Scroll print buffer.

3.33. Tree List

Files: lib_tree.h, lib_tree.c
Config: USE_TREE (default 1)

The Tree library is used to organize data items into a tree structure. The sTree structure is placed inside a data item so that the items can be chained into a tree.

Tree entry

```
typedef struct {  
    struct sTree_*  parent; // pointer to parent (NULL=none, =root)  
    struct sTree_*  next;  // pointer to next neighbor (NULL=none)  
    struct sTree_*  prev;  // pointer to previous neighbor (NULL=none)  
    struct sTree_*  first; // pointer to first child (NULL=none)  
    struct sTree_*  last;  // pointer to last child (NULL=none)  
} sTree;
```

void TreInit(sTree* tree);

tree ... pointer to tree entry

Initialize tree entry (set all pointers to NULL).

void TreeAddFirst(sTree* tree, sTree* entry);

tree ... pointer to parent tree entry

entry ... pointer to child tree entry

Add initialized new child entry (or branch) into begin of the list. New child can have branch of children and it must have initialized entries first and last.

void TreeAddLast(sTree* tree, sTree* entry);

tree ... pointer to parent tree entry

entry ... pointer to child tree entry

Add initialized new child entry (or branch) into end of list. New child can have branch of children and it must have initialized entries first and last.

void TreeAddAfter(sTree* tree, sTree* entry);

tree ... pointer to current tree entry

entry ... pointer to neighbor tree entry

Add initialized neighbor entry (or branch) after current entry. Neighbor can have branch of children and it must have initialized entries first and last.

void TreeAddBefore(sTree* tree, sTree* entry);

tree ... pointer to current tree entry

entry ... pointer to neighbor tree entry

Add initialized neighbor entry (or branch) before current entry. Neighbor can have branch of children and it must have initialized entries first and last.

void TreeRemove(sTree* tree);

tree ... pointer to tree entry

Detach entry from parent and neighbors. Children will stay attached to entry, parent becomes NULL.

sTree* TreeFindNext(sTree* tree);

tree ... pointer to current tree entry

Find next tree entry (cyclic, including children).

sTree* TreeFindNextStop(sTree* tree, sTree* stop);

tree ... pointer to current tree entry

stop ... pointer to stop tree entry

Find next tree entry with stop mark (including children). Returns NULL instead of stop entry.

TREEFOREACHCHILD(entry, parent)

entry ... variable of sTree* type

parent ... parent of sTree* type

Macro header of loop to walking through children of tree branch in forward direction.

TREEFOREACHCHILDBACK(entry, parent)

entry ... variable of sTree* type

parent ... parent of sTree* type

Macro header of loop to walking through children of tree branch in backward direction.

TREEFOREACH(entry, root)

entry ... variable of sTree* type

root ... starting entry of sTree* type (it will stop on this entry)

Macro header of loop to walking through all tree entries.

BASEFROMTREE(entry, type, member)

entry ... pointer to tree entry of sTree type

type ... type of owner of sTree entry

member ... name of member variable of sTree entry in its owner

Get pointer to base object from pointer to tree entry.

3.34. Video Player

Files: lib_video.h, lib_video.c

Config: USE_VIDEO (default 0)

The video player is used to play videos from the SD card. The video is played full screen, with a resolution of 320 x 240 pixels. The video files are prepared by PicoPadVideo (from the _tools folder). See PicoPadVideo for instructions on how to create a video file.

A video file (typical file extension is *.VID) is a sequence of frames containing an image and sound. The video file is designed to be played at 10 frames per second, on a 320 x 240 pixel resolution, 16-bit RGB565 color display. The audio is designed to be played back in 8-bit, 22050 samples per second, mono.

Each frame of the video has the following structure (frame size is 41117 bytes):

- 256 words u16 containing the color palette (512 bytes)
- image in 8-bit palette format, with resolution of 160 x 240 pixels (38400 bytes)
- 8-bit 22050 Hz mono sound, 2205 samples (2205 bytes)

The original 320x240 pixel image is converted to 160 pixels wide when the video is created and is converted to 8-bit palette. During playback, each pixel is duplicated to final resolution of 320 x 240 pixels.

In order to be used, the video player must be enabled by setting the USE_VIDEO configuration parameter to 1. At the same time, the USE_FRAMEBUF parameter is automatically set to 0, which disables the default frame buffer FrameBuf declared in the display driver. The video player defines its own FrameBuf buffer because it may be larger than the default one. In addition to the image data, frame loading from the SD card to the buffer takes place.

Video descriptor

```
typedef struct {
    u32    frametime;      // time of last frame
    u32    frames;         // total number of frames
    u32    frame;          // current frame
    sFile  file;           // open file descriptor
    u8     bufinx;         // current buffer index 0 or 1
    s8     volume;          // sound volume 0..15
    Bool   mute;            // sound mute
    Bool   pause;           // paused
    Bool   ctrl;             // display control
} sVideo;
```

Bool VideoOpen(sVideo* video, const char* filename);

video ... pointer to video descriptor (not initialized)

filename ... filename of the video file *.VID

Opens video. Returns False on error (file not found). Must be paired with VideoClose(), if successful. Disk must be mounted with DiskMount().

void VideoClose(sVideo* video);

video ... pointer to video descriptor (initialized)

Close video. Must be paired with successful VideoOpen().

void VideoDispCtrl(sVideo* video);

video ... pointer to video descriptor (initialized)

Repaint video control.

Bool VideoPlayFrame(sVideo* video);

video ... pointer to video descriptor (initialized)

Play next video frame. Returns False on error or end of file.

int VideoLen(sVideo* video);

video ... pointer to video descriptor (initialized)

Get video length in seconds.

int VideoPos(sVideo* video);

video ... pointer to video descriptor (initialized)

Get current video position in seconds.

void VideoShiftPos(sVideo* video, int shift);

video ... pointer to video descriptor (initialized)

shift ... relative shift current position in seconds

Shift relative video position in seconds.

void VideoSetVol(sVideo* video, s8 vol);

video ... pointer to video descriptor (initialized)

vol ... required sound volume 0..20

Video set volume 0..15.

void VideoSetPause(sVideo* video, Bool pause);

video ... pointer to video descriptor (initialized)

pause ... pause video

Video set pause.

void VideoSetMute(sVideo* video, Bool mute);

video ... pointer to video descriptor (initialized)

mute ... mute sound

Video set mute.

void VideoSetCtrl(sVideo* video, Bool ctrl);

video ... pointer to video descriptor (initialized)

ctrl ... set control bar

Set video control bar.

4. USB Library

The USB library mediates the USB connection via the internal USB controller of the RP2040 processor. Only 1 port is supported, either in device or host mode. The current version of the library supports communication protocols CDC (Communication Device Class - serial port) in host and device mode, and communication protocol HID (Human Interface Device Class) in host and device mode - as a keyboard and mouse. Expansions to other devices are planned for future versions of the PicoLibSDK library, including support for HUB (not currently implemented in the library).

The USB library was inspired by the original TinyUSB library with one notable difference - inside the USB driver, interrupt messages are not passed by events, but interrupt events are processed immediately. As a result, the driver has a very fast response to events on the USB bus and does not require running the Task handler. Any timing events are handled in response to SOF packets or in response to the system timer. It should be remembered that all internal driver servicing is done within the IRQ interrupt. Therefore, communication between the driver and the application must be done using lockable messages such as a ring buffer.

A special addition to the USB interface is the USB Mini-Port library, which is used for fast data transfer between two devices. It is particularly suitable as a multiplayer gaming connection.

The USB SDK source files are located in the `_sdk` folder.

4.1. USB Mini-Port

Files: `sdk_usbport.h`, `sdk_usbport.c`

Config: `USE_USBPORT (default 0)`

The USB Mini-Port is a special non-standard modification of the USB interface, designed to quickly connect two devices via a USB cable at Full-Speed (12 Mbps). Unlike the standard USB interface, no device enumeration or transfer of descriptors is used. Data is transferred as 64-byte packets over 15 communication channels in simplex mode (semi-duplex mode can also be used with limitations). Each channel is capable of transferring data at 64 KB per second; with 15 channels (indexed 1..15), the total data transfer speed is 960 KB per second, with checksums provided.

When connecting two USB devices, one device must be initialized as Master and the other as Slave. From the program's point of view, the further communication takes place in the same way, regardless of the mode in which the device is initialized.

Selected USB port

<code>USBPORT_OFF</code>	0	USB port is not initialized
<code>USBPORT_MASTER</code>	1	master mode selected (host mode)
<code>USBPORT_SLAVE</code>	2	slave mode selected (device mode)

Error codes

Error code is greater than 64 to ensure distinction from packet length.

USBPORT_ERR_OK	0	all OK
USBPORT_ERR_INIT	65	driver is not initialized
USBPORT_ERR_CON	66	opposite device is not connected
USBPORT_ERR_BUSY	67	channel is busy (transfer is in progress)
USBPORT_ERR_COMP	68	transfer is complete (or it was never launched)
USBPORT_ERR_LEN	69	incorrect length of formatted packet
USBPORT_ERR_CRC	70	CRC error of received packet

Initialize

u8 UsbPortGet();

Get current selected USB port **USBPORT_*** (0=not initialized, 1=master, 2=slave).

Bool UsbPortIsInit();

Check if USB port is initialized.

Bool UsbPortIsMaster();

Check if master mode is selected (host mode of USB).

Bool UsbPortIsSlave();

Check if slave mode is selected (device mode of USB).

u8* UsbPortBuf(u8 chan);

chan ... channel 1..15

Get DPRAM buffer address of the channel (64 bytes long + 64 bytes reserve after it).

void UsbPortInit(u8 port);

port ... required port **USBPORT_MASTER** or **USBPORT_SLAVE**

Initialize USB port.

void UsbPortTerm();

Terminate USB port.

Bool UsbPortConnected();

Check if opposite device is connected.

Multi-channel transfer

After receiving packet, next packet will start receiving automatically (simplex mode). To swap receiving and transmitting direction, reset channel on both sides using `UsbPortReset()`.

u8 UsbPortState(u8 chan);

chan ... channel 1..15

Get channel state. Returns error code:

USBPORT_ERR_OK channel is waiting and not active

USBPORT_ERR_INIT driver is not initialized

USBPORT_ERR_CON opposite device is not connected

USBPORT_ERR_BUSY channel is busy (transfer is in progress)

USBPORT_ERR_COMP transfer is complete (or it was never launched)

void UsbPortReset(u8 chan);

chan ... channel 1..15

Reset channel. Stop current transmission. Must be executed on both sides to ensure DATA0/DATA1 synchronization.

void UsbPortClear(u8 chan);

chan ... channel 1..15

Clear complete status of the channel.

Bool UsbPortSendChanReady(u8 chan);

chan ... channel 1..15

Check if transmit channel is ready to send next packet.

u8 UsbPortSendChan(u8 chan, const void* buf, int len);

chan ... channel 1..15

buf ... buffer with packet data to send

len ... length of data to send (0 to 64 bytes)

Send packet. Returns error code:

USBPORT_ERR_OK packet was sent OK

USBPORT_ERR_INIT driver is not initialized

USBPORT_ERR_CON opposite device is not connected

USBPORT_ERR_BUSY channel is busy (transfer is in progress)

Bool UsbPortRecvChanReady(u8 chan);

chan ... channel 1..15

Check if next received packet is ready.

```
u8 UsbPortRecvChan(u8 chan, void* buf, int len);
```

chan ... channel 1..15

buf ... buffer to load received data

len ... max. size of data to receive

Receive packet. If the received packet is longer than the buffer size, the packet data will be truncated. After a packet is received, a request to receive the next packet is automatically initiated. If it returns number between 0 and 64, it is length of received packet. If it returns number greater than 64, it is following error code:

USBPORT_ERR_INIT driver is not initialized

USBPORT_ERR_CON opposite device is not connected

USBPORT_ERR_BUSY channel is busy (transfer is in progress)

Transfer of formatted packet

After receiving packet, next packet will start receiving automatically (simplex mode). To swap receiving and transmitting direction, reset channel on both sides using `UsbPortReset()`.

Recommended to support at least hello packet, which will be sent after connection by host to device on channel 1, and by device to host on channel 2.

Packets are protected by the CRC16A checksum. The packet data is checksummed starting at offset 2 and the resulting checksum is stored at offset 0 as a 16-bit word.

Common packet (4 to 64 bytes)

Pay attention to the alignment of entries in the packet and the alignment of the entire packet structure in memory to ensure compatibility and portability. If you cannot keep the alignment, use the `PACKED` attribute of the packet.

```
typedef struct {  
    u16    crc;        // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)  
    u8     len;        // 2: (1) total packet length (4 to 64 bytes)  
    u8     cmd;        // 3: (1) command (USBPORT_CMD_*)  
    u8    data[60];    // 4: packet data (start of data array is u32 aligned)  
} sUsbPortPkt;
```

Ping packet (4 bytes) ... can be used instead of SOF frame USB packets

```
USBPORT_CMD_PING    0      // command: ping packet
```

```
typedef struct {  
    u16    crc;        // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)  
    u8     len;        // 2: (1) total packet length (= 4)  
    u8     cmd;        // 3: (1) command (= USBPORT_CMD_PING)  
} sUsbPortPktPing;
```

Hello packet (14 to 64 bytes) ... first packet that should be sent after connection

```
USBPORT_CMD_HELLO 1 // command: hello packet

typedef struct {

    u16    crc;          // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)
    u8     len;          // 2: (1) total packet length (14 to 64 bytes)
    u8     cmd;          // 3: (1) command (= USBPORT_CMD_HELLO)
    u32    uid;          // 4: (4) program unique identification number
    u16    ver;          // 8: (2) protocol version in hundredths (100 = v1.00)
    u16    var;          // 10: (2) program variant
    u8     infolen;      // 12: (1) length of info string 0..50 (without trailing zero)
    char   info[51];     // 13: (1..51) info string (typically ASCIIZ program name)

} sUsbPortPktHello;
```

Requirements (usually 8 bytes) ... mask of required packets to get from opposite side

```
USBPORT_CMD_REQ 2 // command: requirements

typedef struct {

    u16    crc;          // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)
    u8     len;          // 2: (1) total packet length (usually 8)
    u8     cmd;          // 3: (1) command (= USBPORT_CMD_REQ)
    u32    req;          // 4: (usually 4) bit mask of required packets
                        // from opposite side (B1 = hello packet required)

} sUsbPortPktReq;
```

Set random seed (usually 12 bytes) ... usually sent by master on start to setup game

```
USBPORT_CMD_SEED 3 // command: set seed of random generator

typedef struct {

    u16    crc;          // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)
    u8     len;          // 2: (1) total packet length (usually 12)
    u8     cmd;          // 3: (1) command (= USBPORT_CMD_SEED)
    u32    seedL;        // 4: (4) seed low
    u32    seedH;        // 8: (4) seed high

} sUsbPortPktSeed;
```

Quit the game (4 bytes) ... cancel connection

```
USBPORT_CMD_QUIT 4 // command: quit the game

typedef struct {

    u16    crc;          // 0: (2) checksum Crc16AFast (CRC-16 CCITT normal)

} sUsbPortPktQuit;
```

```

    u8      len;          // 2: (1) total packet length (= 4)
    u8      cmd;          // 3: (1) command (= USBPORT_CMD_QUIT)
} sUsbPortPktQuit;

```

Test pattern (8 to 64 bytes) ... used to check connection quality

```

USBPORT_CMD_TEST      5      // command: test pattern
typedef struct {
    u16      crc;          // 0: checksum Crc16AFast (CRC-16 CCITT normal)
    u8       len;          // 2: total packet length (8 to 64 bytes)
    u8       cmd;          // 3: command (USBPORT_CMD_TEST)
    u32      cnt;          // 4: sequence counter (to check dropped packets)
    u8     data[56];        // 8: test pattern (random data)
} sUsbPortPktTest;

```

First application packet (packets 0..9 are reserved for system common packets)

```
USBPORT_CMD_APP      10     // first application packet
```

Bool UsbPortSendPktReady(u8 chan);

Check if transmit channel is ready to send next packet.

u8 UsbPortSendPkt(u8 chan, const void* pkt);

chan ... channel 1..15

pkt ... buffer with sUsbPortPkt packet to send ('crc' will be
calculated in temporary buffer later, 'len' must be filled)

Send packet. Returns error code:

USBPORT_ERR_OK	packet was sent OK
USBPORT_ERR_INIT	driver is not initialized
USBPORT_ERR_CON	opposite device is not connected
USBPORT_ERR_BUSY	channel is busy (transfer is in progress)
USBPORT_ERR_LEN	incorrect length of formatted packet

Bool UsbPortRecvPktReady(u8 chan);

chan ... channel 1..15

Check if next received packet is ready.

u8 UsbPortRecvPkt(u8 chan, void* pkt);

chan ... channel 1..15

pkt ... buffer of size 64 bytes to receive sUsbPortPkt packet

Receive packet. Packet will appear in destination buffer even if length is wrong or if CRC is wrong. If real length of received packet is wrong, it will be stored in length entry of packet. Returns error code:

USBPORT_ERR_OK	packet was received OK
USBPORT_ERR_INIT	driver is not initialized
USBPORT_ERR_CON	opposite device is not connected
USBPORT_ERR_BUSY	channel is busy (transfer is in progress)
USBPORT_ERR_LEN	incorrect length of formatted packet
USBPORT_ERR_CRC	CRC error of received packet

Single-channel simple transfer

Channels:

USBPORT_OUT	1 // direction OUT from host to device
USBPORT_IN	2 // direction IN from device to host

Bool UsbPortSendReady();

Check if transmit channel is ready to send next packet.

Bool UsbPortSend(const void* pkt);

pkt ... buffer with sUsbPortPkt packet to send
(‘crc’ will be calculated auto later, ‘len’ must be filled)

Send packet. Returns False on transfer error.

Bool UsbPortRecvReady();

Check if next received packet is ready.

Bool UsbPortRecv(void* pkt);

pkt ... buffer of size 64 bytes to receive sUsbPortPkt packet

Receive packet. Returns False if packet not received.

4.2. USB CDC Device

Files: `sdk_usb_dev_cdc.h`, `sdk_usb_dev_cdc.c`

Config: `USE_USB_DEV_CDC` (default 0)

The USB CDC Device is used to connect to a host as a device with a serial communication channel. Configuration is done by setting the configuration parameter `USE_USB_DEV_CDC` to a value of 1 to 4, which represents the number of communication channels. It is possible to connect to the host using up to 4 independent communication channels, each with a baud rate in each direction of 64 KB per second (524 Kbaud). The interface is initialized with the `UsbDevInit(&UsbDevCdcSetupDesc)` command.

Instead of the following set of functions, you can also use the set of functions common to device and host mode, so that you do not have to distinguish whether the device is operating in host or device mode. For a description of the common function set, see the following chapter, USB CDC Host.

`void UsbDevInit(&UsbDevCdcSetupDesc);`

Initialize USB CDC device. Number of serial ports (number of interfaces) is selected with configuration parameter `USE_USB_DEV_CDC` = 1 to 4.

`void UsbTerm();`

Terminate USB interface.

`Bool UsbDevCdclnxIsMounted(u8 cdc_inx);`

`cdc_inx` ... CDC interface 0..3

Check if device interface with specified index is mounted.

`Bool UsbDevCdclsMounted();`

Check if device interface 0 is mounted.

`int UsbDevCdclnxReadReady(u8 cdc_inx);`

`cdc_inx` ... CDC interface 0..3

Get bytes available for reading from CDC interface with specified index.

`int UsbDevCdcReadReady();`

Get bytes available for reading from CDC interface 0.

`char UsbDevCdclnxReadChar(u8 cdc_inx);`

`cdc_inx` ... CDC interface 0..3

Read one character from CDC interface with specified index. Returns 0 if not ready.

char UsbDevCdcReadChar();

Read one character from CDC interface 0. Returns 0 if not ready.

int UsbDevCdclnxReadData(u8 cdc_inx, void* buf, int bufsize);

cdc_inx ... CDC interface 0..3

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface with specified index. Returns number of bytes.

int UsbDevCdcReadData(void* buf, int bufsize);

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface 0. Returns number of bytes.

void UsbDevCdclnxReadFlush(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Flush received data in CDC interface with specified index.

void UsbDevCdcReadFlush();

Flush received data in CDC interface 0.

int UsbDevCdclnxWriteReady(u8 cdc_inx)

cdc_inx ... CDC interface 0..3

Get bytes available for writing to CDC interface with specified index.

int UsbDevCdcWriteReady();

Get bytes available for writing to CDC interface 0.

Bool UsbDevCdclnxWriteChar(u8 cdc_inx, char ch);

cdc_inx ... CDC interface 0..3

ch ... character to write

Write one character to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbDevCdcWriteChar(char ch);

ch ... character to write

Write one character to CDC interface 0, wait until ready. Returns False if device is not connected.

Bool UsbDevCdclnxWriteData(u8 cdc_inx, const void* buf, int len);

cdc_inx ... CDC interface 0..3

buf ... buffer with data to send

len ... length of data

Write data to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbDevCdcWriteData(const void* buf, int len);

buf ... buffer with data to send

len ... length of data

Write data to CDC interface 0, wait until ready. Returns False if device is not connected.

4.3. USB CDC Host

Files: `sdk_usb_host_cdc.h`, `sdk_usb_host_cdc.c`

Config: `USE_USB_HOST_CDC` (default 0)

The USB CDC Host is designed to connect another device with a serial communication channel. Configuration is done by setting the configuration parameter `USE_USB_HOST_CDC` to a value of 1 to 4, which represents the number of communication channels. It is possible to connect device using up to 4 independent communication channels, each with a baud rate in each direction of 64 KB per second (524 KBaud). The interface is initialized with the **UsbHostInit()** command.

void UsbHostInit();

USB init in host mode. Number of serial ports (number of interfaces) is selected with configuration parameter `USE_USB_HOST_CDC` = 1 to 4.

void UsbTerm();

Terminate USB interface.

Bool UsbHostCdcInxIsMounted(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Check if device interface with specified index is mounted.

Bool UsbHostCdclIsMounted();

Check if device interface 0 is mounted.

int UsbHostCdcInxReadReady(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Get bytes available for reading from CDC interface with specified index.

int UsbHostCdcReadReady();

Get bytes available for reading from CDC interface 0.

char UsbHostCdcInxReadChar(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Read one character from CDC interface with specified index. Returns 0 if not ready.

char UsbHostCdcReadChar();

Read one character from CDC interface 0. Returns 0 if not ready.

int UsbHostCdcInxReadData(u8 cdc_inx, void* buf, int bufsize);

cdc_inx ... CDC interface 0..3

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface with specified index. Returns number of bytes.

int UsbHostCdcReadData(void* buf, int bufsize);

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface 0. Returns number of bytes.

void UsbHostCdclnxReadFlush(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Flush received data in CDC interface with specified index.

void UsbHostCdcReadFlush();

Flush received data in CDC interface 0.

int UsbHostCdclnxWriteReady(u8 cdc_inx)

cdc_inx ... CDC interface 0..3

Get bytes available for writing to CDC interface with specified index.

int UsbHostCdcWriteReady();

Get bytes available for writing to CDC interface 0.

Bool UsbHostCdclnxWriteChar(u8 cdc_inx, char ch);

cdc_inx ... CDC interface 0..3

ch ... character to write

Write one character to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbHostCdcWriteChar(char ch);

ch ... character to write

Write one character to CDC interface 0, wait until ready. Returns False if device is not connected.

Bool UsbHostCdclnxWriteData(u8 cdc_inx, const void* buf, int len);

cdc_inx ... CDC interface 0..3

buf ... buffer with data to send

len ... length of data

Write data to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbHostCdcWriteData(const void* buf, int len);

buf ... buffer with data to send

len ... length of data

Write data to CDC interface 0, wait until ready. Returns False if device is not connected.

Interface common for host or device

Bool UsbCdcInxIsMounted(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Check if device interface with specified index is mounted.

Bool UsbCdcIsMounted();

Check if device interface 0 is mounted.

int UsbCdcInxReadReady(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Get bytes available for reading from CDC interface with specified index.

int UsbCdcReadReady();

Get bytes available for reading from CDC interface 0.

char UsbCdcInxReadChar(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Read one character from CDC interface with specified index. Returns 0 if not ready.

char UsbCdcReadChar();

Read one character from CDC interface 0. Returns 0 if not ready.

int UsbCdcInxReadData(u8 cdc_inx, void* buf, int bufsize);

cdc_inx ... CDC interface 0..3

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface with specified index. Returns number of bytes.

int UsbCdcReadData(void* buf, int bufsize);

buf ... destination buffer

bufsize ... buffer size

Read data from CDC interface 0. Returns number of bytes.

void UsbCdcInxReadFlush(u8 cdc_inx);

cdc_inx ... CDC interface 0..3

Flush received data in CDC interface with specified index.

void UsbCdcReadFlush();

Flush received data in CDC interface 0.

int UsbCdcInxWriteReady(u8 cdc_inx)

cdc_inx ... CDC interface 0..3

Get bytes available for writing to CDC interface with specified index.

int UsbCdcWriteReady();

Get bytes available for writing to CDC interface 0.

Bool UsbCdcInxWriteChar(u8 cdc_inx, char ch);

cdc_inx ... CDC interface 0..3

ch ... character to write

Write one character to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbCdcWriteChar(char ch);

ch ... character to write

Write one character to CDC interface 0, wait until ready. Returns False if device is not connected.

Bool UsbCdcInxWriteData(u8 cdc_inx, const void* buf, int len);

cdc_inx ... CDC interface 0..3

buf ... buffer with data to send

len ... length of data

Write data to CDC interface with specified index, wait until ready. Returns False if device is not connected.

Bool UsbCdcWriteData(const void* buf, int len);

buf ... buffer with data to send

len ... length of data

Write data to CDC interface 0, wait until ready. Returns False if device is not connected.

4.4. USB HID Device

Files: `sdk_usb_dev_hid.h`, `sdk_usb_dev_hid.c`

Config: `USE_USB_DEV_HID` (default 0)

The USB HID Device interface is used to connect to a host as a device such as a keyboard, mouse or joystick. Configuration is done by setting the `USE_USB_DEV_HID` configuration parameter to 1 or 2. If the value is 2, the device connects as a combined keyboard + mouse interface. With a value of 1, you can select whether the device connects as a keyboard, mouse, joystick, gamepad or power control.

Table to convert ASCII character (0x00..0x7f) to HID key code:

```
modifier = (AsciiToHidKey[char*2] != 0) ? USB_KEY_MODI_LSHIFT : 0;  
key_code = AsciiToHidKey[char*2 + 1];  
ASCII_TO_HIDKEY_NUM 0x80  number of codes  
const u8 AsciiToHidKey[2*ASCII_TO_HIDKEY_NUM];  
  
void UsbDevInit(&UsbDevHidSetupDesc2);  
Initialize USB HID device as combined device keyboard and mouse. USE_USB_DEV_HID configuration parameter should be set to 2.  
  
void UsbDevInit(&UsbDevHidSetupDescKeyb);  
Initialize USB HID device as keyboard. USE_USB_DEV_HID configuration parameter should be set to 1.  
  
void UsbDevInit(&UsbDevHidSetupDescMouse);  
Initialize USB HID device as mouse. USE_USB_DEV_HID configuration parameter should be set to 1.  
  
void UsbDevInit(&UsbDevHidSetupDescJoystick);  
Initialize USB HID device as joystick. USE_USB_DEV_HID configuration parameter should be set to 1.  
  
void UsbDevInit(&UsbDevHidSetupDescGamepad);  
Initialize USB HID device as gamepad. USE_USB_DEV_HID configuration parameter should be set to 1.  
  
void UsbDevInit(&UsbDevHidSetupDescPower);  
Initialize USB HID device as power control. USE_USB_DEV_HID configuration parameter should be set to 1.  
  
Bool UsbDevHidInxIsMounted(u8 hid_inx);  
hid_inx ... interface index
```

Check if device interface of specified index is mounted.

Bool UsbDevHidIsMounted();

Check if device interface 0 is mounted.

Bool UsbDevHidInxSendReady(u8 hid_inx);

hid_inx ... interface index

Check if next report can be sent on specified interface.

Bool UsbDevHidSendReady();

Check if next report can be sent on interface 0.

Bool UsbDevHidSendReport(u8 ift_num, const void* buf, u8 len, u8 rep_id);

ift_num ... HID interface number

buf ... buffer with report structure

len ... report length in bytes

rep_id ... report ID (0 = do not use)

Send report to specified interface. Check `UsbDevHidInxSendReady()` if next report can be sent. Returns False if cannot send.

Bool UsbDevHidSendKeyRep(const sUsbHidKey* rep);

rep ... keyboard report

Send keyboard report. Returns False on error.

Bool UsbDevHidSendKey(u8 key, u8 modi);

key ... key code to send

modi ... mask of modifiers **USB_KEY_MODI_***

Send keyboard key. Returns False on error. Output speed is 30 keys/sec.

Bool UsbDevHidSendChar(char ch);

ch ... ASCII character

Send keyboard character. Returns False on error. Output speed is 30 chars/sec. The target computer must have an English keyboard layout in order to maintain the correct character mapping.

Bool UsbDevHidSendText(const char* txt);

txt ... ASCII text

Send keyboard text. Returns False on error. Output speed is 30 chars/sec. The target computer must have an English keyboard layout in order to maintain the correct character mapping.

Bool UsbDevHidSendMouse(const sUsbHidMouse* rep);

rep ... mouse report

Send mouse report. Returns False on error.

Bool UsbDevHidSendMouse(s8 dx, s8 dy, Bool left, Bool right, Bool mid);

dx ... relative X coordinate

dy ... relative Y coordinate

left ... left mouse button

right ... right mouse button

mid ... middle mouse button

Send mouse. Returns False on error. Output speed is max 1000 rep/sec.

Bool UsbDevHidSendJoy(const sUsbHidJoy* rep);

rep ... joystick report

Send joystick report. Returns False on error.

Bool UsbDevHidSendPad(const sUsbHidPad* rep);

rep ... gamepad report

Send gamepad report. Returns False on error.

Bool UsbDevHidSendPwr(const sUsbHidPwr* rep);

rep ... power report

Send power report. Returns False on error.

Keyboard modifiers

USB_KEY_MODI_LCTRL	B0	left Control
USB_KEY_MODI_LSHIFT	B1	left Shift
USB_KEY_MODI_LALT	B2	left Alt
USB_KEY_MODI_LWIN	B3	left Window (GUI)
USB_KEY_MODI_RCTRL	B4	right Control
USB_KEY_MODI_RSHIFT	B5	right Shift
USB_KEY_MODI_RALT	B6	right Alt
USB_KEY_MODI_RWIN	B7	right Window (GUI)

Keyboard report

(3 to 8 bytes; 6KRO = 6-key rollover; used by BIOS at boot time; 8 bytes)

typedef struct {

u8 modi; // 0: (1) modifiers mask USB_KEY_MODI_* (1 = pressed)

```

u8    res;      // 1: (1) ... reserved, set to 0
u8    key[6];   // 2: (6) key codes of currently pressed keys (0 = not pressed)
               // If user will press more than 6 keys, key will be
               // reported with phantom code HID_KEY_ERR_ROLLOVER=0x01
} sUsbHidKey;

```

Mouse buttons bitmap

USB_MOUSE_BTN_LEFT	B0	left button
USB_MOUSE_BTN_RIGHT	B1	right button
USB_MOUSE_BTN_MID	B2	middle button
USB_MOUSE_BTN_BACK	B3	backward button
USB_MOUSE_BTN_FOR	B4	forward button

Mouse report (USB mice is polled at 125 Hz = 8 ms; 5 bytes)

```

typedef struct {
    u8    btn;      // 0: (1) buttons mask for currently pressed buttons
    s8    x;        // 1: (1) delta X movement
    s8    y;        // 2: (1) delta Y movement
    s8    wheel;   // 3: (1) delta wheel movement
    s8    pan;     // 4: (1) AC pan
} sUsbHidMouse;

```

Joystick report (4 bytes)

```

typedef struct PACKED {
    s8    x;        // 0: (1) X position
    s8    y;        // 1: (1) Y position
    u8    btn;     // 2: (1) buttons
                  //     bit 0..3: hat switch
                  //     bit 4..7: button 1..4
    s8    throt;   // 3: (1) throttle
} sUsbHidJoy;

```

Gamepad report (11 bytes)

```

typedef struct PACKED {
    s8    x;        // 0: (1) delta x movement
    s8    y;        // 1: (1) delta y movement
    s8    z;        // 2: (1) delta z movement
    ...
} sUsbHidJoy;

```

```
s8    rz;      // 3: (1) delta Rz rotation
s8    rx;      // 4: (1) delta Rx rotation
s8    ry;      // 5: (1) delta Ry rotation
u8    hat;     // 6: (1) 8-bit hat direction switch
u32   btn;     // 7: (4) buttons mask for currently pressed buttons
} sUsbHidPad;
```

Power control report (1 byte)

```
typedef struct PACKED {
    u8    val;    // value 0..3 (0=do nothing, 1=power off, 2 standby,
                  3 wakeup host)
} sUsbHidPwr;
```

4.5. USB HID Host

Files: `sdk_usb_host_hid.h`, `sdk_usb_host_hid.c`

Config: `USE_USB_HOST_HID` (default 0)

The USB HID Host interface allows devices such as an external keyboard or external mouse to be connected. Configuration is done by setting the `USE_USB_HOST_HID` configuration switch to a value of 1 or more, which represents the maximum number of HID interfaces served. It is recommended to use a value of 4 because keyboards have multiple interfaces (including power control).

void UsbHostInit();

USB init in host mode. Set `USE_USB_HOST_HID` to 4.

void UsbTerm();

Terminate USB interface.

Bool UsbHostHidInxIsMounted(u8 cdc_inx);

hid_inx ... interface index

Check if device interface with specified index is mounted.

Bool UsbHostHidIsMounted();

Check if device interface 0 is mounted.

Bool UsbHostHidInxSendReady(u8 hid_inx);

hid_inx ... interface index

Check if next report can be sent to send buffer of interface with specified index.

Bool UsbHostHidSendReady();

Check if next report can be sent to send buffer of interface 0.

Bool UsbHostHidSendReport(u8 hid_inx, const void* buf, u8 len, u8 rep_id);

hid_inx ... interface index

buf ... buffer with report structure

len ... report length in bytes, including `rep_id` byte if it is used

rep_id ... report ID (0 = do not use)

Send report (check `UsbHostHidInxSendReady()` if next report can be sent). Returns False if cannot send.

Bool UsbKeyIsMounted();

Check if USB keyboard is mounted.

Bool UsbMouseIsMounted();

Check if USB mouse is mounted.

u32 UsbGetKey();

Get USB key. Returns u32 packed keyboard event, or 0 if no key.

bit 0..7: key code **HID_KEY_*** (**NOKEY** = no valid key code)

bit 8..15: modifiers **USB_KEY_MODI_***

bit 16..23: ASCII character **CH_*** (**NOCHAR** = no valid character)

u32 UsbGetKeyRel();

Get USB key, including release key. Returns u32 packed keyboard event, or 0 if no key.

bit 0..7: key code **HID_KEY_*** (**NOKEY** = no valid key code)

bit 8..15: modifiers **USB_KEY_MODI_***

bit 16..23: ASCII character **CH_*** (**NOCHAR** = no valid character or release key)

bit 24: 0=press key, 1=release key

In case of key release, ASCII character is invalid (= 0).

u16 UsbKeyToScan(u32 key);

key ... USB key, as returned from UsbGetKeyRel() function (including release flag)

Convert USB key to PC scan code. Output:

bit 0..6: PC key scan code (0=invalid)

bit 7: 1=release key

bit 8: 1=extended key

See file sdk_usb_hid.h for list of PC scan codes **PC_KEYSCAN_***.

Bool UsbKeyIsPressed(u8 key)

key ... **HID_KEY_*** key code

Check if key is pressed (including modifiers).

char UsbGetChar();

Get USB character. Returns **NOCHAR** if no character.

void UsbFlushKey();

Flush USB keys.

u32 UsbGetMouse();

Get USB mouse. Returns u32 packed mouse event, or 0 if no mouse event.

bit 0..7: buttons mask **USB_MOUSE_BTN_***, B7 is set to indicate valid mouse event

bit 8..15: delta X movement (signed s8)

bit 16..23: delta Y movement (signed s8)
bit 24..31: delta wheel movement (signed s8)

Bool UsbKeyNoPressed();

Check no pressed key.

void UsbKeyWaitNoPressed();

Wait for no key pressed.

4.6. USB Device

Files: `sdk_usb_dev.h, sdk_usb_dev.c`

Config: `USE_USB_DEV (default 0)`

USB Device is a layer for base handling USB devices. It is used internally by USB device drivers and you need it only if you want to write a new USB device driver. USB Device is an intermediate layer between the device driver and the physical USB layer.

USB_PID

USB product ID. Devices with different interface must have different product ID, or PC will report system error. The USB_PID is made up of device configuration switches, indicating the number of each device interface.

```
u8 UsbDevAddress;           // device assigned address (>0 device is addressed)
Bool UsbDevConnected;      // device is connected (= 1st SETUP packet received)
Bool UsbDevSuspended;      // device is suspended
Bool UsbDevSelfPowered;    // device is self powered (updated by UsbDevSetCfg)
Bool UsbDevWakeUpEn;       // remote wake up host by driver is enabled
u8 UsbDevSetupBuff[64];     // setup buffer

// device class driver
typedef struct {
    void    (*init());        // initialize class driver
    void    (*term());        // terminate class driver
    void    (*reset());       // reset class driver (device is unplugged)
    u16    (*open(const sUsbDescltf* itf, u16 max_len));
                           // open device class interface
                           // (returns size of used interface, 0=not supported)
    Bool   (*ctrl(u8 stage)); // process control transfer complete
                           // (returns False to stall)
    void   (*comp(u8 epinx, u8 xres, u16 len)); // process data transfer complete
    void   (*sof(u16 sof)); // receiving SOF (start of frame; NULL=not used)
    void   (*sched());       // schedule driver (NULL=not used,
                           // raised with UsbRescheduleDrv())
} sUsbDevDrv;
```

Utilities

Bool UsbIsAddressed();

Check if device is addressed (has assigned device address),

Bool UsbIsConnected();

Check if device is connected to host (= 1st SETUP packet has been received).

Bool UsbIsSuspended();

Check if device is suspended.

Bool UsbIsSelfPowered();

Check if device is self powered

void UsbDevSofEnable();

Enable SOF interrupt.

void UsbDevSofDisable();

Disable SOF interrupt (if not used by drivers).

void UsbDevWakeup();

Request host to wake up (only if suspended and if wakeup is enabled). In case of remote wake up, no RESUME signal will be received, we must resume by SOF packet.

void UsbDevSetStall(u8 epinx);

epinx ... endpoint index 0..31

Send STALL signal to the host.

void UsbDevClrStall(u8 epinx);

epinx ... endpoint index 0..31

Clear STALL request.

Process SETUP control request

const sUsbDescCfg* UsbDevGetDescCfg(u8 cfg);

cfg ... configuration number (1...) as passed by configuration descriptor

Get configuration descriptor by configuration number (returns NULL if not found).

const sUsbDescCfg* UsbDevGetDescCfgInx(u8 cfginx);

cfginx ... configuration index 0..

Get configuration descriptor by configuration index (returns NULL if not found).

Bool UsbDevGetDesc();

Process "get descriptor" request (returns False to stall).

Bool UsbDevSetCfg(u8 cfg);

cfg ... configuration number (1...) as passed by configuration descriptor

Process "set configure" request. Returns False = stall control endpoint. Function finds interface and map it to driver.

void UsbDevSetupAck();

Acknowledging at Status Stage.

Bool UsbDevClassCtrl(const sUsbDrv* drv);

drv ... pointer to device driver

Invoke class driver control request at SETUP stage, set continue to DATA stage. Returns False to stall.

void UsbDevSetupStart(void* buffer, u16 len);

Start transfer data to/from control endpoint (in required SETUP direction). In some special cases data to be sent can be in UsbDevSetupBuff buffer (ASCII text descriptor).

Bool UsbDevSetupReq();

Process SETUP control request (start of SETUP transaction). Returns False = stall control endpoint.

Transfer complete

void UsbDevSetupNext();

Queue next transaction in Data Stage. In some special cases data to be sent can be in UsbDevSetupBuff buffer (ASCII text descriptor).

void UsbDevSetupComp(u8 epinx, u8 xres, u16 len);

epinx ... endpoint index 0 or 1

xres ... transfer result **USB_XRES_***

len ... transferred bytes

Process SETUP transfer complete (DATA or STATUS stage).

void UsbDevComp(u8 epinx, u8 xres);

epinx ... endpoint index 0..31

xres ... transfer result **USB_XRES_***

Process "transfer complete" event after transfer all buffers.

Interrupt service

void UsbDevResume();

Resume device (used internally from UsbDevIrq).

void UsbDevIrq();

USB device IRQ handler.

Initialize

Should be called from CPU0.

void UsbDevEpReset();

Reset all non-control endpoints.

void UsbDevCfgReset();

Reset configuration of USB device drivers - resets drivers and clears base variables. Leaves SETUP data untouched.

void UsbDevSetupReset();

Reset SETUP data.

void UsbDevReset();

Reset USB device drivers - resets drivers and clears base variables, resets SETUP data.

void UsbDevEpInit(u8 epinx, u16 pktmax, u8 xfer);

epinx ... endpoint index 0..31

pktmax ... max. packet size

xfer ... transfer type **USB_XFER_***

Initialize device endpoint.

void UsbDevEpOpen(const sUsbDescEp* desc);

desc ... endpoint request descriptor

Initialize device endpoint by endpoint descriptor.

Bool UsbDevOpenEpPair(const u8* p_desc, u8 ep_count, u8* epinx_out, u8* epinx_in);

p_desc ... pointer to endpoint descriptors

ep_count ... number of endpoints (1 or 2)

epinx_out ... pointer to endpoint index OUT

epinx_in ... pointer to endpoint index IN

Parse endpoint descriptors as IN/OUT pair. Returns False to stall.

void UsbDevInit(const sUsbDevSetupDesc* desc);

desc ... pointer to application device setup descriptor

USB init in device mode.

4.7. USB Host

Files: `sdk_usb_host.h`, `sdk_usb_host.c`

Config: `USE_USB_HOST` (default 0)

USB Host is a layer for base handling USB host drivers. It is used internally by USB host drivers and you need it only if you want to write a new USB host driver. USB Host is an intermediate layer between the host driver and the physical USB layer.

Host enumeration state

<code>USB_HOST_ENUM_IDLE</code>	idle, no enumeration
<code>USB_HOST_ENUM_RESET_START</code>	start reset signal
<code>USB_HOST_ENUM_RESET_STOP</code>	stop reset signal (D-/D+ goes to SE0 for 10 ms)
<code>USB_HOST_ENUM_START</code>	wait to start of enumeration
<code>USB_HOST_ENUM_SET_ADDR</code>	set address
<code>USB_HOST_ENUM_SET_ADDRW</code>	delay after set address
<code>USB_HOST_ENUM_DEV_DESC</code>	get device descriptor
<code>USB_HOST_ENUM_CFG_DESC</code>	get base configuration descriptor
<code>USB_HOST_ENUM_CFG_FULL</code>	get full configuration
<code>USB_HOST_ENUM_SET_CFG</code>	set config
<code>USB_HOST_ENUM_STOP</code>	stop enumeration

<code>u8 UsbHostSetupBuff[320];</code>	setup buffer for host
<code>u8 UsbHostSetupStage;</code>	host setup stage <code>USB_STAGE_*</code>
<code>u8 UsbHostEnumState;</code>	current state of device enumeration (0 = not enumerating)
<code>u8 UsbHostEnumAddr;</code>	address of enumerated device

Host class driver

```
typedef struct {
    void    (*init());      // initialize class driver
    void    (*term());      // terminate class driver
    u16    (*open(u8 dev_addr, const sUsbDescltf* ift, u16 max_len);
                // open class interface (returns size
                // of used interface, 0=not supported)
    Bool    (*cfg(u8 dev_addr, u8 ift_num);
                // configure interface (NULL = not used)
                // Function cfg() can setup device. After it,
                function UsbHostCfgComp() will be called,
```

```

with highest interface number of the driver.

Bool (*comp(u8 dev_addr, u8 dev_epinx, u8 xres, u16 len);
           // transfer complete callback (dev_epinx =
           device endpoint index 0..31)

void (*close(u8 dev_addr);      // close device
void (*sof(u16 sof); // sending SOF (start of frame; NULL=not used)
void (*sched();    // schedule driver (NULL=not used,
                  raised with UsbRescheduleDrv()))

} sUsbHostDrv;

```

Utilities

void UsbDevSofEnable();

Enable SOF interrupt.

void UsbDevSofDisable();

Disable SOF interrupt (if not used by drivers).

Bool UsbHostCheckAddr(u8 dev_addr);

dev_addr ... device address

Check if device address is valid.

sUsbHostDev* UsbHostGetDev(u8 dev_addr);

dev_addr ... device address

Get device structure. Returns dev[0] if the address is invalid, as a fallback treatment, since in most cases the address is already checked. To check the validity of the address, use UsbHostCheckAddr() function.

u8 UsbGetHubSpeed(u8 dev_addr);

dev_addr ... device address

Get hub (or root) device speed. Returns **USB_SPEED_FS** or **USB_SPEED_LS**.

Bool UsbNeedPreamble(u8 dev_addr);

dev_addr ... device address

Check if device needs preamble.

u8 UsbHostDevEp(u8 dev_addr, u8 dev_epinx);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

Get endpoint from device address and device endpoint index. Returns **USB_DRVID_INVALID** if not found. Tries opposite direction as well.

Bool UsbHostIsBusy(u8 dev_addr, u8 dev_epinx);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

Check if transfer is busy (active).

Bool UsbHostEpClaim(u8 dev_addr, u8 dev_epinx);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

Claim host endpoint by class driver. Returns False if cannot be claimed - already claimed or still active.

void UsbHostEpUnclaim(u8 dev_addr, u8 dev_epinx);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

Unclaim host endpoint by class driver.

Transfer complete

void UsbHostSetupIdle(u8 dev_addr, u8 xres);

dev_addr ... device address

xres ... transfer result **USB_XRES_***

Invoke SETUP transfer complete callback and idle. If OK, UsbSetupDataXfer contains length of data, UsbSetupRequest contains request packet.

void UsbHostSetupComp(u8 dev_addr, u8 dev_epinx, u8 xres, u16 len);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

xres ... transfer result **USB_XRES_***

len ... length of transferred data

Process SETUP transfer complete on endpoint 0.

void UsbHostComp(u8 ep, u8 xres);

ep ... endpoint 0..15

xres ... transfer result **USB_XRES_***

Proces "transfer completed" event after transfer all buffers.

Device Enumeration

Bool UsbHostParseCfg(u8 dev_addr, const u8* p_desc);

dev_addr ... device address

Parse configuration descriptor. Returns False on error.

void UsbHostEnumStop();

Stop unfinished enumeration.

Bool UsbHostGetDesc(u8 dev_addr, u8 desc_type, u8 inx, u16 lang, void* buf, u16 len, pUsbHostSetupCompCB cb);

dev_addr ... device address

desc_type ... descriptor type

inx ... descriptor index

lang ... language ID

buf ... data buffer to receive descriptor

len ... length of data

cb ... callback

Get descriptor from device.

void UsbHostCfgComp(u8 dev_addr, u8 ift_num);

dev_addr ... device address

ift_num ... current interface number (Completion continues on
the following interface, starting with 0xff)

Set config complete. This function can be called from drv->cfg functions, to continue to next driver.

void UsbHostEnum(u8 dev_addr, u8 xres);

dev_addr ... device address

xres ... transfer result **USB_XRES_***

Process enumeration.

Interrupt service

void UsbHostOnTime();

USB host timer. Called from SysTick_Handler() every 5 ms.

void UsbHostDevRemove(u8 hub_addr, u8 hub_port);

hub_addr ... hub address, 0 = roothub

hub_port ... hub port, 0 = all devices on downstream hub

Remove connected device.

void UsbHostIrq();

USB host IRQ handler.

Initialize

void UsbHostEpInit(u8 ep, u8 dev_addr, u8 dev_epinx, u16 pktmax, u8 xfer, u8 inter);

ep ... endpoint number 0..15, corresponding to interrupt endpoint number

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

pktmax ... max. packet size (limited to 64 bytes)

xfer ... transfer type **USB_XFER_***

inter ... interval the host controller should poll this endpoint, in [ms]

(Re)initialize host endpoint.

u8 UsbHostEpAlloc(u8 xfer);

Allocate endpoint. Returns ep0 for transfers **USB_XFER_CTRL**.

u8 UsbHostEpOpen(u8 dev_addr, const sUsbDescEp* desc);

dev_addr ... device address

desc ... endpoint descriptor

Initialize host endpoint by endpoint descriptor. Returns endpoint number 0..15.

void UsbHostEp0Open(u8 dev_addr, u16 pktmax);

dev_addr ... device address

pktmax ... size of data

Initialize host control endpoint 0 to transfer data.

void UsbHostDevClose(u8 dev_addr);

dev_addr ... device address

Close all opened endpoints belong to this device.

void UsbHostClrDev(u8 dev_addr);

dev_addr ... device address

Clear device.

Bool UsbHostOpenEpPair(u8 dev_addr, const u8* p_desc, u8 ep_count, u8* epinx_out, u8* epinx_in, u16* epout_max, u16* epin_max);

dev_addr ... device address

p_desc ... pointer to endpoint descriptors

ep_count ... number of endpoints (1 or 2)

epinx_out ... pointer to endpoint index OUT

epinx_in ... pointer to endpoint index IN

epout_max ... max. size of packet OUT

epin_max ... max. size of packet IN

Parse 2 endpoint descriptors as IN/OUT pairs. Returns False on error.

void UsbHostInit();

USB init in host mode.

Transfer

void UsbHostXferStart(u8 dev_addr, u8 dev_epinx, u8* buf, u16 len, Bool use_ring);

dev_addr ... device address

dev_epinx ... device endpoint index 0..31

buf ... pointer to data buffer, or pointer to ring buffer

len ... length of data

use_ring ... use ring buffer (buf is pointer to sRing ring buffer filled with 'len' data)

Start host transfer.

void UsbHostSetupSend(u8 dev_addr);

dev_addr ... device address

Send setup packet. Setup packet is in UsbSetupRequest buffer.

void UsbHostCtrlXfer(u8 dev_addr, void* buf, pUsbHostSetupCompCB cb);

dev_addr ... device address

buf ... pointer to data buffer

cb ... callback

Start control transfer with callback. UsbSetupRequest contains filled setup packet, UsbSetupDataXfer will contain data length.

4.8. USB Physical

Files: `sdk_usb_phy.h`, `sdk_usb_phy.c`

Config: `USE_USB` (default 0)

USB Physical is base layer for handling USB hardware. It is used internally by USB host and device drivers and you need it only if you want to write a new USB driver.

<code>USB_ENDPOINT_NUM</code>	16	total number of endpoints (index 0..15)
<code>USB_SETUP_PKT_SIZE</code>	8	size of setup packet
<code>USB_PACKET_MAX</code>	64	max. size of normal packet
<code>USB_DIR_IN</code>	0	IN packet direction (from device to host)
<code>USB_DIR_OUT</code>	1	OUT packet direction (from host to device)
<code>USB_SPEED_OFF</code>	0	device speed - disconnected
<code>#define USB_SPEED_LS</code>	1	device speed - low speed (1.5 Mbps)
<code>#define USB_SPEED_FS</code>	2	device speed - full speed (12 Mbps)

`sEndpoint UsbEndpoints[32];` endpoint descriptors

`sUsbSetupPkt UsbSetupRequest;` setup request packet, saved in SETUP stage

`Bool UsbHostMode;` host mode (or device mode otherwise)

`u8 UsbDevCfgNum;` current active configuration (0 = not configured)

`Bool UsbDevIsInit;` device is initialized

`Bool UsbHostIsInit;` host is initialized

USB_EPINX(ep, dir)

Convert endpoint number (0..15) and direction (`USB_DIR_*`) to endpoint index.

USB_DIR(epinx)

Get endpoint direction `USB_DIR_*` from endpoint index.

USB_EP(epinx)

Get endpoint number 0..15 from endpoint index.

USB endpoint descriptor

Device: The index into the `UsbEndpoints` array is also an index into the control registers and can be used to derive the endpoint number (bits 1..4) and direction (bit 0).

Host: Host uses only endpoint descriptors with index 0..15. Endpoints 1..15 correspond to interrupt number 1..15, endpoint 0 corresponds to control endpoint EPX.

`typedef struct {`

```

    Bool          used;           // this endpoint is used ("configured")
    volatile Bool active;        // transfer is active (busy)
    volatile Bool claimed;      // endpoint is claimed by driver
    Bool          stalled;       // endpoint is stalled
    Bool          use_ring;      // use ring buffer instead of fixed buffer
    Bool          rx;            // receive direction of transfer
    u8            dev_addr;      // host: device address (0..127)
    u8            dev_epinx;     // host: device endpoint index
    u8            next_pid;      // next packet identification PID (0 or 1)
    u8            xfer;          // transfer type USB_XFER_*
    u16           pktmax;        // max. packet size
    u16           rem_len;       // remaining length of data to transfer
    u16           xfer_len;      // length of already transferred data
    u8*           data_buf;      // pointer to USB data buffer in USB DPRAM
    void*         user_buf;      // pointer to user buffer or ring buffer sRing
    volatile u32* ep_ctrl;       // endpoint control register
    volatile u32* buf_ctrl;     // buffer control register
} sEndpoint;

```

Utilities

u16 UsbGetSof();

Read SOF (start of frame number).

Bool UsbIsMounted();

Check if device is mounted (configured, UsbDevCfgNum > 0).

Bool UsbEpIsBusy(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Check if endpoint is active (busy).

void UsbIntEnable();

Interrupt enable.

void UsbIntDisable();

Interrupt disable.

void UsbDevConnect();

Connect device to USB bus.

void UsbDevDisconnect();

Disconnect device from USB bus.

u8 UsbGetDevSpeed();

Det device speed (returns **USB_SPEED_***).

Bool UsbEpClaim(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Claim endpoint by class driver. Returns False if cannot be claimed - already claimed or still active.

void UsbEpUnclaim(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Unclaim endpoint by class driver.

void UsbRescheduleDrv();

Request to reschedule interrupt from class driver.

void UsbRescheduleTimer();

Host: request to reschedule interrupt from timer.

Transfer control

void UsbDelay12();

Delay 12 system ticks. Needed before setting bit to start transmission.

u32 UsbNextPrep(u8 epinx, u8 bufid);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

bufid ... buffer ID 0 or 1

Prepare transfer of one buffer. Returns control word.

void UsbNextBuf(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Start transfer of next endpoint buffer.

void UsbXferStart(u8 epinx, void* buf, u16 len, Bool use_ring);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

buf ... pointer to user buffer or pointer to ring buffer

len ... length of data

use_ring ... use ring buffer (buf is pointer to sRing ring buffer with 'len' data)

Start new transfer.

u16 UsbXferSync(u8 epinx, u8 bufid);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

bufid ... buffer ID 0 or 1

Synchronize one completed buffer. Returns number of transferred bytes.

Bool UsbXferCont(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Transfer continue. Returns True if transfer is complete.

Initialize

u8* UsbRamAlloc(int size);

size ... size of buffer

Allocate RAM buffer. Returns NULL on error. Size is aligned to 64 bytes.

void UsbRamFree(u8* addr, int size);

addr ... address of buffer

size ... size of buffer

Deallocate RAM buffer previously allocated by UsbRamAlloc(). Address can be NULL.

void UsbEpTerm(u8 epinx);

epinx ... endpoint index 0..31 (device) or 0..15 (host)

Terminate endpoint.

void UsbEpBindDrv(u8* ep2drv, const u8* p_desc, u16 desc_len, u8 drv_id);

ep2drv ... mapping array of size USB_ENDPOINT_NUM2

p_desc ... pointer to interface descriptor

desc_len ... max. length of descriptors

drv_id ... driver index

Bind endpoints to driver.

void UsbTerm();

USB terminate.

void UsbPhyInit();

Base USB init. Common part for device and host.

5. Big Integers

The Big Integers library is used for calculations with large integers. It was created primarily out of the need to calculate Bernoulli numbers, which are needed to calculate coefficients for calculating linear factorials. Big Integer library can be found at folder `_lib\bigint`.

The library is also available as a PC version on following link, and includes a database of 5000 even Bernoulli numbers B2 to B10000:

https://www.breatharian.eu/sw/bigint/index_en.html

<https://github.com/Panda381/BigInt>

5.1. Big Integers and Bernoulli

Files: bigint.h, bigint.c, bernoulli.c

Config: USE_BIGINT (default 1)

The Big Integer library is activated by setting the USE_BIGINT configuration parameter to 1. In addition, it uses the BIGINT_BERN_NUM configuration parameter specifying the size of the Bernoulli number table. This parameter is preset to 512 meaning that the table contains pre-generated Bernoulli numbers B2 to B1024. With this table size, linear factorials can be calculated for real numbers up to real8192 (size of the number is 1024 bytes).

BIGINT_BERN_NUM	1024	number of table Bernoulli numbers
BIGINT_BASE_BITS	32	number of bits of base segment
BIGINT_BASE	u32	type of base segment unsigned
BIGINT_BASES	s32	type of base segment signed
BIGINT_BASE_BYTES	4	number of bytes per base segment

Big integer - constant

```
typedef struct {
    const BIGINT_BASE*   data;      // array of segments
    int                  num;       // number of segments (0=zero number)
    Bool                sign;      // sign flag
} cbigint;
```

Pre-generated Bernoulli numbers:

```
const cbigint bern_num[BIGINT_BERN_NUM];           // numerators
const cbigint bern_den[BIGINT_BERN_NUM];           // denominators
```

Big integer

```
typedef struct {
```

BIGINT_BASE*	data;	// array of segments
int	num;	// number of valid segments (0=zero number)
int	max;	// maximum allocated segments
Bool	sign;	// sign flag

```
} bigint;
```

void BigIntInit(bigint* num);

num ... pointer to bigint

Initialize big integer.

void BigIntTerm(bigint* num);

num ... pointer to bigint

Terminate big integer.

bigint* BigIntGetArr(int num);

num ... number of bitint numbers in the array

Allocate array of bigint numbers. Returns pointer to array.

void BigIntFreeArr(bigint* arr, int num);

arr ... array of bigint numbers

num ... number of bitint numbers in the array

Release array of bigint numbers.

bigint* BigIntResizeArr(bigint* arr, int oldnum, int newnum);

arr ... array of bigint numbers

oldnum ... old number of bigint numbers in the array

newnum ... new number of bigint numbers in the array

Resize array of bigint numbers. Returns new address of array.

Bool BigIntSetSize(bigint* num, int n);

num ... pointer to bigint

n ... new number of segments

Set number of segments. Content will be lost. Returns False on memory error.

void BigIntResize(bigint* num, int n);

num ... pointer to bigint

n ... new number of segments

Resize number of segments. Content will be preserved. New segments will be cleared.
Memory error is not handled.

void BigIntReduce(bigint* num);

num ... pointer to bigint

Reduce - delete zero segments.

void BigIntCopy(bigint* num, const bigint* src);

num ... pointer to destination bigint

src ... pointer to source bigint

Copy from another bigint number.

void BigIntCopyC(bigint* num, const cbigint* src);

num ... pointer to destination bigint

src ... pointer to constant source cbigint

Copy from constant cbigint number.

void BigIntExch(bigint* num1, bigint* num2);

num1 ... pointer to 1st bigint

num2 ... pointer to 2nd bigint

Exchange numbers.

int BigIntComp(const bigint* num1, const bigint* num2);

num1 ... pointer to 1st bigint

num2 ... pointer to 2nd bigint

Compare numbers. Result 1 if num1>num2, 0 if num1==num2, -1 if num1<num2.

int BigIntCompAbs(const bigint* num1, const bigint* num2);

num1 ... pointer to 1st bigint

num2 ... pointer to 2nd bigint

Compare absolute value of numbers. Result 1 if num1>num2, 0 if num1==num2, -1 if num1<num2.

int BigIntBitLen(const bigint* num);

num ... pointer to bigint

Get bit length of the number (= 1 + bit position of highest '1').

int BigIntCtz(const bigint* num);

num ... pointer to bigint

Get number of lower bits '0' (count trailing zeros).

void BigIntMul2(bigint* num);

num ... pointer to bigint

Shift number 1 bit left (= multiply * 2).

void BigIntDiv2(bigint* num);

num ... pointer to bigint

Shift number 1 bit right (= divide / 2).

void BigIntLeft(bigint* num, int shift);

num ... pointer to bigint

shift ... number of shifts

Shift number more bits left.

void BigIntRight(bigint* num, int shift);

num ... pointer to bigint

shift ... number of shifts

Shift number more bits right.

Bool BigIntIsZero(const bigint* num);

num ... pointer to bigint

Check if number is zero.

Bool BigIntIsNeg(const bigint* num);

num ... pointer to bigint

Check if number is negative.

void BigIntNeg(bigint* num);

num ... pointer to bigint

Negate number.

void BigIntAbs(bigint* num);

num ... pointer to bigint

Absolute value.

Bool BigIntEqInt(const bigint* num, BIGINT_BASES n);

num ... pointer to bigint

Check if number is equal integer value.

void BigIntSet0(bigint* num);

num ... pointer to bigint

Set value 0.

void BigIntSet1(bigint* num);

num ... pointer to bigint

Set value 1.

void BigIntSetInt(bigint* num, BIGINT_BASES n);

num ... pointer to bigint

n ... value to set

Set integer value.

void BigIntAddSub(bigint* num, const bigint* num1, const bigint* num2, Bool sub);

num ... pointer to destination bigint

num1 ... pointer to 1st source bigint

num2 ... pointer to 2nd source bigint

sub ... True to subtract, False to addition

Add/sub two numbers (this = num1 +- num2, operands and destination can be the same).

void BigIntAdd(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint

num1 ... pointer to 1st source bigint

num2 ... pointer to 2nd source bigint

Add two numbers (this = num1 + num2, operands and destination can be the same).

void BigIntSub(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint

num1 ... pointer to 1st source bigint

num2 ... pointer to 2nd source bigint

Subtract two numbers (this = num1 - num2, operands and destination can be the same).

void BigIntMul(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint

num1 ... pointer to 1st source bigint

num2 ... pointer to 2nd source bigint

Multiply two numbers (this = num1 * num2, operands and destination can be the same).

void BigIntDivRem(bigint* num, const bigint* num1, const bigint* num2, bigint* rem);

num ... pointer to destination bigint

num1 ... pointer to 1st source bigint
num2 ... pointer to 2nd source bigint
rem ... pointer to remainder of result (NULL if not required)

Divide two numbers with remainder (this = num1 / num2).

void BigIntDiv(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint
num1 ... pointer to 1st source bigint
num2 ... pointer to 2nd source bigint

Divide two numbers (this = num1 / num2).

void BigIntMod(bigint* num, const bigint* num2);

num ... pointer to destination bigint
num2 ... pointer to 2nd source bigint
rem ... pointer to remainder of result (NULL if not required)

Get modulo - remainder (this = this % num2, remainder will always be ≥ 0).

void BigIntGCDBin(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint
num1 ... pointer to 1st bigint
num2 ... pointer to 2nd bigint

Find greatest common divisor GCD. Uses binary GCD algorithm. Result will always be ≥ 0 .

void BigIntGCDEuc(bigint* num, const bigint* num1, const bigint* num2);

void BigIntGCD(bigint* num, const bigint* num1, const bigint* num2);

num ... pointer to destination bigint
num1 ... pointer to 1st bigint
num2 ... pointer to 2nd bigint

Find greatest common divisor GCD. Uses Euclidean GCD algorithm. Result will be ≥ 0 .

void BigIntMul10(bigint* num, u16 carry);

num ... pointer to bigint
carry ... value of digit 0..9

Multiply number * 10 and add digit.

char BigIntDiv10(bigint* num);

num ... pointer to bigint

Divide number / 10 and return digit.

void BigIntFromText(bigint* num, const char* text);

num ... pointer to destination bigint

text ... source ASCIIZ text

Import number from ASCIIZ text.

int BigIntToText(const bigint* num, char* buf, int size);

num ... pointer to bigint

buf ... pointer to destination buffer

size ... size of destination buffer

Export number to text buffer. Returns number of characters.

int BigIntBernMax();

Get max. index of table Bernoulli number (0 = table not supported).

void BigIntBernNum(bigint* num, int inx);

num ... pointer to destination bigint

inx ... index of Bernoulli number 0..BernMax()

Load table Bernoulli number - numerator.

void BigIntBernDen(bigint* num, int inx);

num ... pointer to destination bigint

inx ... index of Bernoulli number 0..BernMax()

Load table Bernoulli number - denominator.

typedef void (bernonulli_cb(int permille);

Prototype of callback function to indicate Bernoulli progress, in 0..1000 per mille.

void Bernoulli(int n, bigint* numer, bigint* denom, bernonulli_cb cb);

n ... required number of Bernoulli numbers (generate even numbers B2..B2*n)

numer ... pointer to array of 'n' bigint numbers to store result numerators

denom ... pointer to array of 'n' bigint numbers to store result denominators

cb ... callback function to indicate progress (NULL=not used)

Generate array of even Bernoulli numbers as fraction. GetArr() function can be used to allocate arrays of bigint.

6. Real Numbers

The Real Numbers library is used for calculations with large numbers in floating-point format. Real Numbers library can be found at folder `_lib\real`. The library is written to allow easy universal definition of floatint-point numbers in different formats, while being easily portable to other systems. However, the consequence of portability and versatility is that the library is not very optimized for speed and can be considerably slower than e.g. the RP2040 internal double calculations optimized in assembler.

Various methods are commonly used to calculate scientific functions: the Cordic, Taylor series, Chebyshev approximation, Newton method, Mercator series. Cordic is a popular method in calculators because it makes do with bit shifts only, without the need for multiplication. The other popular method is the Chebyshev approximation, which, unlike e.g. the Taylor series, makes do with fast multiplication, without the need to use slow division of numbers. When this library is used in a PC, the Chebyshev approximation is primarily used.

In RP2040, however, the situation is different. It has to save more memory; larger tables of constants may not even fit in Flash memory. But above all - the external Flash memory is rather slow to access. For large tables the cache memory cannot be used and so calculations using table methods (Cordic, Chebyshev) can be very slow. Therefore, the calculation using Taylor series is mainly used. However, instead of full division by the coefficients, accelerated division by small numbers is used and this makes this method relatively fast as well.

Angle unit

<code>UNIT_DEG</code>	0	degrees
<code>UNIT_RAD</code>	1	radians
<code>UNIT_GRAD</code>	2	grads
<code>u8 Unit;</code>	current angle unit <code>UNIT_*</code>	

Radix numeric base

<code>BASE_DEC</code>	0	decimal
<code>BASE_BIN</code>	1	binary
<code>BASE_OCT</code>	2	octal
<code>BASE_HEX</code>	3	hexadecimal
<code>u8 Base;</code>	current numeric radix base <code>BASE_*</code>	

Exponent mode

<code>EXP_AUTO</code>	0	auto mode
<code>EXP_FIX</code>	1	fixed mode (no exponent on small numbers)
<code>EXP_EE</code>	2	exponent mode
<code>EXP_ENG</code>	3	engineering technical mode (exponent multiply of 3)

```
u8 ExpMode;           current exponent mode EXP_*
```

Rounding

```
FIX_OFF      -1    fixed decimals are off
```

```
int Fix;          current fixed decimal places (0..digits, or FIX_OFF=off)
```

6.1. Formats of Real Numbers

Where possible, library tries to follow the IEEE754 number format specifications. The numbers are stored in memory from the lowest bytes of the mantissa to the highest. The highest bit of a number contains the sign of the number. The next lower bits of the number store an unsigned biased exponent.

There are 21 predefined number formats in the library, with number sizes ranging from 16 bits (2 bytes) to 12288 bits (1536 bytes), precision up to 3690 digits. Large formats are limited primarily by memory size. First, the constant tables take up a lot of space in Flash memory, and second, the functions use auxiliary variables in the stack during calculations. Some functions may require stack space for up to 10 variables. For real4096 numbers, this means 5KB and this is already above the stack limit of 4 KB. Therefore, real4096 numbers and large are only useful for simpler calculations, or you may need to enlarge the stack. The real12288 numbers have limited capabilities and are intended only for calculating constants for smaller types of numbers.

Numbers use 2 types of formats. The first format "normal" corresponds to the IEEE754 specification, i.e. the number contains a mantissa, an exponent and a sign in the highest bit. The second format "extended" is an extended format, where the mantissa is extended over the whole number (at the expense of the exponent), mantissa contains implied bit '1' and the exponent with sign is added to the number as an additional segment. In the extended format, the internal computation of scientific functions (Taylor series) takes place, thus ensuring the correct value of the result, without loss of precision during intermediate calculations.

Real number structure

```
typedef struct {  
    BASE    n[BASE_NUM]; // mantissa, exponent and sign  
} REAL;
```

Extended real number structure - used only for constants in Flash

```
typedef struct {  
    BASE    n[BASE_NUM]; // mantissa with implied bit '1'  
    EXP     exp;         // extended exponent and sign bit  
} REALEXT;
```

For operations with extended numbers, the normal form of REAL number is used, with the exponent being passed as an additional parameter and the function returning the resulting exponent. Exponent is in packed format, with sign in highest bit.

Real16

configuration: USE_REAL16
type: real16, extended real16ext
size: 2 bytes, 16 bits
segment size: 16 bits
number of segments: 1
mantissa: 10 bits
precision: 3.31 digits, extended 4.82 digits
exponent: 5 bits
exponent range: +- 4.52
exponent bias: 15
max. integer factorial: 8
function prefix: Real16xxx
reference number: real48

Real32

configuration: USE_REAL32
type: real32, extended real32ext
size: 4 bytes, 32 bits
segment size: 32 bits
number of segments: 1
mantissa: 23 bits
precision: 7.22 digits, extended 9.63 digits
exponent: 8 bits
exponent range: +- 38.23
exponent bias: 127
max. integer factorial: 34
function prefix: Real32xxx
reference number: real64

Real48

configuration: USE_REAL48
type: real48, extended real48ext
size: 6 bytes, 48 bits
segment size: 16 bits
number of segments: 3
mantissa: 37 bits

precision: 11.44 digits, extended 14.45 digits
exponent: 10 bits
exponent range: +- 153.83
exponent bias: 511
max. integer factorial: 98
function prefix: Real48xxx
reference number: real96

Real64

configuration: USE_REAL64
type: real64, extended real64ext
size: 8 bytes, 64 bits
segment size: 32 bits
number of segments: 2
mantissa: 52 bits
precision: 15.95 digits, extended 19.27 digits
exponent: 11 bits
exponent range: +- 307.95
exponent bias: 1023
max. integer factorial: 170
function prefix: Real64xxx
reference number: real128

Real80

configuration: USE_REAL80
type: real80, extended real80ext
size: 10 bytes, 80 bits
segment size: 16 bits
number of segments: 5
mantissa: 64 bits
precision: 19.57 digits, extended 24.08 digits
exponent: 15 bits
exponent range: +- 4931.77
exponent bias: 16383
max. integer factorial: 1754
function prefix: Real80xxx
reference number: real160

Real96

configuration: USE_REAL96
type: real96, extended real96ext
size: 12 bytes, 96 bits
segment size: 32 bits
number of segments: 3
mantissa: 82 bits
precision: 24.99 digits, extended 28.90 digits
exponent: 13 bits
exponent range: +- 1232.72
exponent bias: 4095
max. integer factorial: 536
function prefix: Real96xxx
reference number: real160

Real128

configuration: USE_REAL128
type: real128, extended real128ext
size: 16 bytes, 128 bits
segment size: 32 bits
number of segments: 4
mantissa: 112 bits
precision: 34.02 digits, extended 38.53 digits
exponent: 15 bits
exponent range: +- 4931.77
exponent bias: 16383
max. integer factorial: 1754
function prefix: Real128xxx
reference number: real192

Real160

configuration: USE_REAL160
type: real160, extended real160ext
size: 20 bytes, 160 bits
segment size: 32 bits
number of segments: 5
mantissa: 143 bits

precision: 43.35 digits, extended 48.16 digits
exponent: 16 bits
exponent range: +- 9863.85
exponent bias: 32767
max. integer factorial: 3210
function prefix: Real160xxx
reference number: real256

Real192

configuration: USE_REAL192
type: real192, extended real192ext
size: 24 bytes, 192 bits
segment size: 32 bits
number of segments: 6
mantissa: 174 bits
precision: 52.68 digits, extended 57.80 digits
exponent: 17 bits
exponent range: +- 19728.00
exponent bias: 65535
max. integer factorial: 5910
function prefix: Real192xxx
reference number: real256

Real256

configuration: USE_REAL256
type: real256, extended real256ext
size: 32 bytes, 256 bits
segment size: 32 bits
number of segments: 8
mantissa: 236 bits
precision: 71.34 digits, extended 77.06 digits
exponent: 19 bits
exponent range: +- 78912.91
exponent bias: 262143
max. integer factorial: 20366
function prefix: Real256xxx
reference number: real384

Real384

configuration: USE_REAL384
type: real384, extended real384ext
size: 48 bytes, 384 bits
segment size: 32 bits
number of segments: 12
mantissa: 362 bits
precision: 109.27 digits, extended 115.60 digits
exponent: 21 bits
exponent range: +- 315652.53
exponent bias: 1048575
max. integer factorial: 71421
function prefix: Real384xxx
reference number: real512

Real512

configuration: USE_REAL512
type: real512, extended real512ext
size: 64 bytes, 512 bits
segment size: 32 bits
number of segments: 16
mantissa: 488 bits
precision: 147.20 digits, extended 154.13 digits
exponent: 23 bits
exponent range: +- 1262611.01
exponent bias: 4194303
max. integer factorial: 254016
function prefix: Real512xxx
reference number: real768

Real768

configuration: USE_REAL768
type: real768, extended real768ext
size: 96 bytes, 768 bits
segment size: 32 bits
number of segments: 24
mantissa: 742 bits

precision: 223.67 digits, extended 231.19 digits
exponent: 25 bits
exponent range: +- 5050444.96
exponent bias: 16777215
max. integer factorial: 913846
function prefix: Real768xxx
reference number: real1024

Real1024

configuration: USE_REAL1024
type: real1024, extended real1024ext
size: 128 bytes, 1024 bits
segment size: 32 bits
number of segments: 32
mantissa: 996 bits
precision: 300.13 digits, extended 308.25 digits
exponent: 27 bits
exponent range: +- 20201780.74
exponent bias: 67108863
max. integer factorial: 3318996
function prefix: Real1024xxx
reference number: real1536

Real1536

configuration: USE_REAL1536
type: real1536, extended real1536ext
size: 192 bytes, 1536 bits
segment size: 32 bits
number of segments: 48
mantissa: 1505 bits
precision: 453.35 digits, extended 462.38 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real1536xxx
reference number: real2048

Real2048

configuration: USE_REAL2048
type: real2048, extended real2048ext
size: 256 bytes, 2048 bits
segment size: 32 bits
number of segments: 64
mantissa: 2017 bits
precision: 607.48 digits, extended 616.51 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real2048xxx
reference number: real3072

Real3072

configuration: USE_REAL3072
type: real3072, extended real3072ext
size: 384 bytes, 3072 bits
segment size: 32 bits
number of segments: 96
mantissa: 3041 bits
precision: 915.73 digits, extended 924.76 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real3072xxx
reference number: real4096

Real4096

configuration: USE_REAL4096
type: real4096, extended real4096ext
size: 512 bytes, 4096 bits
segment size: 32 bits
number of segments: 128
mantissa: 4065 bits

precision: 1223.99 digits, extended 1233.02 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real4096xxx
reference number: real6144

Real6144

configuration: USE_REAL6144
type: real6144, extended real6144ext
size: 768 bytes, 6144 bits
segment size: 32 bits
number of segments: 192
mantissa: 6113 bits
precision: 1840.50 digits, extended 1849.53 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real6144xxx
reference number: real8192

Real8192

configuration: USE_REAL8192
type: real8192, extended real8192ext
size: 1024 bytes, 8192 bits
segment size: 32 bits
number of segments: 256
mantissa: 8161 bits
precision: 2457.01 digits, extended 2466.04 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real8192xxx
reference number: real12288

Real12288

configuration: USE_REAL12288
type: real12288, extended real12288ext
size: 1536 bytes, 12288 bits
segment size: 32 bits
number of segments: 384
mantissa: 12257 bits
precision: 3690.03 digits, extended 3699.06 digits
exponent: 30 bits
exponent range: +- 161614248.01
exponent bias: 536870911
max. integer factorial: 23310032
function prefix: Real12288xxx
reference number: real16384

6.2. Constants

For calculations, a set of predefined constants is available in both normal and extended formats. Constants and variables are passed to functions using pointers. For example, the sum of the constants 'Pi' and '1' to the variable 'a':

```
real128 a;  
Real128Add(&a, &Real128Const1, &Real128ConstPi);
```

RealxxxConst0	RealxxxConst0Ext	// 0
RealxxxConst1	RealxxxConst1Ext	// 1
RealxxxConstM1	RealxxxConstM1Ext	// -1
RealxxxConst2	RealxxxConst2Ext	// 2
RealxxxConst3	RealxxxConst3Ext	// 3
RealxxxConst05	RealxxxConst05Ext	// 0.5
RealxxxConstM05	RealxxxConstM05Ext	// -0.5
RealxxxConst075	RealxxxConst075Ext	// 0.75
RealxxxConst83	RealxxxConst83Ext	// 8/3 (2.6666666667)
RealxxxConst10	RealxxxConst10Ext	// 10
RealxxxConst100	RealxxxConst100Ext	// 100
RealxxxConstExpMax	RealxxxConstExpMaxExt	// exp(x) max
RealxxxConstExpMin	RealxxxConstExpMinExt	// exp(x) min
RealxxxConst01	RealxxxConst01Ext	// 0.1 (1e-1)
RealxxxConst001	RealxxxConst001Ext	// 0.01 (1e-2)

RealxxxConst1eM4	RealxxxConst1eM4Ext	// 0.0001 (1e-4)
RealxxxConst1eM8	RealxxxConst1eM8Ext	// 0.00000001 (1e-8)
RealxxxConstLn2	RealxxxConstLn2Ext	// ln(2) (0.693147180559945)
RealxxxConstRLn2	RealxxxConstRLn2Ext	// 1/ln(2) (1.44269504088896)
RealxxxConstLn10	RealxxxConstLn10Ext	// ln(10) (2.30258509299405)
RealxxxConstRLn10	RealxxxConstRLn10Ext	// 1/ln(10) (0.434294481903252)
RealxxxConstLog2	RealxxxConstLog2Ext	// log(2) (0.301029995663981)
RealxxxConstRLog2	RealxxxConstRLog2Ext	// 1/log(2) (3.32192809488736)
RealxxxConstEul	RealxxxConstEulExt	// Eul (2.71828182845905)
RealxxxConstPi05	RealxxxConstPi05Ext	// pi/2 (1.5707963267949)
RealxxxConstMPi05	RealxxxConstMPi05Ext	// -pi/2 (-1.5707963267949)
RealxxxConstPi	RealxxxConstPiExt	// pi (3.14159265358979)
RealxxxConstMPi	RealxxxConstMPiExt	// -pi (-3.14159265358979)
RealxxxConstPi2	RealxxxConstPi2Ext	// pi^2 (6.28318530717959)
RealxxxConstMPi2	RealxxxConstMPi2Ext	// -pi^2 (-6.28318530717959)
RealxxxConstRPi2	RealxxxConstRPi2Ext	// 1/(pi^2) (0.159154943091895)
RealxxxConstLnPi22	RealxxxConstLnPi22Ext	// ln(pi^2)/2 (0.918938533204673)
RealxxxConst180Pi	RealxxxConst180PiExt	// 180/pi (57.2957795130823)
RealxxxConstPi180	RealxxxConstPi180Ext	// pi/180 (0.0174532925199433)
RealxxxConst200Pi	RealxxxConst200PiExt	// 200/pi (63.6619772367581)
RealxxxConstPi200	RealxxxConstPi200Ext	// pi/200 (0.015707963267949)
RealxxxConstPhi	RealxxxConstPhiExt	// phi (sectio aurea)
RealxxxConstSqrt2	RealxxxConstSqrt2Ext	// sqrt(2)
RealxxxConstRSqrt2	RealxxxConstRSqrt2Ext	// 1/sqrt(2)

6.3. Get Setup

A functions used to get parameters of real numbers - typically for displaying information about type of numbers.

int RealxxxRealBytes();

Get number size in bytes.

int RealxxxRealBits();

Get number size in bits.

int RealxxxMantBits();

Get bits of mantissa.

double RealxxxMantDigDecD();

Get number of decimal digits of mantissa (precision), as double number.

int RealxxxMantDigDec();

Get number of decimal digits of mantissa (precision), as integer.

int RealxxxMantDigBin();

Get number of binary digits of mantissa (precision).

int RealxxxMantDigOct();

Get number of octal digits of mantissa (precision).

int RealxxxMantDigHex();

Get number of hexadecimal digits of mantissa (precision).

double RealxxxMantDigDecDExt();

Get number of decimal digits of mantissa in extended format (precision), as double number.

int RealxxxMantDigDecExt();

Get number of decimal digits of mantissa in extended format (precision), as integer.

int RealxxxExpBits();

Get exponent size in number of bits.

BASE RealxxxExpBias();

Get exponent bias.

int RealxxxExpDig();

Get number of digits of exponent (number in decimal format).

double RealxxxExpRangeD();

Get exponent range (max. decimal value), as double number.

BASE RealxxxExpRange();

Get exponent range (max. decimal value), as integer.

int RealxxxBufSizeDec();

Required max. size of edit buffer to decode number in decimal format.

int RealxxxBufSizeBin();

Required max. size of edit buffer to decode number in binary format.

int RealxxxBufSizeOct();

Required max. size of edit buffer to decode number in octal format.

int RealxxxBufSizeHex();

Required max. size of edit buffer to decode number in hexadecimal format.

BASE RealxxxFactMax();

Get max. factorial.

BASE RealxxxFactThres();

Get threshold to use linear factorial instead of integer factorial.

int RealxxxBernMax();

Get max. Bernoulli table index (= 1024).

Bool RealxxxHasRef();

Check if reference number is supported.

int RealxxxChebFncNum(int fnc);

fnc ... function **CHEBCB_***

Get number of Chebyshev coefficients (0 = not supported).

Chebyshev callback functions

CHEBCB_LN 0 Ln()

CHEBCB_EXP 1 Exp()

CHEBCB_SIN 2 Sin()

CHEBCB_ASIN 3 ASin()

CHEBCB_ATAN 4 ATan()

CHEBCB_SQRT 5 Sqrt()

const REALEXT* RealxxxChebCoefTab(int fnc);

fnc ... function **CHEBCB_***

Get pointer to Chebyshev coefficients (NULL = not supported).

int RealxxxCordAtanNum();

Get number of atan coefficients for Cordic (0 = not supported).

6.4. Flags and State Manipulation

Meaningful only in normal format, not extended.

BASE RealxxxGetExp(const REAL* num);

num ... pointer to real number

Get exponent unsigned (with bias).

BASES RealxxxGetExpS(const REAL* num);

num ... pointer to real number

Get exponent signed (without bias).

void RealxxxSetExp(REAL* num, BASE exp);

num ... pointer to real number

exp ... exponent

Set exponent unsigned (with bias).

void RealxxxSetExpS(REAL* num, BASES exp);

num ... pointer to real number

exp ... exponent

Set exponent signed (without bias).

Bool RealxxxIsNeg(const REAL* num);

num ... pointer to real number

Check if number is negative (can be zero or infinity, too).

u8 RealxxxGetSign(const REAL* num);

num ... pointer to real number

Get sign (0 or 1).

void RealxxxSetSign(REAL* num, u8 sign);

num ... pointer to real number

sign ... sign 0 or <>0

Set sign.

void RealxxxCopySign(REAL* num, const REAL* src);

num ... pointer to destination real number

src ... pointer to source real number

Copy sign from other number.

Bool RealxxxIsZero(const REAL* num);

num ... pointer to real number

Check if number is zero (can be positive or negative zero).

Bool RealxxxIsInf(const REAL* num);

num ... pointer to real number

Check if number is infinity (can be positive or negative infinity).

6.5. Set/Get Number

void RealxxxCopy(REAL* num, const REAL* src);

num ... pointer to destination real number

src ... pointer to source real number

Copy number from another number.

void RealxxxExch(REAL* num, REAL* num2);

num ... pointer to 1st real number

num2 ... pointer to 2nd real number

Exchange numbers.

Set Constant

void RealxxxSet0(REAL* num);

num ... pointer to real number

Set value 0.

void RealxxxSet0Ext(REAL* num);

num ... pointer to real number

Set value 0 to extended number. Set exponent to **EXPEXT_0**.

void RealxxxSet1(REAL* num);

num ... pointer to real number

Set value 1.

void RealxxxSet1Ext(REAL* num);

num ... pointer to real number

Set value 1 to extended number. Set exponent to **EXPEXT_1**.

void RealxxxSetM1(REAL* num);

num ... pointer to real number

Set value -1.

void RealxxxSetM1Ext(REAL* num);

num ... pointer to real number

Set value -1 to extended number. Set exponent to EXPEXT_1 | EXPEXT_SIGN.

void RealxxxSet2(REAL* num);

num ... pointer to real number

Set value 2.

void RealxxxSet2Ext(REAL* num);

num ... pointer to real number

Set value 2 to extended number. Set exponent to EXPEXT_1+1.

void RealxxxSet05(REAL* num);

num ... pointer to real number

Set value 0.5.

void RealxxxSet05Ext(REAL* num);

num ... pointer to real number

Set value 0.5 to extended number. Set exponent to EXPEXT_1-1.

void RealxxxSetM05(REAL* num);

num ... pointer to real number

Set value -0.5.

void RealxxxSetM05Ext(REAL* num);

num ... pointer to real number

Set value -0.5 to extended number. Set exponent to (EXPEXT_1-1) | EXPEXT_SIGN.

void RealxxxSetInf(REAL* num);

num ... pointer to real number

Set +infinity value.

void RealxxxSetInfExt(REAL* num);

num ... pointer to real number

Set +infinity value to extended number. Set exponent to EXPEXT_INF.

void RealxxxSetMax(REAL* num);

num ... pointer to real number

Set maximal valid number.

void RealxxxSetMaxExt(REAL* num);

num ... pointer to real number

Set maximal valid number to extended number. Set exponent to **EXPEXT_MAX-1**.

void RealxxxSetMin(REAL* num);

num ... pointer to real number

Set minimal valid number.

void RealxxxSetMinExt(REAL* num);

num ... pointer to real number

Set minimal valid number to extended number. Set exponent to **EXPEXT_0+1**.

Set Unsigned Int

void RealxxxSetUInt(REAL* num, BASE n);

num ... pointer to real number

n ... value to set

Set unsigned integer value.

EXP RealxxxSetUIntExt(REAL* num, BASE n);

num ... pointer to real number

n ... value to set

Set unsigned integer value to extended number. Returns exponent.

void RealxxxSetU8(REAL* num, u8 n);

num ... pointer to real number

n ... value to set

Set u8 integer value.

EXP RealxxxSetU8Ext(REAL* num, u8 n);

num ... pointer to real number

n ... value to set

Set u8 integer value to extended number. Returns exponent.

void RealxxxSetU16(REAL* num, u16 n);

num ... pointer to real number

n ... value to set

Set u16 integer value.

EXP RealxxxSetU16Ext(REAL* num, u16 n);

num ... pointer to real number

n ... value to set

Set u16 integer value to extended number. Returns exponent.

void RealxxxSetU32(REAL* num, u32 n);

num ... pointer to real number

n ... value to set

Set u32 integer value.

EXP RealxxxSetU32Ext(REAL* num, u32 n);

num ... pointer to real number

n ... value to set

Set u32 integer value to extended number. Returns exponent.

void RealxxxSetU64(REAL* num, u64 n);

num ... pointer to real number

n ... value to set

Set u64 integer value.

EXP RealxxxSetU64Ext(REAL* num, u64 n);

num ... pointer to real number

n ... value to set

Set u64 integer value to extended number. Returns exponent.

Set Signed Int

void RealxxxSetSInt(REAL* num, BASES n);

num ... pointer to real number

n ... value to set

Set signed integer value.

EXP RealxxxSetSIntExt(REAL* num, BASES n);

num ... pointer to real number

n ... value to set

Set signed integer value to extended number. Returns exponent with sign.

void RealxxxSetS8(REAL* num, s8 n);

num ... pointer to real number

n ... value to set

Set s8 integer value.

EXP RealxxxSetS8Ext(REAL* num, s8 n);

num ... pointer to real number

n ... value to set

Set s8 integer value to extended number. Returns exponent with sign.

void RealxxxSetS16(REAL* num, s16 n);

num ... pointer to real number

n ... value to set

Set s16 integer value.

EXP RealxxxSetS16Ext(REAL* num, s16 n);

num ... pointer to real number

n ... value to set

Set s16 integer value to extended number. Returns exponent with sign.

void RealxxxSetS32(REAL* num, s32 n);

num ... pointer to real number

n ... value to set

Set s32 integer value.

EXP RealxxxSetS32Ext(REAL* num, s32 n);

num ... pointer to real number

n ... value to set

Set s32 integer value to extended number. Returns exponent with sign.

void RealxxxSetS64(REAL* num, s64 n);

num ... pointer to real number

n ... value to set

Set s64 integer value.

EXP RealxxxSetS64Ext(REAL* num, s64 n);

num ... pointer to real number

n ... value to set

Set s64 integer value to extended number. Returns exponent with sign.

Get Unsigned Int

BASE RealxxxGetUInt_(const REAL* num, Bool abs);

num ... pointer to real number

abs ... True = get absolute value, False = clamp to positive number

Get unsigned integer value, rounded towards zero, absolute or clamp.

BASE RealxxxGetUInt(const REAL* num);

num ... pointer to real number

Get unsigned integer value, rounded towards zero, clamped to positive value.

BASE RealxxxGetUIntAbs(const REAL* num);

num ... pointer to real number

Get unsigned integer value, rounded towards zero, absolute value of the number.

BASE RealxxxGetUIntExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get unsigned integer value from extended number, rounded towards zero, clamped to positive value.

BASE RealxxxGetUIntAbsExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get unsigned integer value from extended number, rounded towards zero, absolute value of the number.

u8 RealxxxGetU8_(const REAL* num, Bool abs);

num ... pointer to real number

abs ... True = get absolute value, False = clamp to positive number

Get u8 integer value, rounded towards zero, absolute or clamp.

u8 RealxxxGetU8(const REAL* num);

num ... pointer to real number

Get u8 integer value, rounded towards zero, clamped to positive value.

u8 RealxxxGetU8Abs(const REAL* num);

num ... pointer to real number

Get u8 integer value, rounded towards zero, absolute value of the number.

u8 RealxxxGetU8Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u8 integer value from extended number, rounded towards zero, clamped to positive value.

u8 RealxxxGetU8AbsExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u8 integer value from extended number, rounded towards zero, absolute value of the number.

u8 RealxxxGetU16_(const REAL* num, Bool abs);

num ... pointer to real number

abs ... True = get absolute value, False = clamp to positive number

Get u16 integer value, rounded towards zero, absolute or clamp.

u8 RealxxxGetU16(const REAL* num);

num ... pointer to real number

Get u16 integer value, rounded towards zero, clamped to positive value.

u8 RealxxxGetU16Abs(const REAL* num);

num ... pointer to real number

Get u16 integer value, rounded towards zero, absolute value of the number.

u8 RealxxxGetU16Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u16 integer value from extended number, rounded towards zero, clamped to positive value.

u8 RealxxxGetU16AbsExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u16 integer value from extended number, rounded towards zero, absolute value of the number.

u8 RealxxxGetU32_(const REAL* num, Bool abs);

num ... pointer to real number

abs ... True = get absolute value, False = clamp to positive number

Get u32 integer value, rounded towards zero, absolute or clamp.

u8 RealxxxGetU32(const REAL* num);

num ... pointer to real number

Get u32 integer value, rounded towards zero, clamped to positive value.

u8 RealxxxGetU32Abs(const REAL* num);

num ... pointer to real number

Get u32 integer value, rounded towards zero, absolute value of the number.

u8 RealxxxGetU32Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u32 integer value from extended number, rounded towards zero, clamped to positive value.

u8 RealxxxGetU32AbsExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u32 integer value from extended number, rounded towards zero, absolute value of the number.

u8 RealxxxGetU64_(const REAL* num, Bool abs);

num ... pointer to real number

abs ... True = get absolute value, False = clamp to positive number

Get u64 integer value, rounded towards zero, absolute or clamp.

u8 RealxxxGetU64(const REAL* num);

num ... pointer to real number

Get u64 integer value, rounded towards zero, clamped to positive value.

u8 RealxxxGetU64Abs(const REAL* num);

num ... pointer to real number

Get u64 integer value, rounded towards zero, absolute value of the number.

u8 RealxxxGetU64Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u64 integer value from extended number, rounded towards zero, clamped to positive value.

u8 RealxxxGetU64AbsExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get u64 integer value from extended number, rounded towards zero, absolute value of the number.

Get Signed Int

BASES RealxxxGetSInt(const REAL* num);

num ... pointer to real number

Get signed integer value, rounded towards zero.

BASES RealxxxGetSIntExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get signed integer value from extended number, rounded towards zero.

s8 RealxxxGetS8(const REAL* num);

num ... pointer to real number

Get s8 integer value, rounded towards zero.

s8 RealxxxGetS8Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get s8 integer value from extended number, rounded towards zero.

s16 RealxxxGetS16(const REAL* num);

num ... pointer to real number

Get s16 integer value, rounded towards zero.

s16 RealxxxGetS16Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get s16 integer value from extended number, rounded towards zero.

s32 RealxxxGetS32(const REAL* num);

num ... pointer to real number

Get s32 integer value, rounded towards zero.

s32 RealxxxGetS32Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get s32 integer value from extended number, rounded towards zero.

s64 RealxxxGetS64(const REAL* num);

num ... pointer to real number

Get s64 integer value, rounded towards zero.

s64 RealxxxGetS64Ext(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Get s64 integer value from extended number, rounded towards zero.

Import

void RealxxxImport(REAL* num, const void* src, int realbytes, int expbits);

num ... pointer to destination number

src ... pointer to source REAL number of another type

realbytes ... size of source REAL number in number of bytes

expbits ... size of exponent in number of bits of source REAL number

Import from REAL number of another type.

EXP RealxxxImportExt(REAL* num, const void* src, int realbytes, int expbits);

num ... pointer to destination extended number

src ... pointer to source REAL normal number of another type

realbytes ... size of source REAL number in number of bytes

expbits ... size of exponent in number of bits of source REAL number

Import from normal REAL number of another type to extended number. Returns exponent with sign.

void RealxxxFromFloat(REAL* num, float n);

num ... pointer to destination number

n ... source float number

Import from float.

void RealxxxFromDouble(REAL* num, double n);

num ... pointer to destination number

n ... source double number

Import from double.

void RealxxxImportRef(REAL* dst, const void* src);

dst ... pointer to destination number

src ... pointer to source REALREF normal reference number
Import number from reference real number.

EXP RealxxxImportRefExt(REAL* dst, const void* src);

dst ... pointer to destination extended number
src ... pointer to source REALREF normal reference number

Import number from normal reference real number to extended number. Returns exponent with sign.

EXP RealxxxImportRefExt2(REAL* dst, const void* src, EXP exp);

dst ... pointer to destination extended number
src ... pointer to source REALREF normal reference number
exp ... exponent with sign

Import number from extended reference real number to extended number. Returns exponent with sign.

void RealxxxImportBigInt(REAL* num, const bigint* src);

num ... pointer to destination number
src ... pointer to source bigint number

Import number from bigint.

EXP RealxxxImportBigIntExt(REAL* num, const bigint* src);

num ... pointer to destination number
src ... pointer to source bigint number

Import number from bigint to extended number. Returns exponent with sign.

void RealxxxImportBigIntC(REAL* num, const cbigint* src);

num ... pointer to destination number
src ... pointer to source cbigint number

Import number from constant cbigint.

EXP RealxxxImportBigIntCExt(REAL* num, const cbigint* src);

num ... pointer to destination number
src ... pointer to source cbigint number

Import number from constant cbigint to extended number. Returns exponent with sign.

Export

float RealxxxToFloat(const REAL* num);

num ... pointer to real number

Export to float.

float RealxxxToFloatExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Export to float from extended number.

double RealxxxToDouble(const REAL* num);

num ... pointer to real number

Export to double.

double RealxxxToDoubleExt(const REAL* num, EXP exp);

num ... pointer to real number

exp ... exponent with sign

Export to double from extended number.

void RealxxxExportRef(const REAL* src, void* dst);

src ... pointer to source number

dst ... pointer to destination REALREF normal reference number

Export number to reference real number.

void RealxxxExportRefExt(const REAL* src, EXP exp, void* dst);

src ... pointer to source extended number

exp ... exponent with sign

dst ... pointer to destination REALREF normal reference number

Export extended number to normal reference real number.

void RealxxxExportBigInt(const REAL* num, bigint* dst);

num ... pointer to source number

dst ... pointer to destination bigint number

Export number to bigint.

Test random

void RealxxxTestRandom(REAL* num, Bool noneg, BASE expmin, BASE expmax);

num ... pointer to destination number

noneg ... no negative number

expmin ... minimal value of exponent (biased)

expmax ... maximal value of exponent (biased)

Generate test random number.

EXP RealxxxTestRandomExt(REAL* num, Bool noneg, EXP expmin, EXP expmax);

num ... pointer to destination number

noneg ... no negative number

expmin ... minimal value of exponent (biased)

expmax ... maximal value of exponent (biased)

Generate test random number. Returns exponent with sign.

6.6. Arithmetics Operations

Destination and source can be the same registers.

Add and Sub

void RealxxxAddSub(REAL* num, const REAL* num1, const REAL* num2, Bool sub);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

sub ... True to subtract, False to addition

Add/sub 2 numbers.

void RealxxxAdd(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Add 2 numbers.

void RealxxxSub(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Subtract 2 numbers.

EXP RealxxxAddExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand
num2 ... pointer to 2nd source operand
exp2 ... exponent with sign of 2nd source operand

Extended add 2 numbers. Returns exponent with sign.

EXP RealxxxSubExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number
num1 ... pointer to 1st source operand
exp1 ... exponent with sign of 1st source operand
num2 ... pointer to 2nd source operand
exp2 ... exponent with sign of 2nd source operand

Extended subtract 2 numbers. Returns exponent with sign.

Bitwise

void RealxxxBit(REAL* num, const REAL* num1, const REAL* num2, u8 oper);

num ... pointer to destination number
num1 ... pointer to 1st source operand
num2 ... pointer to 2nd source operand
oper ... operation **BITOPER_***

Bitwise operation of 2 numbers.

BITOPER_AND	0
BITOPER_OR	1
BITOPER_XOR	2

void RealxxxBitAnd(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number
num1 ... pointer to 1st source operand
num2 ... pointer to 2nd source operand

Bitwise AND operation of 2 numbers.

void RealxxxBitOr(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number
num1 ... pointer to 1st source operand
num2 ... pointer to 2nd source operand

Bitwise OR operation of 2 numbers.

void RealxxxBitXor(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Bitwise XOR operation of 2 numbers.

Multiply

void RealxxxMul(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Multiply 2 numbers.

EXP RealxxxMulExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand

num2 ... pointer to 2nd source operand

exp2 ... exponent with sign of 2nd source operand

Extended multiply 2 numbers. Returns exponent with sign.

void RealxxxMulUInt(REAL* num, const REAL* src, BASE n);

num ... pointer to destination number

src ... pointer to source operand

n ... integer number

Multiply by unsigned integer number u16 or u32.

void RealxxxMulSInt(REAL* num, const REAL* src, BASES n);

num ... pointer to destination number

src ... pointer to source operand

n ... integer number

Multiply by signed integer number s16 or s32.

EXP RealxxxMulUIntExt(REAL* num, const REAL* src, EXP exp, BASE n);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

n ... integer number

Extended multiply by unsigned integer number u16 or u32. Returns exponent with sign.

EXP RealxxxMulISIntExt(REAL* num, const REAL* src, EXP exp, BASES n);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

n ... integer number

Extended multiply by signed integer number s16 or s32. Returns exponent with sign.

Divide

void RealxxxDiv(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Divide 2 numbers (num = num1 / num2).

EXP RealxxxDivExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand

num2 ... pointer to 2nd source operand

exp2 ... exponent with sign of 2nd source operand

Extended divide 2 numbers (num = num1 / num2). Returns exponent with sign.

void RealxxxDivU16(REAL* num, const REAL* src, u16 n);

num ... pointer to destination number

src ... pointer to source operand

n ... integer number

Divide by u16 integer number.

void RealxxxDivS16(REAL* num, const REAL* src, s16 n);

num ... pointer to destination number

src ... pointer to source operand

n ... integer number

Divide by s16 integer number.

EXP RealxxxDivU16Ext(REAL* num, const REAL* src, EXP exp, u16 n);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

n ... integer number

Extended divide by unsigned integer number u16. Returns exponent with sign.

EXP RealxxxDivS16Ext(REAL* num, const REAL* src, EXP exp, s16 n);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

n ... integer number

Extended divide by signed integer number s16. Returns exponent with sign.

Power and Root

void RealxxxPow2(REAL* num, int pow2);

num ... pointer to number

pow2 ... integer power of 2 (number of shifts: left if > 0, right if < 0)

Multiply by integer power of 2 (bit shift left if positive = multiply, or right if negative = divide).

EXP RealxxxPow2Ext(REAL* num, EXP exp, int pow2);

num ... pointer to number

exp ... exponent with sign

pow2 ... integer power of 2 (number of shifts: left if > 0, right if < 0)

Multiply by integer power of 2 (bit shift left if positive = multiply, or right if negative = divide). Returns exponent with sign.

void RealxxxPow(REAL* num, const REAL* base, const REAL* exp);

num ... pointer to destination number

base ... pointer to base source operand

exp ... pointer to exponent source operand

Power of 2 numbers ($y = \text{base}^{\text{exp}}$).

void RealxxxRoot(REAL* num, const REAL* base, const REAL* exp);

num ... pointer to destination number

base ... pointer to base source operand

exp ... pointer to exponent source operand

Power of 2 numbers ($y = \text{base}^{(1/\text{exp})}$).

Modulus

void RealxxxModFloor(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Modulus with floor rounding ($\text{num} = \text{num1} \% \text{num2}$). Result has same sign as divisor num2 (range 0..num2).

EXP RealxxxModFloorExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand

num2 ... pointer to 2nd source operand

exp2 ... exponent with sign of 2nd source operand

Extended modulus with floor rounding ($\text{num} = \text{num1} \% \text{num2}$). Result has same sign as divisor num2 (range 0..num2). Returns exponent with sign.

void RealxxxModTrunc(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Modulus with trunc rounding ($\text{num} = \text{num1} \% \text{num2}$). Result has same sign as dividend num1 (range 0..num2 or 0..-num2).

EXP RealxxxModTruncExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand

num2 ... pointer to 2nd source operand

exp2 ... exponent with sign of 2nd source operand

Extended modulus with trunc rounding ($\text{num} = \text{num1} \% \text{num2}$). Result has same sign as dividend num1 (range 0..num2 or 0..-num2). Returns exponent with sign.

void RealxxxModRound(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Modulus with nearest rounding. Result is in range -num2/2..+num2/2.

EXP RealxxxModRoundExt(REAL* num, const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num ... pointer to destination number
num1 ... pointer to 1st source operand
exp1 ... exponent with sign of 1st source operand
num2 ... pointer to 2nd source operand
exp2 ... exponent with sign of 2nd source operand

Extended modulus with nearest rounding. Result is in range -num2/2..+num2/2. Returns exponent with sign.

Compare

s8 RealxxxComp(const REAL* num1, const REAL* num2);

num1 ... pointer to 1st source operand
num2 ... pointer to 2nd source operand

Compare 2 numbers. Returns -1 if num1<num2, 0 if num1=num2, +1 if num1>num2.

s8 RealxxxCompExt(const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num1 ... pointer to 1st source operand
exp1 ... exponent with sign of 1st source operand
num2 ... pointer to 2nd source operand
exp2 ... exponent with sign of 2nd source operand

Externded compare 2 numbers. Returns -1 if num1<num2, 0 if num1=num2, +1 if num1>num2.

Bool RealxxxCompEps(const REAL* num1, const REAL* num2, int eps);

num1 ... pointer to 1st source operand
num2 ... pointer to 2nd source operand
eps ... difference (in number of bits)

Compare 2 numbers to equality with ignoring small difference.

Bool RealxxxCompEpsExt(const REAL* num1, EXP exp1, const REAL* num2, EXP exp2, int eps);

num1 ... pointer to 1st source operand
exp1 ... exponent with sign of 1st source operand
num2 ... pointer to 2nd source operand
exp2 ... exponent with sign of 2nd source operand

eps ... difference (in number of bits)

Extended compare 2 numbers to equality with ignoring small difference.

int RealxxxCheckDif(const REAL* num1, const REAL* num2);

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Check difference of numbers. Returns number of different bits 0..MANT_BITS.

int RealxxxCheckDifExt(const REAL* num1, EXP exp1, const REAL* num2, EXP exp2);

num1 ... pointer to 1st source operand

exp1 ... exponent with sign of 1st source operand

num2 ... pointer to 2nd source operand

exp2 ... exponent with sign of 2nd source operand

Extended check difference of numbers. Returns number of different bits 0..REAL_BITS.

Greatest Common Divisor

void RealxxxGCD(REAL* num, const REAL* num1, const REAL* num2);

num ... pointer to destination number

num1 ... pointer to 1st source operand

num2 ... pointer to 2nd source operand

Find greatest common divisor (GCD) of two integer numbers.

6.7. Functions

Sign

void RealxxxNeg(REAL* num);

num ... pointer to number

Negate. Can change sign of zero, too.

void RealxxxAbs(REAL* num);

num ... pointer to number

Absolute value.

Signum

void RealxxxSignumFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Set signum. Sets to 0 if number is zero, 1 if positive or -1 if negative.

void RealxxxSignum(REAL* num);

num ... pointer to number

Set signum. Sets to 0 if number is zero, 1 if positive or -1 if negative.

s8 RealxxxComp0(const REAL* num);

num ... pointer to number

Compare number to zero. Returns -1 if num<0, 0 if num=0, +1 if num>0. Alternative function is signum RealxxxSignum(), which sets result to the number.

Square x²

void RealxxxSqrFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Square x².

void RealxxxSqr(REAL* num);

num ... pointer to number

Square x².

EXP RealxxxSqrFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended square x². Returns exponent with sign.

EXP RealxxxSqrExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended square x². Returns exponent with sign.

Reciprocal value 1/x

void RealxxxRecFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Reciprocal value 1/x.

void RealxxxRec(REAL* num);

num ... pointer to number

Reciprocal value 1/x.

EXP RealxxxRecFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended reciprocal value 1/x. Returns exponent with sign.

EXP RealxxxRecExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended reciprocal value 1/x. Returns exponent with sign.

Mul2

void RealxxxMul2(REAL* num);

num ... pointer to number

Multiply by 2 (increment exponent).

EXP RealxxxMul2Ext(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended multiply by 2 (increment exponent). Returns exponent with sign.

Div2

void RealxxxDiv2(REAL* num);

num ... pointer to number

Divide by 2 (decrement exponent).

EXP RealxxxDiv2Ext(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended divide by 2 (decrement exponent). Returns exponent with sign.

Inc

void RealxxxIncFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Increment number (increase by 1).

void RealxxxInc(REAL* num);

num ... pointer to number

Increment number (increase by 1).

EXP RealxxxIncFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended increment number (increase by 1). Returns exponent with sign.

EXP RealxxxIncExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended increment number (increase by 1). Returns exponent with sign.

Dec

void RealxxxDecFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Decrement number (decrease by 1).

void RealxxxDec(REAL* num);

num ... pointer to number

Decrement number (decrease by 1).

EXP RealxxxDecFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended decrement number (decrease by 1). Returns exponent with sign.

EXP RealxxxDecExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended decrement number (decrease by 1). Returns exponent with sign.

Trunc

void RealxxxTrunc(REAL* num);

num ... pointer to number

Truncate, round towards zero (= "integer" function).

EXP RealxxxTruncExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended truncate, round towards zero (= "integer" function). Returns exponent with sign.

void RealxxxTruncFrac(REAL* num);

num ... pointer to number

Truncation fraction "num - trunc(num)". Result is -1..0 for num<0 or 0..+1 for num>=0.

EXP RealxxxTruncFracExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended truncation fraction "num - trunc(num)". Result is -1..0 for num<0 or 0..+1 for num>=0. Returns exponent with sign.

Round

void RealxxxRoundFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Round number to nearest integer. Half is rounded to higher numbers (+1.5 -> 2.0, -1.5 -> -2.0).

void RealxxxRound(REAL* num);

num ... pointer to number

Round number to nearest integer. Half is rounded to higher numbers (+1.5 -> 2.0, -1.5 -> -2.0).

EXP RealxxxRoundFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended round number to nearest integer. Half is rounded to higher numbers (+1.5 -> 2.0, -1.5 -> -2.0). Returns exponent with sign.

EXP RealxxxRoundExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended round number to nearest integer. Half is rounded to higher numbers (+1.5 -> 2.0, -1.5 -> -2.0). Returns exponent with sign.

void RealxxxRoundFrac(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Rounding fraction "num - round(src)" = -0.5..+0.5.

void RealxxxRoundFrac(REAL* num);

num ... pointer to number

Rounding fraction "num - round(num)" = -0.5..+0.5.

EXP RealxxxRoundFracFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended rounding fraction "num - round(src)" = -0.5..+0.5. Returns exponent with sign.

EXP RealxxxRoundFracExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended rounding fraction "num - round(num)" = -0.5..+0.5. Returns exponent with sign.

Floor

void RealxxxFloor(REAL* num);

num ... pointer to number

Rounding down.

EXP RealxxxFloorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended rounding down. Returns exponent with sign.

void RealxxxFloorFrac(REAL* num);

num ... pointer to number

Rounding down fraction "num - floor(num)" = 0..+1.

EXP RealxxxFloorFracExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended rounding down fraction "num - floor(num)" = 0..+1. Returns exponent with sign.

Ceil

void RealxxxCeil(REAL* num);

num ... pointer to number

Rounding up.

EXP RealxxxCeilExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended rounding up. Returns exponent with sign.

void RealxxxCeilFrac(REAL* num);

num ... pointer to number

Rounding up fraction "num - ceil(num)" = -1..0.

EXP RealxxxCeilFracExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended rounding up fraction "num - ceil(num)" = -1..0. Returns exponent with sign.

Check

Bool RealxxxIsInt(const REAL* num);

num ... pointer to number

Check if number is integer.

Bool RealxxxIsIntExt(const REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended check if number is integer.

Bool RealxxxIsOddInt(const REAL* num);

num ... pointer to number

Check if number is odd integer.

Bool RealxxxIsOddIntExt(const REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended check if number is odd integer.

Bool RealxxxIsPow2(const REAL* num);

num ... pointer to number

Check if number is power of 2.

Bool RealxxxIsPow2Ext(const REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended check if number is power of 2.

Convert Angle

void RealxxxToRad(REAL* num);

num ... pointer to number

Convert angle from current angle unit to radians. Current angle unit is in **u8 Unit**.

EXP RealxxxToRadExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended convert angle from current angle unit to radians. Current angle unit is in **u8 Unit**. Returns exponent with sign.

void RealxxxFromRad(REAL* num);

num ... pointer to number

Convert angle from radians to current angle unit. Current angle unit is in **u8 Unit**.

EXP RealxxxFromRadExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended convert angle from radians to current angle unit. Current angle unit is in **u8 Unit**. Returns exponent with sign.

Normalize Angle

void RealxxxNormRadU(REAL* num);

num ... pointer to number

Normalize angle in radians - unsigned, normalize to range 0..PI*2 (0..360°).

EXP RealxxxNormRadUExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended normalize angle in radians - unsigned, normalize to range 0..PI*2 (0..360°). Returns exponent with sign.

void RealxxxNormRadS(REAL* num);

num ... pointer to number

Normalize angle in radians - signed, normalize to range -PI..+PI (-180°..+180°).

EXP RealxxxNormRadSExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended normalize angle in radians - signed, normalize to range -PI..+PI (-180°..+180°). Returns exponent with sign.

6.8. Ln Function

To calculate the natural logarithm it is possible to use 3 implemented methods - Mercator serie, Taylor serie and Chebyshev approximation. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_LN configuration to 2048, if you want to use it (Chebyshev Ln tables are pre-generated up to real2048). Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

Natural logarithm - optimal method

void RealxxxLnFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural logarithm - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

void RealxxxLn(REAL* num);

num ... pointer to number

Natural logarithm - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxLnFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended natural logarithm - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxLnExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural logarithm - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Log10 and Log2

void RealxxxLog10From(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Decimal logarithm.

void RealxxxLog10(REAL* num);

num ... pointer to number

Decimal logarithm.

void RealxxxLog2From(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Binary logarithm.

void RealxxxLog2(REAL* num);

num ... pointer to number

Binary logarithm.

Natural logarithm of 2

void RealxxxLn2(REAL* num);

num ... pointer to destination number

Calculate $\ln(2)$ constant. Used during constant generation, no need to use it normally (generated $\ln(2)$ constant is available).

EXP RealxxxLn2Ext(REAL* num);

num ... pointer to destination number

Extended calculate $\ln(2)$ constant. Returns exponent with sign. Used during constant generation, no need to use it normally (generated $\ln(2)$ constant is available).

Natural logarithm - Mercator serie

void RealxxxLn_MercatorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural logarithm - Mercator serie. It is recommended to rather use the RealxxxLnFrom() function, which has already preset the most optimal method.

void RealxxxLn_Mercator(REAL* num);

num ... pointer to number

Natural logarithm - Mercator serie. It is recommended to rather use the RealxxxLn() function, which has already preset the most optimal method.

EXP RealxxxLn_MercatorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended natural logarithm - Mercator serie. Returns exponent with sign. It is recommended to rather use the RealxxxLnFromExt() function, which has already preset the most optimal method.

EXP RealxxxLn_MercatorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural logarithm - Mercator serie. Returns exponent with sign. It is recommended to rather use the RealxxxLnExt() function, which has already preset the most optimal method.

Natural logarithm - Taylor serie

void RealxxxLn_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural logarithm - Taylor serie. It is recommended to rather use the RealxxxLnFrom() function, which has already preset the most optimal method.

void RealxxxLn_Taylor(REAL* num);

num ... pointer to number

Natural logarithm - Taylor serie. It is recommended to rather use the RealxxxLn() function, which has already preset the most optimal method.

EXP RealxxxLn_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended natural logarithm - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxLnFromExt() function, which has already preset the most optimal method.

EXP RealxxxLn_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural logarithm - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxLnExt() function, which has already preset the most optimal method.

Natural logarithm - Chebyshev approximation

void RealxxxLn_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural logarithm - Chebyshev approximation. It is recommended to rather use the RealxxxLnFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_LN configuration to 2048, if you want to use it (Chebyshev Ln tables are pre-generated up to real2048).

void RealxxxLn_Chebyshev(REAL* num);

num ... pointer to number

Natural logarithm - Chebyshev approximation. It is recommended to rather use the RealxxxLn() function, which has already preset the most optimal method. In the PicoLibSDK

library, using of Chebyshev approximation is disabled - set MAXCHEB_LN configuration to 2048, if you want to use it (Chebyshev Ln tables are pre-generated up to real2048).

`EXP RealxxxLn_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);`

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended natural logarithm - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxLnFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_LN configuration to 2048, if you want to use it (Chebyshev Ln tables are pre-generated up to real2048).

`EXP RealxxxLn_ChebyshevExt(REAL* num, EXP exp);`

num ... pointer to number

exp ... exponent with sign

Extended natural logarithm - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxLnExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_LN configuration to 2048, if you want to use it (Chebyshev Ln tables are pre-generated up to real2048).

6.9. Exp Function

To calculate the natural exponent it is possible to use 2 implemented methods - Taylor serie and Chebyshev approximation. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_EXP configuration to 2048, if you want to use it (Chebyshev Exp tables are pre-generated up to real2048). Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

Natural exponent - optimal method

`void RealxxxExpFrom(REAL* num, const REAL* src);`

num ... pointer to destination number

src ... pointer to source operand

Natural exponent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

`void RealxxxExp(REAL* num);`

num ... pointer to number

Natural exponent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxExpFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended natural exponent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxExpExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural exponent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Exp10 and Exp2

void RealxxxExp10From(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Decimal exponent.

void RealxxxExp10(REAL* num);

num ... pointer to number

Decimal exponent.

void RealxxxExp2From(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Binary exponent.

void RealxxxExp2(REAL* num);

num ... pointer to number

Binary exponent.

Natural exponent - Taylor serie

void RealxxxExp_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural exponent - Taylor serie. It is recommended to rather use the RealxxxExpFrom() function, which has already preset the most optimal method.

void RealxxxExp_Taylor(REAL* num);

num ... pointer to number

Natural exponent - Taylor serie. It is recommended to rather use the RealxxxExp() function, which has already preset the most optimal method.

EXP RealxxxExp_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended natural exponent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxExpFromExt() function, which has already preset the most optimal method.

EXP RealxxxExp_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural exponent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxExpExt() function, which has already preset the most optimal method.

Natural exponent - Chebyshev approximation

void RealxxxExp_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural exponent - Chebyshev approximation. It is recommended to rather use the RealxxxExpFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_EXP configuration to 2048, if you want to use it (Chebyshev Exp tables are pre-generated up to real2048).

void RealxxxExp_Chebyshev(REAL* num);

num ... pointer to number

Natural exponent - Chebyshev approximation. It is recommended to rather use the RealxxxExp() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_EXP configuration to 2048, if you want to use it (Chebyshev Exp tables are pre-generated up to real2048).

EXP RealxxxExp_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended natural exponent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxExpFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_EXP configuration to 2048, if you want to use it (Chebyshev Exp tables are pre-generated up to real2048).

EXP RealxxxExp_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural exponent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxExpExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_EXP configuration to 2048, if you want to use it (Chebyshev Exp tables are pre-generated up to real2048).

6.10. Sqrt Function

To calculate the square root it is possible to use 4 implemented methods - logarithm method, Newton iteration, Taylor serie and Chebyshev approximation. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SQRT configuration to 2048, if you want to use it (Chebyshev Sqrt tables are pre-generated up to real2048). Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Newton iteration.

Sqrt (square root) - optimal method

void RealxxxSqrtFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sqrt (square root) - optimal method (in the PicoLibSDK library, optimal method is Newton iteration).

void RealxxxSqrt(REAL* num);

num ... pointer to number

Sqrt (square root) - optimal method (in the PicoLibSDK library, optimal method is Newton iteration).

EXP RealxxxSqrtFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended sqrt (square root) - optimal method (in the PicoLibSDK library, optimal method is Newton iteration). Returns exponent with sign.

EXP RealxxxSqrtExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sqrt (square root) - optimal method (in the PicoLibSDK library, optimal method is Newton iteration). Returns exponent with sign.

Sqrt (square root) - logarithm method

void RealxxxSqrt_LnFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sqrt (square root) - logarithm method. It is recommended to rather use the RealxxxSqrtFrom() function, which has already preset the most optimal method.

void RealxxxSqrt_Ln(REAL* num);

num ... pointer to number

Sqrt (square root) - logarithm method. It is recommended to rather use the RealxxxSqrt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_LnFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended sqrt (square root) - logarithm method. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtFromExt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_LnExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sqrt (square root) - logarithm method. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtExt() function, which has already preset the most optimal method.

Sqrt (square root) - Newton iteration

void RealxxxSqrt_NewtonFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sqrt (square root) - Newton iteration. It is recommended to rather use the RealxxxSqrtFrom() function, which has already preset the most optimal method.

void RealxxxSqrt_Newton(REAL* num);

num ... pointer to number

Sqrt (square root) - Newton iteration. It is recommended to rather use the RealxxxSqrt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_NewtonFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sqrt (square root) - Newton iteration. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtFromExt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_NewtonExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sqrt (square root) - Newton iteration. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtExt() function, which has already preset the most optimal method.

Sqrt (square root) - Taylor serie

void RealxxxSqrt_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sqrt (square root) - Taylor serie. It is recommended to rather use the RealxxxSqrtFrom() function, which has already preset the most optimal method.

void RealxxxSqrt_Taylor(REAL* num);

num ... pointer to number

Sqrt (square root) - Taylor serie. It is recommended to rather use the RealxxxSqrt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sqrt (square root) - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtFromExt() function, which has already preset the most optimal method.

EXP RealxxxSqrt_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sqrt (square root) - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtExt() function, which has already preset the most optimal method.

Sqrt (square root) - Chebyshev approximation

void RealxxxSqrt_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sqrt (square root) - Chebyshev approximation. It is recommended to rather use the RealxxxSqrtFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SQRT configuration to 2048, if you want to use it (Chebyshev Sqrt tables are pre-generated up to real2048).

void RealxxxSqrt_Chebyshev(REAL* num);

num ... pointer to number

Sqrt (square root) - Chebyshev approximation. It is recommended to rather use the RealxxxSqrt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SQRT configuration to 2048, if you want to use it (Chebyshev Sqrt tables are pre-generated up to real2048).

EXP RealxxxSqrt_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sqrt (square root) - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SQRT configuration to 2048, if you want to use it (Chebyshev Sqrt tables are pre-generated up to real2048).

EXP RealxxxSqrt_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sqrt (square root) - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxSqrtExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SQRT configuration to 2048, if you want to use it (Chebyshev Sqrt tables are pre-generated up to real2048).

6.11. Sin Function

To calculate sine it is possible to use 3 implemented methods - Taylor serie, Chebyshev approximation and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie. If you need to generate both sine and cosine, it may be more efficient to use the SinCos function to generate both values at once.

In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048). In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Sine - optimal method

void RealxxxSinFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

void RealxxxSin(REAL* num);

num ... pointer to number

Sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxSinFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxSinExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Sine - Taylor serie

void RealxxxSin_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sine - Taylor serie. It is recommended to rather use the RealxxxSinFrom() function, which has already preset the most optimal method.

void RealxxxSin_Taylor(REAL* num);

num ... pointer to number

Sine - Taylor serie. It is recommended to rather use the RealxxxSin() function, which has already preset the most optimal method.

EXP RealxxxSin_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sine - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxSinFromExt() function, which has already preset the most optimal method.

EXP RealxxxSin_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sine - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxSinExt() function, which has already preset the most optimal method.

Sine - Chebyshev approximation

void RealxxxSin_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sine - Chebyshev approximation. It is recommended to rather use the RealxxxSinFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

void RealxxxSin_Chebyshev(REAL* num);

num ... pointer to number

Sine - Chebyshev approximation. It is recommended to rather use the RealxxxSin() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxSin_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sine - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxSinFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxSin_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sine - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxSinExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

Sine - Cordic method

void RealxxxSin_CordicFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Sine - Cordic method. It is recommended to rather use the RealxxxSinFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

void RealxxxSin_Cordic(REAL* num);

num ... pointer to number

Sine - Cordic method. It is recommended to rather use the RealxxxSin() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxSin_CordicFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sine - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxSinFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxSin_CordicExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended sine - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxSinExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.12. Cos Function

To calculate cosine it is possible to use 3 implemented methods - Taylor serie, Chebyshev approximation and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie. If you need to generate both sine and cosine, it may be more efficient to use the SinCos function to generate both values at once.

In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048). In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Cosine - optimal method

void RealxxxCosFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

void RealxxxCos(REAL* num);

num ... pointer to number

Cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxCosFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxCosExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Cosine - Taylor serie

void RealxxxCos_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cosine - Taylor serie. It is recommended to rather use the RealxxxCosFrom() function, which has already preset the most optimal method.

void RealxxxCos_Taylor(REAL* num);

num ... pointer to number

Cosine - Taylor serie. It is recommended to rather use the RealxxxCos() function, which has already preset the most optimal method.

EXP RealxxxCos_TaylorFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended cosine - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxCosFromExt() function, which has already preset the most optimal method.

EXP RealxxxCos_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cosine - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxCosExt() function, which has already preset the most optimal method.

Cosine - Chebyshev approximation

void RealxxxCos_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cosine - Chebyshev approximation. It is recommended to rather use the RealxxxCosFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

void RealxxxCos_Chebyshev(REAL* num);

num ... pointer to number

Cosine - Chebyshev approximation. It is recommended to rather use the RealxxxCos() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxCos_ChebyshevFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cosine - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxCosFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxCos_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cosine - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxCosExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

Cosine - Cordic method

void RealxxxCos_CordicFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cosine - Cordic method. It is recommended to rather use the RealxxxCosFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

void RealxxxCos_Cordic(REAL* num);

num ... pointer to number

Cosine - Cordic method. It is recommended to rather use the RealxxxCos() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxCos_CordicFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended cosine - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxCosFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxCos_CordicExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cosine - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxCosExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.13. SinCos Function

If you need to generate both sine and cosine, it may be more efficient to use the SinCos function to generate both values at once. To calculate sine/cosine it is possible to use 2 implemented methods - Taylor serie and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Sine/cosine - optimal method

void RealxxxSinCosFrom(REAL* sin_num, REAL* cos_num, const REAL* src);

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

src ... pointer to source operand

Sine/cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). If sin_num and cos_num point to the same number, result is undefined.

EXP RealxxxSinCosFromExt(REAL* sin_num, REAL* cos_num, EXP* cos_exp, const REAL* src, EXP src_exp);

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

cos_exp ... pointer to get exponent of cosine result

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sine/cosine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign of the sine result. If sin_num and cos_num point to the same number, result is undefined.

Sine/cosine - Taylor serie

```
void RealxxxSinCos_TaylorFrom(REAL* sin_num, REAL* cos_num, const  
REAL* src);
```

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

src ... pointer to source operand

Sine/cosine - Taylor serie. If sin_num and cos_num point to the same number, result is undefined. It is recommended to rather use the RealxxxSinCosFrom() function, which has already preset the most optimal method.

```
EXP RealxxxSinCos_TaylorFromExt(REAL* sin_num, REAL* cos_num, EXP*  
cos_exp, const REAL* src, EXP src_exp);
```

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

cos_exp ... pointer to get exponent of cosine result

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended sine/cosine - Taylor serie. Returns exponent with sign of the sine result. If sin_num and cos_num point to the same number, result is undefined. It is recommended to rather use the RealxxxSinCosFromExt() function, which has already preset the most optimal method.

Sine/cosine - Cordic method

```
void RealxxxSinCos_CordicFrom(REAL* sin_num, REAL* cos_num, const  
REAL* src);
```

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

src ... pointer to source operand

Sine/cosine - Cordic method. If sin_num and cos_num point to the same number, result is undefined. It is recommended to rather use the RealxxxSinCosFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

```
EXP RealxxxSinCos_CordicFromExt(REAL* sin_num, REAL* cos_num, EXP*  
cos_exp, const REAL* src, EXP src_exp);
```

sin_num ... pointer to destination number to get sine result

cos_num ... pointer to destination number to get cosine result

cos_exp ... pointer to get exponent of cosine result
src ... pointer to source operand
src_exp ... exponent with sign of source operand

Extended sine/cosine - Cordic method. Returns exponent with sign of the sine result. If sin_num and cos_num point to the same number, result is undefined. It is recommended to rather use the RealxxxSinCosFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.14. Tan Function

To calculate tangent it is possible to use 3 implemented methods - Taylor serie, Chebyshev approximation and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048). In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Tangent - optimal method

void RealxxxTanFrom(REAL* num, const REAL* src);

num ... pointer to destination number
src ... pointer to source operand

Tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

void RealxxxTan(REAL* num);

num ... pointer to number

Tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxTanFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number
src ... pointer to source operand
exp ... exponent with sign of source operand

Extended tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxTanExt(REAL* num, EXP exp);

num ... pointer to number
exp ... exponent with sign

Extended tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Tangent - Taylor serie

void RealxxxTan_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Tangent - Taylor serie. It is recommended to rather use the RealxxxTanFrom() function, which has already preset the most optimal method.

void RealxxxTan_Taylor(REAL* num);

num ... pointer to number

Tangent - Taylor serie. It is recommended to rather use the RealxxxTan() function, which has already preset the most optimal method.

EXP RealxxxTan_TaylorFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended tangent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxTanFromExt() function, which has already preset the most optimal method.

EXP RealxxxTan_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended tangent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxTanExt() function, which has already preset the most optimal method.

Tangent - Chebyshev approximation

void RealxxxTan_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Tangent - Chebyshev approximation. It is recommended to rather use the RealxxxTanFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

void RealxxxTan_Chebyshev(REAL* num);

num ... pointer to number

Tangent - Chebyshev approximation. It is recommended to rather use the RealxxxTan() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxTan_ChebyshevFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended tangent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxTanFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxTan_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended tangent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxTanExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

Tangent - Cordic method

void RealxxxTan_CordicFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Tangent - Cordic method. It is recommended to rather use the RealxxxTanFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

void RealxxxTan_Cordic(REAL* num);

num ... pointer to number

Tangent - Cordic method. It is recommended to rather use the RealxxxTan() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxTan_CordicFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended tangent - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxTanFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxTan_CordicExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended tangent - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxTanExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.15. CoTan Function

To calculate cotangent it is possible to use 3 implemented methods - Taylor serie, Chebyshev approximation and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048). In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Cotangent - optimal method

void RealxxxCoTanFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cotangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

void RealxxxCoTan(REAL* num);

num ... pointer to number

Cotangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie).

EXP RealxxxCoTanFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cotangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

EXP RealxxxCoTanExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cotangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Returns exponent with sign.

Cotangent - Taylor serie

void RealxxxCoTan_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cotangent - Taylor serie. It is recommended to rather use the RealxxxCoTanFrom() function, which has already preset the most optimal method.

void RealxxxCoTan_Taylor(REAL* num);

num ... pointer to number

Cotangent - Taylor serie. It is recommended to rather use the RealxxxCoTan() function, which has already preset the most optimal method.

EXP RealxxxCoTan_TaylorFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cotangent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanFromExt() function, which has already preset the most optimal method.

EXP RealxxxCoTan_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cotangent - Taylor serie. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanExt() function, which has already preset the most optimal method.

Cotangent - Chebyshev approximation

void RealxxxCoTan_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cotangent - Chebyshev approximation. It is recommended to rather use the RealxxxCoTanFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

void RealxxxCoTan_Chebyshev(REAL* num);

num ... pointer to number

Cotangent - Chebyshev approximation. It is recommended to rather use the RealxxxCoTan() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxCoTan_ChebyshevFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cotangent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

EXP RealxxxCoTan_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended cotangent - Chebyshev approximation. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_SIN configuration to 2048, if you want to use it (Chebyshev Sin tables are pre-generated up to real2048).

Cotangent - Cordic method

void RealxxxCoTan_CordicFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Cotangent - Cordic method. It is recommended to rather use the RealxxxCoTanFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

```
void RealxxxCoTan_Cordic(REAL* num);
```

num ... pointer to number

Cotangent - Cordic method. It is recommended to rather use the RealxxxCoTan() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

```
EXP RealxxxCoTan_CordicFromExt(REAL* num, const REAL* src, EXP exp);
```

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended cotangent - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

```
EXP RealxxxCoTan_CordicExt(REAL* num, EXP exp);
```

num ... pointer to number

exp ... exponent with sign

Extended cotangent - Cordic method. Returns exponent with sign. It is recommended to rather use the RealxxxCoTanExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.16. ASin Function

To calculate arcus sine it is possible to use 3 implemented methods - Taylor serie, Chebyshev approximation and Cordic method. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie.

In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ASIN configuration to 2048, if you want to use it (Chebyshev ASin tables are pre-generated up to real2048). In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192).

Arcus sine - optimal method

```
void RealxxxASinFrom(REAL* num, const REAL* src);
```

num ... pointer to destination number

src ... pointer to source operand

Arcus sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2.

void RealxxxASin(REAL* num);

num ... pointer to number

Arcus sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2.

EXP RealxxxASinFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2. Returns exponent with sign.

EXP RealxxxASinExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus sine - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2. Returns exponent with sign.

Generate constant pi

void RealxxxPi(REAL* num);

num ... pointer to number

Generate constant pi (using Taylor serie, $\pi=6\cdot\arcsin(0.5)$). Used during constant generation, no need to use it normally (generated pi constant is available).

EXP RealxxxPiExt(REAL* num);

num ... pointer to number

Generate extended constant pi (using Taylor serie, $\pi=6\cdot\arcsin(0.5)$). Returns exponent with sign. Used during constant generation, no need to use it normally (generated pi constant is available).

Arcus sine - Taylor serie

void RealxxxASin_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus sine - Taylor serie. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxASinFrom() function, which has already preset the most optimal method.

void RealxxxASin_Taylor(REAL* num);

num ... pointer to number

Arcus sine - Taylor serie. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxASin() function, which has already preset the most optimal method.

EXP RealxxxASin_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended arcus sine - Taylor serie. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxASinFromExt() function, which has already preset the most optimal method.

EXP RealxxxASin_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus sine - Taylor serie. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxASinExt() function, which has already preset the most optimal method.

Arcus sine - Chebyshev approximation

void RealxxxASin_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus sine - Chebyshev approximation. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxASinFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ASIN configuration to 2048, if you want to use it (Chebyshev ASin tables are pre-generated up to real2048).

void RealxxxASin_Chebyshev(REAL* num);

num ... pointer to number

Arcus sine - Chebyshev approximation. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxASin() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ASIN configuration to 2048, if you want to use it (Chebyshev ASin tables are pre-generated up to real2048).

EXP RealxxxASin_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended arcus sine - Chebyshev approximation. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxASinFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ASIN configuration to 2048, if you want to use it (Chebyshev ASin tables are pre-generated up to real2048).

EXP RealxxxASin_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus sine - Chebyshev approximation. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxASinExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ASIN configuration to 2048, if you want to use it (Chebyshev ASin tables are pre-generated up to real2048).

Arcus sine - Cordic method

void RealxxxASin_CordicFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus sine - Cordic method. Result is in range -PI/2..+PI/2. This function is very inaccurate for some values - better not to use it at all. It is recommended to rather use the RealxxxASinFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

void RealxxxASin_Cordic(REAL* num);

num ... pointer to number

Arcus sine - Cordic method. Result is in range -PI/2..+PI/2. This function is very inaccurate for some values - better not to use it at all. It is recommended to rather use the RealxxxASin() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxASin_CordicFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

src_exp ... exponent with sign of source operand

Extended arcus sine - Cordic method. Result is in range -PI/2..+PI/2. Returns exponent with sign. This function is very inaccurate for some values - better not to use it at all. It is recommended to rather use the RealxxxASinFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set

MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

EXP RealxxxASin_CordicExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus sine - Cordic method. Result is in range -PI/2..+PI/2. Returns exponent with sign. This function is very inaccurate for some values - better not to use it at all. It is recommended to rather use the RealxxxASinExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Cordic method is disabled - set MAXCORD_ATAN configuration to 8192, if you want to use it (Cordic ATan table is pre-generated up to real8192). Cordic method has lower accuracy than Taylor series.

6.17. ACos Function

When calculating the arcus cosine, the calculation method cannot be selected as with the previous functions. The optimal method for calculating the arcus sine is used, with the result shifted by PI/2.

void RealxxxACosFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus cosine. Result is in range 0..PI.

void RealxxxACos(REAL* num);

num ... pointer to number

Arcus cosine. Result is in range 0..PI.

EXP RealxxxACosFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus cosine. Result is in range 0..PI. Returns exponent with sign.

EXP RealxxxACosExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus cosine. Result is in range 0..PI. Returns exponent with sign.

6.18. ATan Function

To calculate arcus tangent it is possible to use 2 implemented methods - Taylor serie and Chebyshev approximation. Rather than using a function for a specific method, it is recommended to use the following 4 functions which are already preset to the most optimal method. In the PicoLibSDK library, optimal method is Taylor serie. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ATAN configuration to 2048, if you want to use it (Chebyshev ATan tables are pre-generated up to real2048).

Arcus tangent - optimal method

void RealxxxATanFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2.

void RealxxxATan(REAL* num);

num ... pointer to number

Arcus tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2.

EXP RealxxxATanFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2. Returns exponent with sign.

EXP RealxxxATanExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus tangent - optimal method (in the PicoLibSDK library, optimal method is Taylor serie). Result is in range -PI/2..+PI/2. Returns exponent with sign.

Arcus tangent - Taylor serie

void RealxxxATan_TaylorFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus tangent - Taylor serie. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxATanFrom() function, which has already preset the most optimal method.

void RealxxxATan_Taylor(REAL* num);

num ... pointer to number

Arcus tangent - Taylor serie. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxATan() function, which has already preset the most optimal method.

EXP RealxxxATan_TaylorFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus tangent - Taylor serie. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxATanFromExt() function, which has already preset the most optimal method.

EXP RealxxxATan_TaylorExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus tangent - Taylor serie. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxATanExt() function, which has already preset the most optimal method.

Arcus tangent - Chebyshev approximation

void RealxxxATan_ChebyshevFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus tangent - Chebyshev approximation. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxATanFrom() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ATAN configuration to 2048, if you want to use it (Chebyshev ATan tables are pre-generated up to real2048).

void RealxxxATan_Chebyshev(REAL* num);

num ... pointer to number

Arcus tangent - Chebyshev approximation. Result is in range -PI/2..+PI/2. It is recommended to rather use the RealxxxATan() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ATAN configuration to 2048, if you want to use it (Chebyshev ATan tables are pre-generated up to real2048).

EXP RealxxxATan_ChebyshevFromExt(REAL* num, const REAL* src, EXP src_exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus tangent - Chebyshev approximation. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxATanFromExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ATAN configuration to 2048, if you want to use it (Chebyshev ATan tables are pre-generated up to real2048).

EXP RealxxxATan_ChebyshevExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus tangent - Chebyshev approximation. Result is in range -PI/2..+PI/2. Returns exponent with sign. It is recommended to rather use the RealxxxATanExt() function, which has already preset the most optimal method. In the PicoLibSDK library, using of Chebyshev approximation is disabled - set MAXCHEB_ATAN configuration to 2048, if you want to use it (Chebyshev ATan tables are pre-generated up to real2048).

6.19. ACoTan Function

When calculating the arcus cotangent, the calculation method cannot be selected as with the previous functions. The optimal method for calculating the arcus tangent is used, with the result shifted.

void RealxxxACoTanFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Arcus cotangent. Result is in range 0..+PI.

void RealxxxACoTan(REAL* num);

num ... pointer to number

Arcus cotangent. Result is in range 0..+PI.

EXP RealxxxACoTanFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended arcus cotangent. Result is in range 0..+PI. Returns exponent with sign.

EXP RealxxxACoTanExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended arcus cotangent. Result is in range 0..+PI. Returns exponent with sign.

6.20. Polar Coordinates

void RealxxxATan2(REAL* num, const REAL* x, const REAL* y);

num ... pointer to destination number

x ... pointer to source number X

y ... pointer to source number Y

Arcus tangent of polar coordinates. Result is in range -PI .. +PI.

EXP RealxxxATan2Ext(REAL* num, const REAL* x, EXP x_exp, const REAL* y, EXP y_exp);

num ... pointer to destination number

x ... pointer to source number X

x_exp .. exponent with sign of source number X

y ... pointer to source number Y

y_exp .. exponent with sign of source number Y

Extended arcus tangent of polar coordinates. Result is in range -PI .. +PI. Returns exponent with sign.

void RealxxxRadius(REAL* num, const REAL* x, const REAL* y);

num ... pointer to destination number

x ... pointer to source number X

y ... pointer to source number Y

Radius of coordinates, radius = $\sqrt{x^2 + y^2}$.

EXP RealxxxRadiusExt(REAL* num, const REAL* x, EXP x_exp, const REAL* y, EXP y_exp);

num ... pointer to destination number

x ... pointer to source number X

x_exp .. exponent with sign of source number X

y ... pointer to source number Y

y_exp .. exponent with sign of source number Y

Extended radius of coordinates, radius = $\sqrt{x^2 + y^2}$. Returns exponent with sign.

void RealxxxCart2Pol(REAL* angle, REAL* radius, const REAL* x, const REAL* y);

angle ... pointer to destination number with angle

radius ... pointer to destination number with radius

x ... pointer to source number X

y ... pointer to source number Y

Convert Cartesian coordinates to polar. Result angle is in range 0..2*PI.

```
void RealxxxCart2PolExt(REAL* angle, EXP* angle_exp, REAL* radius, EXP*
    radius_exp, const REAL* x, EXP x_exp, const REAL* y, EXP y_exp);
```

angle ... pointer to destination number with angle

angle_exp ... pointer to exponent of destination number with angle

radius ... pointer to destination number with radius

radius_exp ... pointer to exponent of destination number with radius

x ... pointer to source number X

x_exp .. exponent with sign of source number X

y ... pointer to source number Y

y_exp .. exponent with sign of source number Y

Extended convert Cartesian coordinates to polar. Result angle is in range 0..2*PI. Returns exponent with sign.

```
void RealxxxPol2Cart(REAL* x, REAL* y, const REAL* angle, const REAL*
    radius);
```

x ... pointer to destination number X

y ... pointer to destination number Y

angle ... pointer to source number with angle

radius ... pointer to source number with radius

Convert polar coordinates to Cartesian.

```
void RealxxxPol2CartExt(REAL* x, EXP* x_exp, REAL* y, EXP* y_exp, const
    REAL* angle, EXP angle_exp, const REAL* radius, EXP radius_exp);
```

x ... pointer to destination number X

x_exp .. pointer to exponent with sign of destination number X

y ... pointer to destination number Y

y_exp .. pointer to exponent with sign of destination number Y

angle ... pointer to source number with angle

angle_exp ... exponent of source number with angle

radius ... pointer to source number with radius

radius_exp ... exponent of source number with radius

Extended convert polar coordinates to Cartesian. Returns exponent with sign.

6.21. Trigonometric Test Function

This trigonometric test function is also known as “The Forensics Evaluation Algorithm”. This is a popular test used to test the accuracy of pocket calculators. In the test, a formula is calculated

`arcsin(arccos(arctan(tan(cos(sin(9))))))`

with the angular measure expressed in degrees. The deviation of the result from the number 9 is used to evaluate the accuracy of the calculator. For more information, see <http://www.datamath.org/Forensics.htm>.

The function uses 9 intermediate numbers in the stack. The function finds the bit position of the result deviation from the correct value of "9" and returns number of bits correctly calculated. Number of correctly calculated digits can be calculated:

$$\text{Number_of_correct_digits} = \text{number_of_correct_bits} * 0.30103 .$$

int RealxxxTest9(REAL* num);

num ... number with result value

Calculate test in normal precision. Returns number or correct bits. Result number of correct digits:

real16: 0.30 digits, lost 3.01 digits (total 3.31 digits)
real32: 2.41 digits, lost 4.82 digits (total 7.22 digits)
real48: 6.32 digits, lost 5.12 digits (total 11.44 digits)
real64: 10.84 digits, lost 5.12 digits (total 15.95 digits)
real80: 14.75 digits, lost 4.82 digits (total 19.57 digits)
real96: 20.47 digits, lost 4.52 digits (total 24.99 digits)
real128: 29.20 digits, lost 4.82 digits (total 34.02 digits)
real160: 38.23 digits, lost 5.12 digits (total 43.35 digits)
real192: 48.16 digits, lost 4.52 digits (total 52.68 digits)
real256: 66.53 digits, lost 4.82 digits (total 71.34 digits)
real384: 104.16 digits, lost 5.12 digits (total 109.27 digits)
real512: 143.29 digits, lost 3.91 digits (total 147.20 digits)
real768: 218.55 digits, lost 5.12 digits (total 223.67 digits)
real1024: 295.01 digits, lost 5.12 digits (total 300.13 digits)
real1536: 448.53 digits, lost 4.82 digits (total 453.35 digits)
real2048: 602.96 digits, lost 4.52 digits (total 607.48 digits)
real3072: 910.31 digits, lost 5.42 digits (total 915.73 digits)
real4096: 1218.87 digits, lost 5.12 digits (total 1223.99 digits)
real6144: 1835.98 digits, lost 4.52 digits (total 1840.50 digits)
real8192: 2452.49 digits, lost 4.52 digits (total 2457.01 digits)
real12288: 3686.71 digits, lost 3.31 digits (total 3690.03 digits)

int RealxxxTest9Ext(REAL* num);

num ... number with result value

Calculate test in extended precision. Returns number or correct bits. Result number of correct digits:

real16ext: 0.00 digits, lost 4.52 digits, lost on normal 3.31 digits (total 4.82 digits)
real32ext: 4.21 digits, lost 5.12 digits, lost on normal 3.01 digits (total 9.63 digits)
real48ext: 9.63 digits, lost 4.52 digits, lost on normal 1.81 digits (total 14.45 digits)
real64ext: 14.15 digits, lost 4.82 digits, lost on normal 1.81 digits (total 19. 27 digits)
real80ext: 18.36 digits, lost 5.42 digits, lost on normal 1.20 digits (total 24. 08 digits)
real96ext: 23.48 digits, lost 5.12 digits, lost on normal 1.51 digits (total 28. 90 digits)
real128ext: 33.41 digits, lost 4.82 digits, lost on normal 0.60 digits (total 38.53 digits)
real160ext: 43.35 digits, lost 4.52 digits, lost on normal 0.00 digits (total 48.16 digits)
real192ext: 53.88 digits, lost 3.61 digits, lost on normal 0.00 digits (total 57.80 digits)
real256ext: 71.65 digits, lost 5.12 digits, lost on normal 0.00 digits (total 77.06 digits)
real384ext: 109.88 digits, lost 5.42 digits, lost on normal 0.00 digits (total 115.60 digits)
real512ext: 148.71 digits, lost 5.12 digits, lost on normal 0.00 digits (total 154.13 digits)
real768ext: 225.47 digits, lost 5.42 digits, lost on normal 0.00 digits (total 231.19 digits)
real1024ext: 303.14 digits, lost 4.82 digits, lost on normal 0.00 digits (total 308.25 digits)
real1536ext: 456.36 digits, lost 5.72 digits, lost on normal 0.00 digits (total 462.38 digits)
real2048ext: 610.19 digits, lost 6.02 digits, lost on normal 0.00 digits (total 616.51 digits)
real3072ext: 919.04 digits, lost 5.42 digits, lost on normal 0.00 digits (total 924.76 digits)
real4096ext: 1227.60 digits, lost 5.12 digits, lost on normal 0.00 digits (total 1233.02 digits)
real6144ext: 1843.21 digits, lost 6.02 digits, lost on normal 0.00 digits (total 1849.53 digits)
real8192ext: 2459.72 digits, lost 6.02 digits, lost on normal 0.00 digits (total 2466.04 digits)
real12288ext: 3685.21 digits, lost 13.55 digits, lost on normal 4.82 digits (total 3699.06 digits)

6.22. Hyperbolic Functions

Hyperbolic sine

void RealxxxSinHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic sine.

void RealxxxSinH(REAL* num);

num ... pointer to number

Hyperbolic sine.

EXP RealxxxSinHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic sine. Returns exponent with sign.

EXP RealxxxSinHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic sine. Returns exponent with sign.

Hyperbolic cosine

void RealxxxCosHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic cosine.

void RealxxxCosh(REAL* num);

num ... pointer to number

Hyperbolic cosine.

EXP RealxxxCosHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic cosine. Returns exponent with sign.

EXP RealxxxCosHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic cosine. Returns exponent with sign.

Hyperbolic tangent

void RealxxxTanHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic tangent.

void RealxxxTanh(REAL* num);

num ... pointer to number

Hyperbolic tangent.

EXP RealxxxTanHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic tangent. Returns exponent with sign.

EXP RealxxxTanHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic tangent. Returns exponent with sign.

Hyperbolic cotangent

void RealxxxCoTanHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic cotangent.

void RealxxxCoTanH(REAL* num);

num ... pointer to number

Hyperbolic cotangent.

EXP RealxxxCoTanHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic cotangent. Returns exponent with sign.

EXP RealxxxCoTanHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic cotangent. Returns exponent with sign.

Hyperbolic secant

void RealxxxSecHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic secant.

void RealxxxSecH(REAL* num);

num ... pointer to number

Hyperbolic secant.

EXP RealxxxSecHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic secant. Returns exponent with sign.

EXP RealxxxSecHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic secant. Returns exponent with sign.

Hyperbolic cosecant

void RealxxxCscHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Hyperbolic cosecant.

void RealxxxCsch(REAL* num);

num ... pointer to number

Hyperbolic cosecant.

EXP RealxxxCscHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended hyperbolic cosecant. Returns exponent with sign.

EXP RealxxxCscHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended hyperbolic cosecant. Returns exponent with sign.

Areasine, inverse hyperbolic sine

void RealxxxArSinHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areasine, inverse hyperbolic sine.

void RealxxxArSinH(REAL* num);

num ... pointer to number

Areasine, inverse hyperbolic sine.

EXP RealxxxArSinHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areasine, inverse hyperbolic sine. Returns exponent with sign.

EXP RealxxxArSinHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areasine, inverse hyperbolic sine. Returns exponent with sign.

Areacosine, inverse hyperbolic cosine

void RealxxxArCosHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areacosine, inverse hyperbolic cosine.

void RealxxxArCosH(REAL* num);

num ... pointer to number

Areacosine, inverse hyperbolic cosine.

EXP RealxxxArCosHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areacosine, inverse hyperbolic cosine. Returns exponent with sign.

EXP RealxxxArCosHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areacosine, inverse hyperbolic cosine. Returns exponent with sign.

Areatangent, inverse hyperbolic tangent

void RealxxxArTanHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areatangent, inverse hyperbolic tangent.

void RealxxxArTanH(REAL* num);

num ... pointer to number

Areatangent, inverse hyperbolic tangent.

EXP RealxxxArTanHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areatangent, inverse hyperbolic tangent. Returns exponent with sign.

EXP RealxxxArTanHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areatangent, inverse hyperbolic tangent. Returns exponent with sign.

Areacotangent, inverse hyperbolic cotangent

void RealxxxArCoTanHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areacotangent, inverse hyperbolic cotangent.

void RealxxxArCoTanH(REAL* num);

num ... pointer to number

Areacotangent, inverse hyperbolic cotangent.

EXP RealxxxArCoTanHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areacotangent, inverse hyperbolic cotangent. Returns exponent with sign.

EXP RealxxxArCoTanHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areacotangent, inverse hyperbolic cotangent. Returns exponent with sign.

Areasecant, inverse hyperbolic secant

void RealxxxArSecHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areasecant, inverse hyperbolic secant.

void RealxxxArSecH(REAL* num);

num ... pointer to number

Areasecant, inverse hyperbolic secant.

EXP RealxxxArSecHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areasecant, inverse hyperbolic secant. Returns exponent with sign.

EXP RealxxxArSecHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areasecant, inverse hyperbolic secant. Returns exponent with sign.

Areacosecant, inverse hyperbolic cosecant

void RealxxxArCscHFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Areacosecant, inverse hyperbolic cosecant.

void RealxxxArCscH(REAL* num);

num ... pointer to number

Areacosecant, inverse hyperbolic cosecant.

EXP RealxxxArCscHFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended areacosecant, inverse hyperbolic cosecant. Returns exponent with sign.

EXP RealxxxArCscHExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended areacosecant, inverse hyperbolic cosecant. Returns exponent with sign.

6.23. Random Numbers

void RealxxxRnd(REAL* num);

num ... pointer to number

Generate random number in range $0 \leq .. < 1$ (resolution max. 64 bits).

EXP RealxxxRndExt(REAL* num);

num ... pointer to number

Generate extended random number in range $0 \leq .. < 1$ (resolution max. 64 bits). Returns exponent with sign.

void RealxxxRndMaxFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Generate random number in range $0 \leq .. < \text{src}$ (resolution max. 64 bits).

void RealxxxRndMax(REAL* num);

num ... pointer to number

Generate random number in range $0 \leq .. < \text{num}$ (resolution max. 64 bits).

EXP RealxxxRndMaxFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Generate extended random number in range $0 \leq .. < \text{src}$ (resolution max. 64 bits). Returns exponent with sign.

EXP RealxxxRndMaxExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Generate extended random number in range $0 \leq \dots < \text{num}$ (resolution max. 64 bits). Returns exponent with sign.

6.24. Chebyshev Approximation

Chebyshev callback functions

CHEBCB_LN 0 Ln()

CHEBCB_EXP 1 Exp()

CHEBCB_SIN 2 Sin()

CHEBCB_ASIN 3 ASin()

CHEBCB_ATAN 4 ATan()

CHEBCB_SQRT 5 Sqrt()

typedef void (Realxxxchebfnc_cb)(REAL* num);

Typedef callback function to be approximated by Chebyshev polynomial. Input number is in range -1 .. +1.

typedef EXP (Realxxxchebfnc_cbext)(REAL* num, EXP exp);

Typedef extended callback function to be approximated by Chebyshev polynomial. Input number is in range -1 .. +1.

typedef void (*chebprogress_cb(int permille));

Typedef callback function to indicate Chebyshev progress, in 0..1000 per mille.

void RealxxxChebCB_Ln(REAL* num);

Chebyshev Ln() callback functions. Input number is in range -1 .. +1.

void RealxxxChebCB_Exp(REAL* num);

Chebyshev Exp() callback functions. Input number is in range -1 .. +1.

void RealxxxChebCB_Sin(REAL* num);

Chebyshev Sin() callback functions. Input number is in range -1 .. +1.

void RealxxxChebCB_ASin(REAL* num);

Chebyshev ASin() callback functions. Input number is in range -1 .. +1.

void RealxxxChebCB_ATan(REAL* num);

Chebyshev ATan() callback functions. Input number is in range -1 .. +1.

void RealxxxChebCB_Sqrt(REAL* num);

Chebyshev Sqrt() callback functions. Input number is in range -1 .. +1.

EXP RealxxxChebCB_LnExt(REAL* num, EXP exp);

Chebyshev extended Ln() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

EXP RealxxxChebCB_ExpExt(REAL* num, EXP exp);

Chebyshev extended Exp() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

EXP RealxxxChebCB_SinExt(REAL* num, EXP exp);

Chebyshev extended Sin() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

EXP RealxxxChebCB_ASinExt(REAL* num, EXP exp);

Chebyshev extended ASin() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

EXP RealxxxChebCB_ATanExt(REAL* num, EXP exp);

Chebyshev extended ATan() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

EXP RealxxxChebCB_SqrtExt(REAL* num, EXP exp);

Chebyshev extended Sqrt() callback functions. Input number is in range -1 .. +1. Returns exponent with sign.

void RealxxxChebCoef(Realxxxchebfnc_cb fnc, int num, REAL* poly, chebprogress_cb cb);

fnc ... callback function to be approximated by Chebyshev polynomial

num ... number of coefficients (=order of Chebyshev polynomial, min. 2)

poly ... pointer to array of 'num' polynomial coefficients

cb ... progress callback function (NULL = none)

Calculate coefficients of Chebyshev approximation. Requires 2*num REAL numbers to be temporary allocated using malloc() function.

void RealxxxChebCoefExt(Realxxxchebfnc_cbext fnc, int num, REAL* poly, EXP* poly_exp, chebprogress_cb cb);

fnc ... callback function to be approximated by Chebyshev polynomial

num ... number of coefficients (=order of Chebyshev polynomial, min. 2)

poly ... pointer to array of 'num' polynomial coefficients,
mantissas of extended format

poly_exp ... pointer to array of 'num' polynomial coefficients,
exponents of extended format

cb ... progress callback function (NULL = none)

Calculate coefficients of Chebyshev approximation in extended precision. Requires 2*num REAL numbers to be temporary allocated using malloc() function.

void RealxxxChebCoefRef(int fnc, int num, REAL* poly, chebprogress_cb cb);

fnc ... index of callback function CHEBCB_*

num ... number of coefficients (=order of Chebyshev polynomial, min. 2)

poly ... pointer to array of polynomial coefficients

cb ... progress callback function (NULL = none)

Calculate coefficients of Chebyshev approximation - precise, using reference REAL numbers. Requires 3*num REALREF numbers to be temporary allocated using malloc() function.

**void RealxxxChebCoefRefExt(int fnc, int num, REAL* poly, EXP* poly_exp,
chebprogress_cb cb);**

fnc ... index of callback function CHEBCB_*

num ... number of coefficients (=order of Chebyshev polynomial, min. 2)

poly ... pointer to array of polynomial coefficients, mantissas of extended format

poly_exp ... pointer to array of polynomial coefficients, exponents of extended format

cb progress callback function (NULL = none)

Calculate coefficients of Chebyshev approximation - precise, using reference REAL numbers and extended destination numbers. Requires 3*num REALREF numbers to be temporary allocated using malloc() function.

void RealxxxChebyshev(REAL* num, int n, const REAL* poly);

num ... pointer to number

n ... number of coefficients (=order of Chebyshev polynomial)

poly ... pointer to array of polynomial coefficients

Calculate value of Chebyshev approximation. Input number must be in range -1 .. +1. Do not use Chebyshev approximation with results near 0, it could be inaccurate.

EXP RealxxxChebyshevExt(REAL* num, EXP exp, int n, const REALEXT* poly);

num ... pointer to number

exp ... exponent with sign

n ... number of coefficients (=order of Chebyshev polynomial)

poly ... pointer to array of polynomial coefficients

Extended calculate value of Chebyshev approximation. Input number must be in range -1 .. +1. Do not use Chebyshev approximation with results near 0, it could be inaccurate. Returns exponent with sign.

EXP RealxxxChebyshevExt2(REAL* num, EXP exp, int n, const REAL* poly, const EXP* poly_exp);

num ... pointer to number

exp ... exponent with sign

n ... number of coefficients (=order of Chebyshev polynomial)

poly ... pointer to array of polynomial coefficients

poly_exp ... pointer to array of exponents of polynomial coefficients

Extended calculate value of Chebyshev approximation. Input number must be in range -1 .. +1. Do not use Chebyshev approximation with results near 0, it could be inaccurate. Returns exponent with sign.

6.25. Bernoulli Numbers

void RealxxxBernDec(REAL* num, int inx);

num ... pointer to number

inx ... index of Bernoulli number 0..BernMax())

Get Bernoulli table number in decimal form (not as a fraction).

void RealxxxBernNum(REAL* num, int inx);

num ... pointer to number

inx ... index of Bernoulli number 0..BernMax())

Get Bernoulli table number - numerator.

void RealxxxBernDen(REAL* num, int inx);

num ... pointer to number

inx ... index of Bernoulli number 0..BernMax())

Get Bernoulli table number - denominator.

6.26. Factorial

Integer factorial (multiplication)

BASE RealxxxFactInt(REAL* num, BASE n);

num ... pointer to destination number

n ... required factorial

Integer factorial. Number is multiplied repeatedly. Returns max. reached number - can be used to detect maximum allowed value of the integral (set n=(BASE)-1 to calculate max. supported integral).

EXP RealxxxFactIntExt(REAL* num, BASE n);

num ... pointer to destination number

n ... required factorial

Extended integer factorial. Number is multiplied repeatedly. Returns exponent with sign.

Linear factorial (approximation)

void RealxxxFactLinFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Non-integer factorial linear approximation.

void RealxxxFactLin(REAL* num);

num ... pointer to number

Non-integer factorial linear approximation.

EXP RealxxxFactLinFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended non-integer factorial linear approximation. Returns exponent with sign.

EXP RealxxxFactLinExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended non-integer factorial linear approximation. Returns exponent with sign.

Optimised factorial (auto select method)

void RealxxxFactFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Factorial - auto select best method.

void RealxxxFact(REAL* num);

num ... pointer to number

Factorial - auto select best method.

EXP RealxxxFactFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended factorial - auto select best method. Returns exponent with sign.

EXP RealxxxFactExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended factorial - auto select best method. Returns exponent with sign.

Natural logarithm of factorial

void RealxxxFactLnFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Natural logarithm of factorial (using approximation).

void RealxxxFactLn(REAL* num);

num ... pointer to number

Natural logarithm of factorial (using approximation).

EXP RealxxxFactLnFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended natural logarithm of factorial (using approximation). Returns exponent with sign.

EXP RealxxxFactLnExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended natural logarithm of factorial (using approximation). Returns exponent with sign.

Decimal logarithm of factorial

void RealxxxFactLogFrom(REAL* num, const REAL* src);

num ... pointer to destination number

src ... pointer to source operand

Decimal logarithm of factorial (using approximation).

void RealxxxFactLog(REAL* num);

num ... pointer to number

Decimal logarithm of factorial (using approximation).

EXP RealxxxFactLogFromExt(REAL* num, const REAL* src, EXP exp);

num ... pointer to destination number

src ... pointer to source operand

exp ... exponent with sign of source operand

Extended decimal logarithm of factorial (using approximation). Returns exponent with sign.

EXP RealxxxFactLogExt(REAL* num, EXP exp);

num ... pointer to number

exp ... exponent with sign

Extended decimal logarithm of factorial (using approximation). Returns exponent with sign.

6.27. From/To Text

void RealxxxFromText(REAL* num, const char* txt);

num ... pointer to destination number

txt ... pointer to source ASCIIZ text

Encode number from ASCIIZ text. Decimal point can be dott '.', comma ',' or B7. Using global flags: **Base** ... current numeric radix base **BASE_***.

int RealxxxToText(const REAL* num, char* buf, int size);

num ... pointer to source number

buf ... pointer to destination buffer (NULL=only count number of characters)

size ... max. length of destination buffer, without terminating 0 (reserve 1 byte for it)

Decode number to text buffer. Returns length of text, without terminating 0 (but it writes terminating 0). Returns length of exponent in **ExpLen**. Returns length of mantissa in **EditLen**. Returns decimal point flag in **PointOn**.

Used global flags:

Base ... current numeric radix base **BASE_***

ExpMode ... current exponent mode **EXP_***

Fix ... current fixed decimal places (0..digits, or **FIX_OFF**=off)

CharPlus ... character of positive number ('+', '' or 0=do not use)

CharExp ... character of exponent ('e', 'E' or 0=do not use)

ExpPlus ... character of positive exponent ('+', '' or 0=do not use)

CharDec ... character used as decimal separator

('.',',' or B7=add flag to previous digit)

RightAlign ... right align text in buffer

MaxDig ... max. number of valid digits (0 = default, negative = cut digits from end)

7. Display Drivers

Display drivers are used to control the hardware of displays. Source files of display drivers are located in the `_display` folder.

7.1. ST7789 TFT

Files: `st7789.c`, `st7789.h`

Config: `USE_ST7789`

The ST7789 TFT display driver supports TFT and LCD displays with ST7789 controller, at a resolution of typically 320x240 pixels, 16-bits per pixel in RGB565 color format. The driver contains a frame buffer into which the graphic output from the program runs. The rendering functions update the changed area. When the drawing is finished, the content of the changed area is transferred to the display by the `DispUpdate()` function.

Configuration

`WIDTH` display width (= 320)

`HEIGHT` display height (= 240)

`FRAMESIZE` frame size in number of colors (= `WIDTH*HEIGHT` = 76800)

`u16 FrameBuf[FRAMESIZE];` frame buffer in RGB 5-6-5 pixel format

`int DispDirtyX1, DispDirtyX2, DispDirtyY1, DispDirtyY2;` dirty window to update

void DispRotation(u8 rot);

`rot` ... display rotation

Set display rotation: 0 Portrait, 1 Landscape, 2 Inverted Portrait, 3 Inverted Landscape.

void DispStartImg(u16 x1, u16 x2, u16 y1, u16 y2);

`x1` ... output window X start coordinate

`x2` ... output window X stop coordinate

`y1` ... output window Y start coordinate

`y2` ... output window Y stop coordinate

LOW level control: start sending image data to display window (`DispSendImg()` must follow).

void DispSendImg(u8 data);

`data` ... data to send to display

LOW level control: send one byte of image data to display (follows after `DispStartImg()`).

void DispSendImg2(u16 data);

data ... data to send to display

LOW level control: send one word of image data to display (follows after DispStartImg()).

void DispStopImg();

LOW level control: stop sending image data (follows after DispStartImg() and DispSendImg()).

void DispSetStrip(int inx);

void DispSetStripNext();

inx ... index of strip back buffer

Set strip of back buffer (-1 = use full FrameBuffer). (only for compatibility with VGA driver)

void DispSetStripOff();

Switch off the back buffer, use only frame buffer to output. (only for compatibility with VGA driver)

void DispLoad();

Load back buffer from frame buffer. (only for compatibility with VGA driver)

void DispDirtyAll();

Set dirty all frame buffer.

void DispDirtyNone();

Set dirty none (clear after update).

void DispDirtyRect(int x, int y, int w, int h);

Update dirty area by rectangle (check valid range).

void DispDirtyPoint(int x, int y);

Update dirty area by one pixel (check valid range).

void DispUpdate();

Update - send dirty window to display.

void DispAutoUpdate(u32 ms);

Auto update after delta time in [ms] of running program.

void DispUpdateAll();

Refresh update all display.

void DispBacklight(u8 backlight);

Display backlight control.

void DispBacklightUpdate();

Display backlight control config update.

void DispInit(u8 rot);

rot ... rotation mode

0 Portrait

1 Landscape

2 Inverted Portrait

3 Inverted Landscape

Initialize display.

void DispTerm();

Terminate display.

void VgaWaitVSync();

Wait for VSync scanline. (only for compatibility with VGA driver)

7.2. Mini-VGA

Files: minivga.h, minivga.h

Config: USE_MINIVGA

The VGA display driver provides output to an external VGA monitor in resolutions 320x240 up to 800x600 pixels, in color bits depths 4, 8, 15 or 16 bits. The output is displayed using a PIO controller and DMA transfer, from the FrameBuf image buffer. When the USE_MINIVGA parameter is set to 2 to 5, an additional buffer, BackBuf, is available in addition to the main image buffer. The additional buffer, BackBuf, is used to render the graphics, with updating of the changed area. After rendering, the contents of the changed area are transferred to the main buffer using the DispUpdate() function.

Buffer BackBuf is used in cases where the graphics need to be drawn sequentially and this would result in flickering of the image. Thus, the graphics are first drawn to the BackBuf buffer and only after complete drawing are transferred to the FrameBuf display buffer. The CPU RAM is limited (264 KB) and cannot accommodate the full size FrameBuf and BackBuf. For this reason, BackBuf can be used reduced in size and rendering is done piecemeal.

The mini-VGA driver can operate in either HSYNC+VSYNC sync mode or CSYNC composite sync mode. With CSYNC composite sync, one output pin can be saved. However, it should be taken into account that some older monitors may not support CSYNC mode. The mode can be set with the VGA_USECSYNC configuration parameter. Setting VGA_USECSYNC to 1 will enable the CSYNC mode. The parameter can either be found in the device configuration or added to the config.h of the project.

Following display resolutions can be used. They are set up by the WIDTH parameter:

WIDTH = 256 ... 256x192 pixels (timings of mode 1024x768 is used)

WIDTH = 320 ... 320x240 pixels (timings of mode 640x480 is used)

WIDTH = 400 ... 400x300 pixels (timings of mode 800x600 is used)

WIDTH = 512 ... 512x384 pixels (timings of mode 1024x768 is used)

WIDTH = 640 ... 640x480 pixels

WIDTH = 800 ... 800x600 pixels

Color formats are determined by the COLBITS setup:

COLBITS = 4 ... YRGB1111 (color format known from PC-CGA or EGA cards)

COLBITS = 8 ... RGB332

COLBITS = 15 ... RGB555

COLBITS = 16 ... RGB565

Following buffer combinations are allowed:

USE_MINIVGA = 1 use only frame buffer, without back buffer

USE_MINIVGA = 2 use back buffer with full size

USE_MINIVGA = 3 use back buffer with 1/2 size

USE_MINIVGA = 4 use back buffer with 1/4 size

USE_MINIVGA = 5 use back buffer with 1/8 size (not supported with 400x300 pixels)

Available combinations of color mode, display resolution and buffer mode:

16 bits RGB565, 15 bits RGB555 (1 pixel = u16; width must be multiple of 2):

256*192 1 buffer 98 KB, 1+1 buffer 196 KB, 1+1/2 buffer 147 KB, 1+1/4 buffer 122 KB, 1+1/8 buffer 110 KB

320*240 1 buffer 153 KB, 1+1/2 buffer 230 KB, 1+1/4 buffer 192 KB, 1+1/8 buffer 172 KB

400*300 1 buffer 240 KB

8 bits RGB332 (1 pixel = u8; width must be multiple of 4):

256*192 1 buffer 49 KB, 1+1 buffer 98 KB, 1+1/2 buffer 73 KB, 1+1/4 buffer 61 KB, 1+1/8 buffer 55 KB

320*240 1 buffer 76 KB, 1+1 buffer 153 KB, 1+1/2 buffer 115 KB, 1+1/4 buffer 96 KB, 1+1/8 buffer 86 KB

400*300 1 buffer 120 KB, 1+1 buffer 240 KB, 1+1/2 buffer 180 KB, 1+1/4 buffer 150 KB

512*384 1 buffer 196 KB, 1+1/4 buffer 245 KB, 1+1/8 buffer 221 KB

4 bits YRGB1111 (1 pixel = 1/2 u8; width must be multiple of 8):

256*192 1 buffer 24 KB, 1+1 buffer 49 KB, 1+1/2 buffer 36 KB, 1+1/4 buffer 30 KB, 1+1/8 buffer 27 KB

320*240 1 buffer 38 KB, 1+1 buffer 76 KB, 1+1/2 buffer 57 KB, 1+1/4 buffer 48 KB, 1+1/8 buffer 43 KB

400*300 1 buffer 60 KB, 1+1 buffer 120 KB, 1+1/2 buffer 90 KB, 1+1/4 buffer 75 KB

512*384 1 buffer 98 KB, 1+1 buffer 196 KB, 1+1/2 buffer 147 KB, 1+1/4 buffer 122 KB, 1+1/8 buffer 110 KB

640*480 1 buffer 153 KB, 1+1/2 buffer 230 KB, 1+1/4 buffer 192 KB, 1+1/8 buffer 172 KB

800*600 1 buffer 240 KB

When drawing is done in stripes, the drawing procedure typically looks like this:

```
int strip;
for (strip = DISP_STRIP_NUM; strip > 0; strip--)
{
    DispSetStripNext();           // next strip
    // DispLoad();                // load back buffer
    ....                          // drawing functions
    DispUpdate();                 // update screen
}
DispSetStripOff();             // set off back buffers
```

The rendering procedure consists of a loop with the number of repetitions of DISP_STRIP_NUM, which represents the number of times the BackBuf fits into the

FrameBuf. The DispSetStripNext() command moves the pointer to the next strip. Following are the Draw rendering functions. On each pass through the loop, the same render commands are executed, but the Draw library trims them so that they only go to the current strip. If the entire image is not redrawn, the DispLoad() function can be called before rendering to restore the contents of the BackBuf buffer from the current FrameBuf strip. After the graphic is rendered, the DispUpdate() function is called to transfer the BackBuf to the current FrameBuf strip. After the rendering loop, the DispSetStripOff() function can be called to turn off the BackBuf strip mode. Subsequent rendering will again be done only to the FrameBuf image buffer (this is the default state when the program starts).

FRAMETYPE FrameBuf[FRAMESIZE]; display buffer

FRAMETYPE BackBuf[BACKBUFSIZE]; back buffer (only if USE_MINIVGA >= 2)

void VgalInit();

Initialize VGA driver (requires system clock **CLK_SYS** to be set to PLL_KHZ). If you initialize the driver from core 0 and want the driver to run on core 1, use the VgaStart() function.

void VgaTerm();

Terminate VGA driver. If you initialized the driver with the VgaStart() function, use the VgaStop() function to terminate.

void VgaCore1Exec(void (*fnc)());

Execute core 1 remote function.

Bool VgaCore1Busy();

Check if core 1 is busy (executing remote function).

void VgaCore1Wait();

Wait if core 1 is busy (executing remote function).

Bool VgalsVSync();

Check VSYNC.

void VgaWaitVSync();

Wait for VSync scanline. In some cases, it may be sufficient to prevent the image from flickering during the drawing by using the VgaWaitVSync() function before the rendering starts, without having to use BackBuf strips for drawing.

void VgaStart();

Start VGA (must be paired with VgaStop()).

void VgaStop();

Terminate VGA (must be paired with VgaStart()).

void VgaRetune(int freq);

freq ... new system frequency in Hz

Retune VGA to different system frequency. Requires double-floating-point libraries.

void DispSetStrip(int inx);

inx ... strip index 0..**VGA_STRIP_NUM**-1, or -1 = use full FrameBuffer

Set strip index of back buffer.

void DispSetStripNext();

Set next strip index of back buffer.

void DispSetStripOff()

Switch off the back buffer. Use only frame buffer to output (same as `DispSetStrip(-1)`).

void DispLoad();

Load back buffer from frame buffer.

void DispDirtyAll();

Set dirty all frame buffer.

void DispDirtyNone();

Set dirty none (clear after update).

void DispDirtyRect(int x, int y, int w, int h);

Update dirty area by rectangle (checks valid range).

void DispDirtyPoint(int x, int y);

Update dirty area by pixel (checks valid range).

void DispUpdate();

Save back buffer to frame buffer.

void DispAutoUpdate(u32 ms);

Auto update after delta time in [ms] of running program.

void DispUpdateAll();

Refresh update all display.

7.3. DVI (HDMI)

Files: dvi.c, dvi.h

Config: USE_DVI

The DVI driver controls the digital video output on the HDMI interface. Only 320x240 pixel, 16-bit RGB565 color resolution is supported. The output is emulated using a 640x480 pixel video mode. The driver requires a processor overclock to 252 MHz. The output is displayed using a PIO controller and DMA transfer, from the FrameBuf image buffer. When the USE_DVI parameter is set to 2 to 5, an additional buffer, BackBuf, is available in addition to the main image buffer. The additional buffer, BackBuf, is used to render the graphics, with updating of the changed area. After rendering, the contents of the changed area are transferred to the main buffer using the DispUpdate() function.

Buffer BackBuf is used in cases where the graphics need to be drawn sequentially and this would result in flickering of the image. Thus, the graphics are first drawn to the BackBuf buffer and only after complete drawing are transferred to the FrameBuf display buffer. The CPU RAM is limited (264 KB) and cannot accommodate the full size FrameBuf and BackBuf. For this reason, BackBuf can be used reduced in size and rendering is done piecemeal.

Following buffer combinations are allowed:

- USE_DVI = 1** use only frame buffer, without back buffer
- USE_DVI = 2** use back buffer with full size
- USE_DVI = 3** use back buffer with 1/2 size
- USE_DVI = 4** use back buffer with 1/4 size
- USE_DVI = 5** use back buffer with 1/8 size

When drawing is done in stripes, the drawing procedure typically looks like this:

```
int strip;  
for (strip = DISP_STRIP_NUM; strip > 0; strip--)  
{  
    DispSetStripNext();      // next strip  
    // DispLoad();           // load back buffer  
    ....                  // drawing functions  
    DispUpdate();           // update screen  
}  
DispSetStripOff();          // set off back buffers
```

The rendering procedure consists of a loop with the number of repetitions of DISP_STRIP_NUM, which represents the number of times the BackBuf fits into the FrameBuf. The DispSetStripNext() command moves the pointer to the next strip. Following are the Draw rendering functions. On each pass through the loop, the same render commands are executed, but the Draw library trims them so that they only go to the current strip. If the entire image is not redrawn, the DispLoad() function can be called before rendering to restore the contents of the BackBuf buffer from the current FrameBuf strip. After the graphic is rendered, the DispUpdate() function is called to transfer the BackBuf to the current FrameBuf strip. After the rendering loop, the DispSetStripOff() function can be called

to turn off the BackBuf strip mode. Subsequent rendering will again be done only to the FrameBuf image buffer (this is the default state when the program starts).

FRAMETYPE FrameBuf[FRAMESIZE]; display buffer

FRAMETYPE BackBuf[BACKBUFSIZE]; back buffer (only if USE_DVI >= 2)

void DviStart();

Start DVI on core 1. Called from core 0. Must be paired with DviStop(). System clock must be set to 252 MHz.

void DviStop();

Terminate DVI on core 1. Called from core 0. Must be paired with DviStart().

void DispSetStrip(int inx);

inx ... strip index 0..**VGA_STRIP_NUM**-1, or -1 = use full FrameBuffer

Set strip index of back buffer.

void DispSetStripNext();

Set next strip index of back buffer.

void DispSetStripOff()

Switch off the back buffer. Use only frame buffer to output (same as DispSetStrip(-1)).

void DispLoad();

Load back buffer from frame buffer.

void DispDirtyAll();

Set dirty all frame buffer.

void DispDirtyNone();

Set dirty none (clear after update).

void DispDirtyRect(int x, int y, int w, int h);

Update dirty area by rectangle (checks valid range).

void DispDirtyPoint(int x, int y);

Update dirty area by pixel (checks valid range).

void DispUpdate();

Save back buffer to frame buffer.

void DispAutoUpdate(u32 ms);

Auto update after delta time in [ms] of running program.

void DispUpdateAll();

Refresh update all display.

Folder Pico/PicoDVI contains samples from the PicoDVI library.



7.4. DVIVGA (HDMI with VGA)

Files: dvivga.c, dvivga.h

Config: USE_DVIVGA

The DVIVGA driver controls the digital video output on the HDMI interface simultaneously with VGA output. Only 320x240 pixel, 16-bit RGB565 color resolution is supported. The output is emulated using a 640x480 pixel video mode. The driver requires a processor overclock to 252 MHz. The output is displayed using a PIO controller and DMA transfer, from the FrameBuf image buffer. When the USE_DVIVGA parameter is set to 2 to 5, an additional buffer, BackBuf, is available in addition to the main image buffer. The additional buffer, BackBuf, is used to render the graphics, with updating of the changed area. After rendering, the contents of the changed area are transferred to the main buffer using the DispUpdate() function.

Buffer BackBuf is used in cases where the graphics need to be drawn sequentially and this would result in flickering of the image. Thus, the graphics are first drawn to the BackBuf buffer and only after complete drawing are transferred to the FrameBuf display buffer. The CPU RAM is limited (264 KB) and cannot accommodate the full size FrameBuf and BackBuf. For this reason, BackBuf can be used reduced in size and rendering is done piecemeal.

Following buffer combinations are allowed:

- USE_DVIVGA = 1 use only frame buffer, without back buffer
- USE_DVIVGA = 2 use back buffer with full size
- USE_DVIVGA = 3 use back buffer with 1/2 size
- USE_DVIVGA = 4 use back buffer with 1/4 size
- USE_DVIVGA = 5 use back buffer with 1/8 size

When drawing is done in stripes, the drawing procedure typically looks like this:

```
int strip;
for (strip = DISP_STRIP_NUM; strip > 0; strip--)
{
    DispSetStripNext();      // next strip
    // DispLoad();           // load back buffer
    ....                   // drawing functions
    DispUpdate();           // update screen
}
DispSetStripOff();          // set off back buffers
```

The rendering procedure consists of a loop with the number of repetitions of DISP_STRIP_NUM, which represents the number of times the BackBuf fits into the FrameBuf. The DispSetStripNext() command moves the pointer to the next strip. Following are the Draw rendering functions. On each pass through the loop, the same render commands are executed, but the Draw library trims them so that they only go to the current strip. If the entire image is not redrawn, the DispLoad() function can be called before

rendering to restore the contents of the BackBuf buffer from the current FrameBuf strip. After the graphic is rendered, the DispUpdate() function is called to transfer the BackBuf to the current FrameBuf strip. After the rendering loop, the DispSetStripOff() function can be called to turn off the BackBuf strip mode. Subsequent rendering will again be done only to the FrameBuf image buffer (this is the default state when the program starts).

FRAMETYPE FrameBuf[FRAMESIZE]; display buffer

FRAMETYPE BackBuf[BACKBUFSIZE]; back buffer (only if USE_DVI >= 2)

void DviCore1Exec(void (*fnc)());

Execute core 1 remote function.

Bool DviCore1Busy();

Check if core 1 is busy (executing remote function).

void DviCore1Wait();

Wait if core 1 is busy (executing remote function).

Bool DvilsVSync();

Check VSYNC.

void DviWaitVSync();

Wait for VSync scanline. In some cases, it may be sufficient to prevent the image from flickering during the drawing by using the DviWaitVSync() function before the rendering starts, without having to use BackBuf strips for drawing.

void DviStart();

Start DVI on core 1. Called from core 0. Must be paired with DviStop(). System clock must be set to 252 MHz.

void DviStop();

Terminate DVI on core 1. Called from core 0. Must be paired with DviStart().

void DispSetStrip(int inx);

inx ... strip index 0..**VGA_STRIP_NUM**-1, or -1 = use full FrameBuffer

Set strip index of back buffer.

void DispSetStripNext();

Set next strip index of back buffer.

void DispSetStripOff()

Switch off the back buffer. Use only frame buffer to output (same as DispSetStrip(-1)).

void DispLoad();

Load back buffer from frame buffer.

void DispDirtyAll();

Set dirty all frame buffer.

void DispDirtyNone();

Set dirty none (clear after update).

void DispDirtyRect(int x, int y, int w, int h);

Update dirty area by rectangle (checks valid range).

void DispDirtyPoint(int x, int y);

Update dirty area by pixel (checks valid range).

void DispUpdate();

Save back buffer to frame buffer.

void DispAutoUpdate(u32 ms);

Auto update after delta time in [ms] of running program.

void DispUpdateAll();

Refresh update all display.

7.5. DispHSTX

Files: `disphstx*.c, disphstx*.h`

Config: `USE_DISPHSTX`

DispHSTX is a driver for Pico2 RP2350 microcontroller, both in ARMv8 and RISC-V Hazard3 mode, allowing to generate DVI (HDMI) and VGA video signal, using HSTX peripheral. It supports 40 different image formats - 9 modes of pixel graphics, 2 text modes, attribute compression, RLE and HSTX compression. Functions for drawing geometric shapes, images and texts are provided for 8 basic graphic modes. The drawing library supports a back buffer - it updates the boundaries indicating the modified "dirty area". If memory is insufficient, the back buffer can be used in strip mode. Several different image formats can be combined on one screen simultaneously by dividing the screen into strips and slots. In both modes, DVI (HDMI) and VGA, color resolution up to 16 bits per pixel is possible. Although in VGA mode the output is only via a 6-bit RGB converter, higher colour resolution is achieved by the Pulse Pattern Modulation PPM, the display is visually close to the 15-bit display. The timing of the generated video signal is limited only by the overclocking capabilities of the microcontroller used. It is usually possible to achieve a resolution of up to 1440x600 pixels.

The description of the DispHSTX driver is too extensive for this manual - see the separate DispHSTX driver manual for a more detailed description.

7.6. DispHSTXMini

Files: `disphstxmini.c`, `disphstxmini.h`

Config: `USE_DISPHSTXMINI`

DispHSTXMini is a minimized version of DispHSTX driver for DVI (HDMI) and VGA display with RP2350 microcontroller. It only supports 640x480 mode timing in both VGA and DVI mode. The `USE_DISPHSTXMINI_VMODE` parameter specifies the graphics mode used, with the following values (default 1):

- 1 = 320x240/16bit, FrameBuf+DispBuf, `sys_clock`=126 MHz
- 2 = 320x240/16bit, FrameBuf+DispBuf, `sys_clock`=252 MHz
- 3 = 320x240/16bit, FrameBuf+DispBuf, `sys_clock` variable, clocked from USB 144 MHz
- 4 = 320x240/16bit, only FrameBuf, `sys_clock` variable, clocked from USB 144 MHz
- 5 = 320x144/16bit, only FrameBuf, `sys_clock` variable, clocked from USB 144 MHz
- 6 = 360x240/16bit, only FrameBuf, `sys_clock` variable, clocked from USB 144 MHz

The DispHSTXMini driver is suitable for use in cases where the system clock needs to be overclocked and where the CPU load needs to be kept to a minimum. By setting the video mode, it can be ensured that the DispHSTXMini driver is clocked by the `USB_PLL` generator (144 MHz), while the system clock `sys_clock` can be changed independently. The second advantage of the driver is that the output is done all hardware via DMA, as a whole frame at a time, without the need to prepare the contents of the graphics line. The processor is thus essentially not burdened by image processing at all. The interrupt from the DMA only comes at the end of the entire frame. The disadvantage of the driver is that the output to VGA is done with reduced colors, in RGB222 graphics mode.

8. Target Devices

Target devices sets device configuration and installs device drivers. Source files of devices are located in the `_devices` folder.

The target device can be selected during compilation as a parameter of the compilation file c.bat ('picoino10', 'picopad08', 'picopad10'). If the target device is not specified, the target device selected as default in the `_setup.bat` file (label ':default') located in the root folder of the library is used.

8.1. PicoPad

Folder: `_devices\picopad`

Config: `USE_PICOPAD, USE_PICOPAD10, USE_PICOPAD08`

The PicoPad is the main target device on which the PicoLibSDK library runs. It is a console with a built-in TFT display with 320x240 pixel resolution in 16-bit RGB565 colors. The current version of PicoPad is marked 1.0 (compilation parameter 'picopad10').

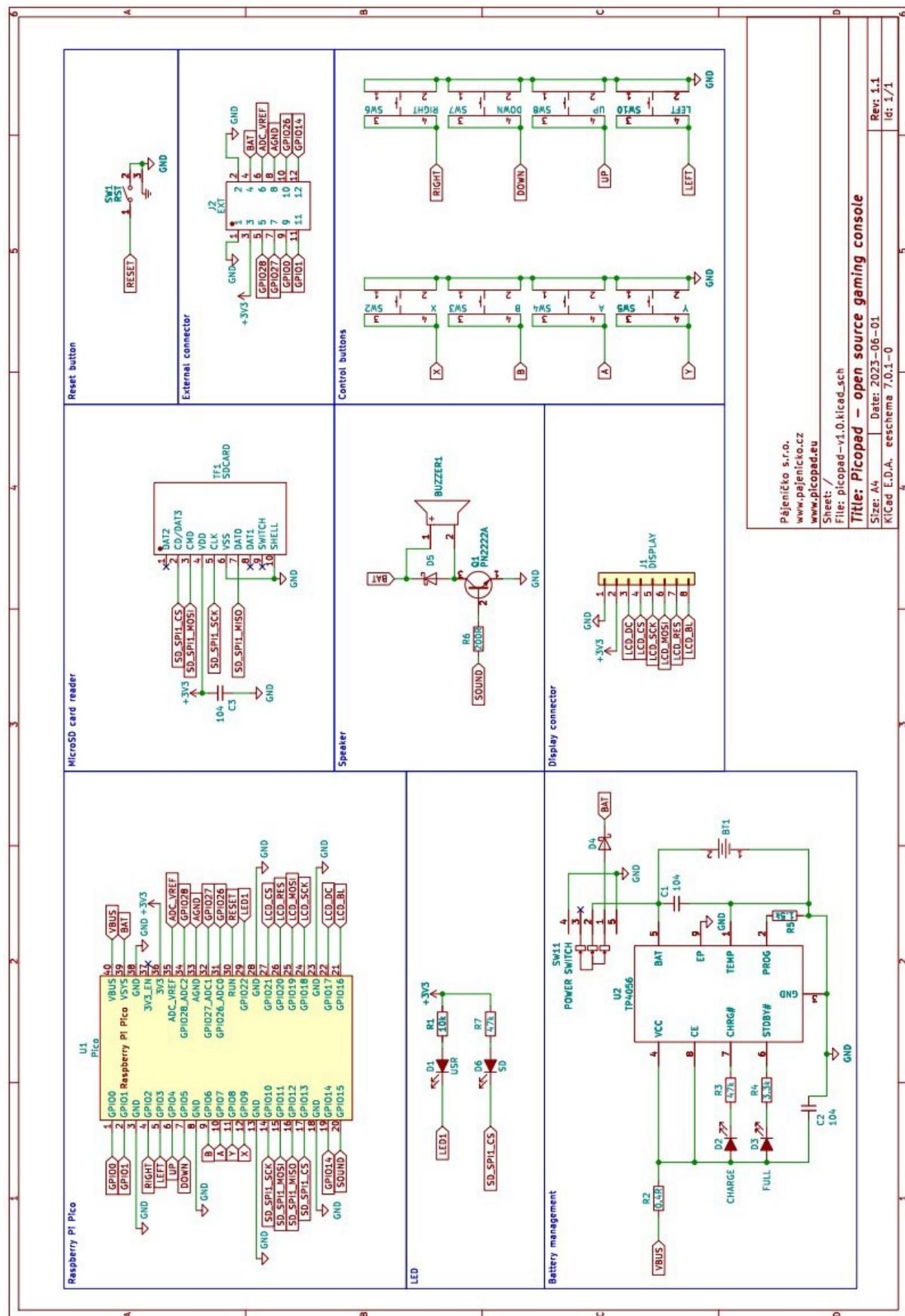
PicoPad 1.0:



The PicoPad can be controlled by 4 directional buttons and 4 control buttons (also on USB keyboard, with +5V power splitter), the usual use of which is in the meaning:

- A** confirmation of options, the main action button (USB keyboard: **Ctrl**),
- B** secondary action button (USB keyboard: **Alt**),
- X** help, scene resolution (USB keyboard: **Shift**),
- Y** interrupt function or exit program (USB keyboard: **Esc**).

Schematic diagram of the PicoPad 1.0:



Most programs and games on the PicoPad can also be controlled from the USB keyboard, which is connected to the Pico's USB connector via a power splitter that also provides an external +5V power supply.

Device

void Devicelnit();

Device init. This function is automatically called during application startup from the internal Runtimelnit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal Devicelnit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Screen shot

void ScreenShot();

Do one screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 320x240 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option.

void SmallScreenShot();

Do one small screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 160x120 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option. It is recommended to rather use the ScreenShot() function and reduce the image in the graphic editor. The resulting image will be of better quality.

LEDs

LED1	0	LED1 index (USR yellow LED on the left)
LED2	1	LED2 internal index (green, on Pico board)

void LedOn(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED ON.

void LedOff(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED OFF.

void LedFlip(u8 inx);

inx ... LED index (LED1 or LED2)

Flip LED.

void LedSet(u8 inx, u8 val);

inx ... LED index (LED1 or LED2)

val ... value 0 or 1

Set LED.

void LedInit();

Initialize LEDs. This function is automatically called during application startup from the internal DeviceInit() function.

void LedTerm();

Terminate LEDs. This function is automatically called during application termination from the internal DeviceTerm() function.

Keys

Key codes:

NOKEY	0	no key from keyboard
KEY_UP	1	up (remapped to CH_UP character)
KEY_LEFT	2	left (remapped to CH_LEFT character)
KEY_RIGHT	3	right (remapped to CH_RIGHT character)
KEY_DOWN	4	down (remapped to CH_DOWN character)
KEY_X	5	X (help, solver) (remapped to CH_TAB character)
KEY_Y	6	Y (cancel, quit) (remapped to CH_ESC character)
KEY_A	7	A (confirm, main action) (remapped to CH_CR character)

KEY_B 8 B (2nd action) (remapped to **CH_SPC** character)

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(char key);

key ... key index **KEY_***

Check if key **KEY_*** is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

u8 KeyGetRel()

Get button from keyboard buffer, including release keys. Returns **NOKEY** if no scan code. B7 = **KEY_RELEASE** = release flag.

char KeyGet();

Get button from keyboard buffer. Returns **NOKEY** if no scan code.

char KeyChar();

Get character from local keyboard. Returns **NOCHAR** if no character. Keys from the KeyGet() function are remapped to the **CH_*** codes.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(char key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

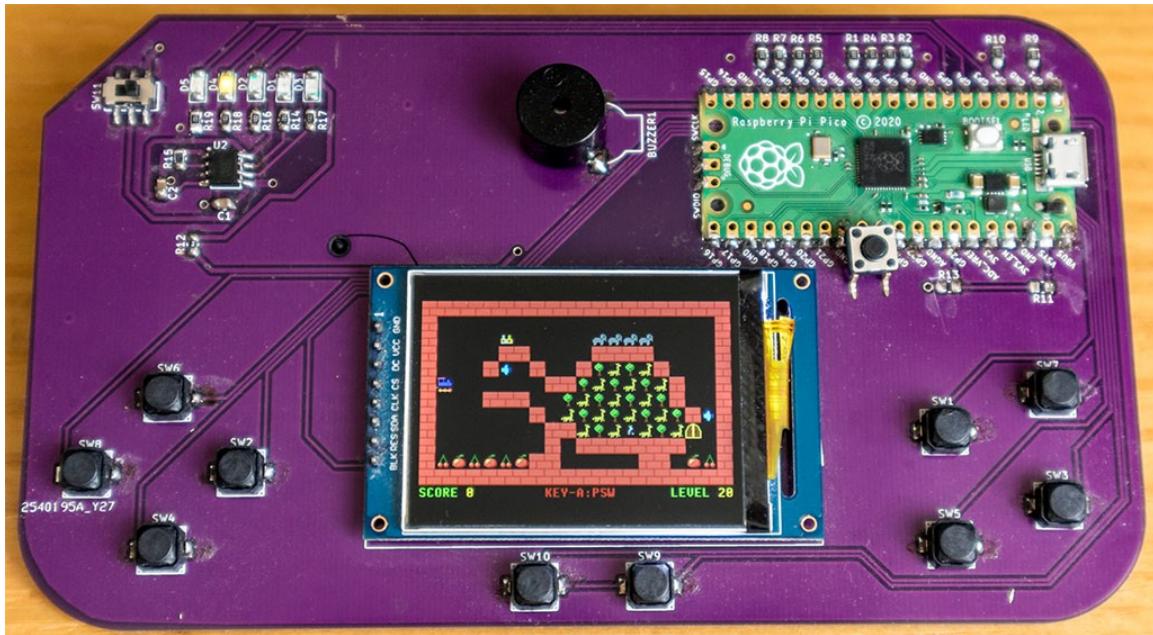
Check no pressed key.

void KeyWaitNoPressed();

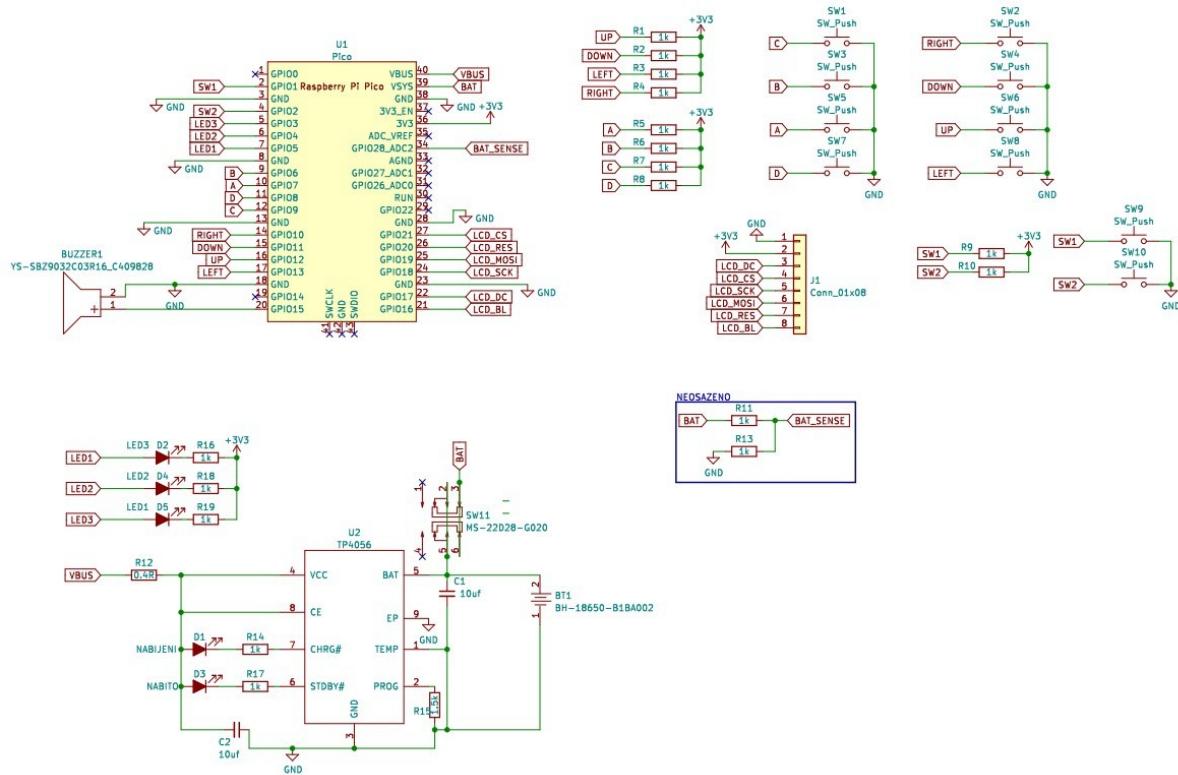
Wait for no key pressed.

The PicoPad compilation is also ready for version 0.8, which is the original PicoPad prototype with slightly different wiring. PicoPad 0.8 is only provided in the compilation as an example of an alternative hardware configuration.

PicoPad 0.8 (first prototype):



Schematic diagram of the PicoPad 0.8:



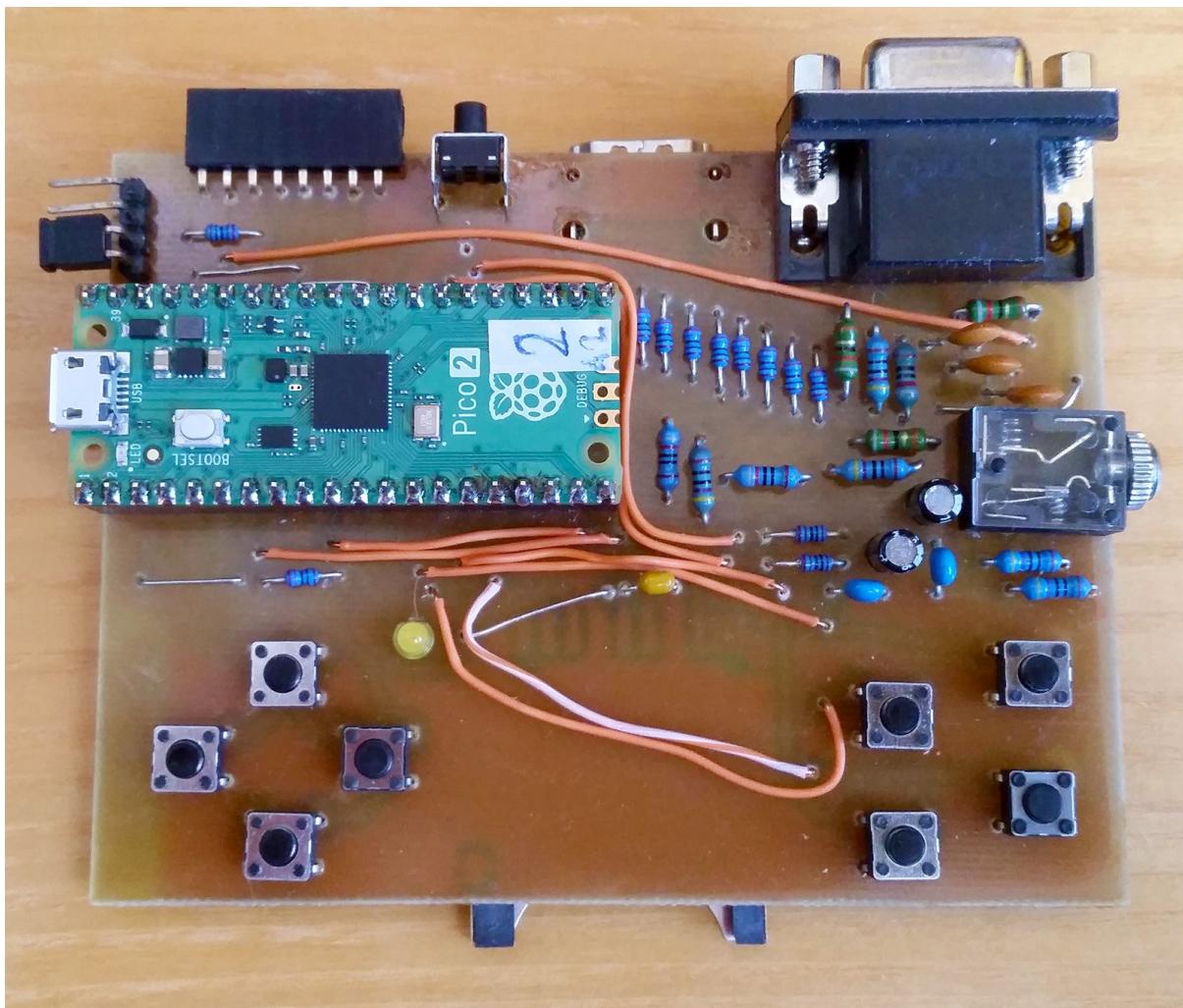
8.2. PicoPadHSTX

Folder: _devices\picopad

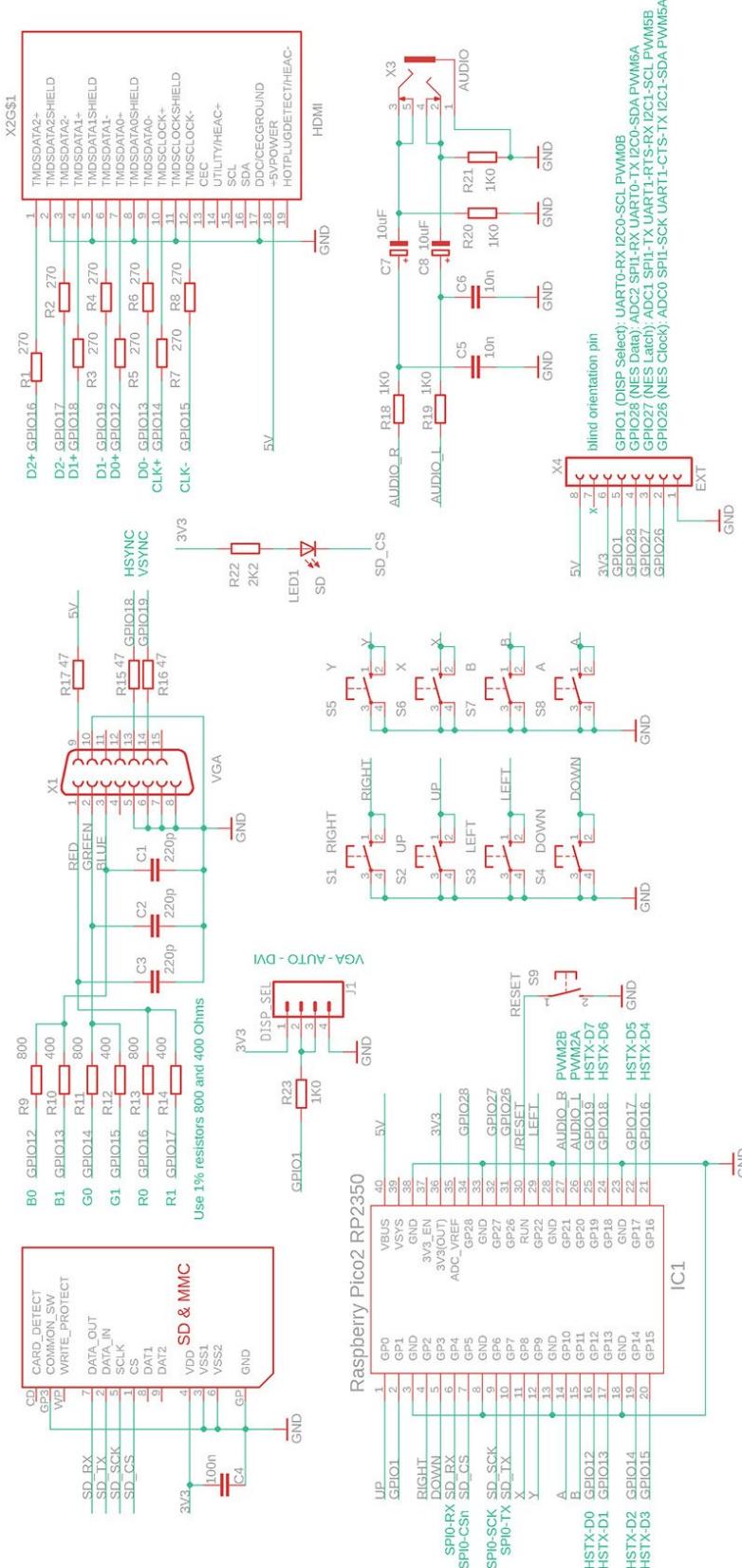
Config: USE_PICOPAD, USE_PICOPADHSTX

PicoPadHSTX is an alternative to the PicoPad console with combined DVI (HDMI) and VGA display output, with RP2350 microcontroller and with stereo sound output. It is typically used in 320x240 pixel resolution with a 16-bit RGB565 color format.

Prototype board of the PicoPadHSTX:



Schematic diagram of the PicoPadHSTX:



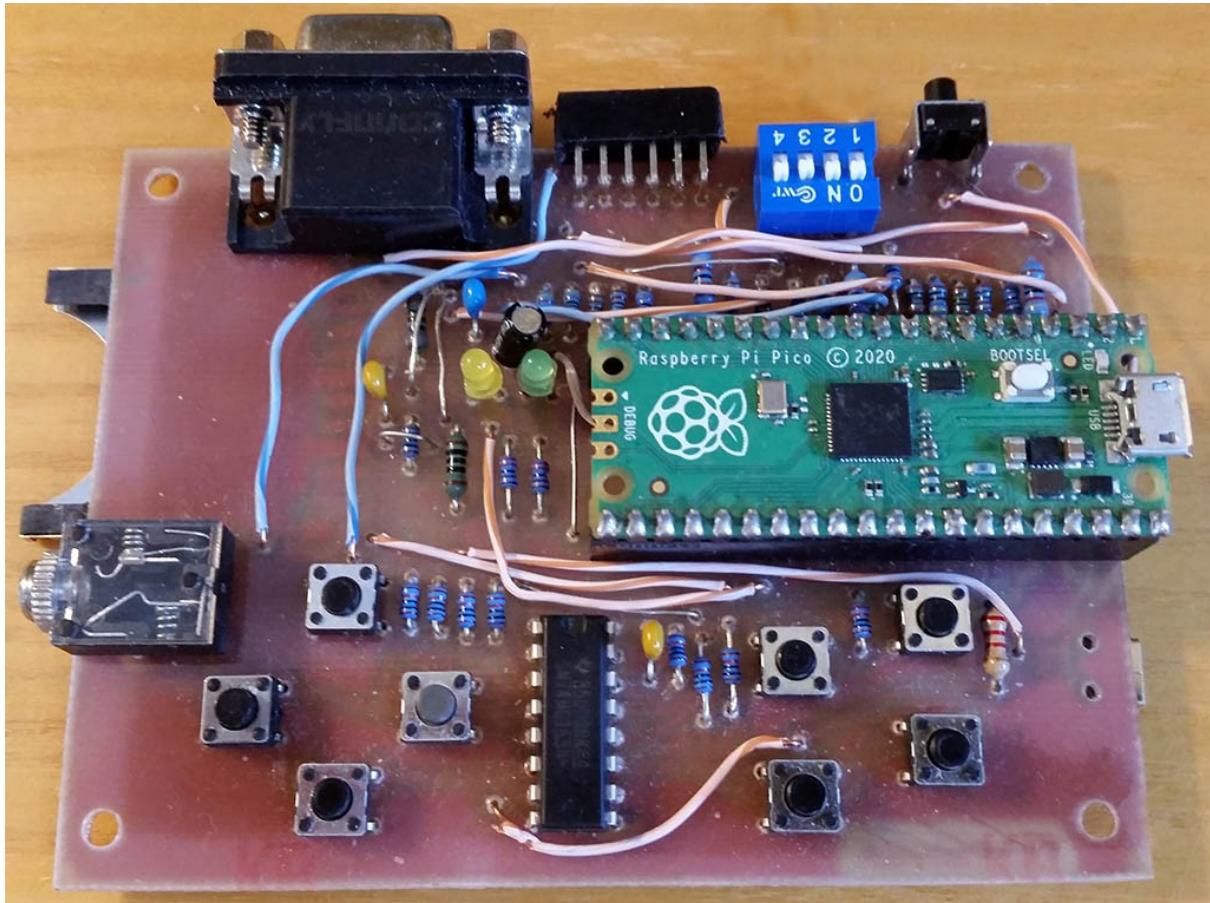
8.3. PicoPadVGA

Folder: _devices\picopad

Config: USE_PICOPAD, USE_PICOPADVGA

PicoPadVGA is an alternative to the PicoPad console with VGA display output. It is typically used in 320x240 pixel resolution with a 16-bit RGB565 color format. A 400x300 pixel resolution is also supported.

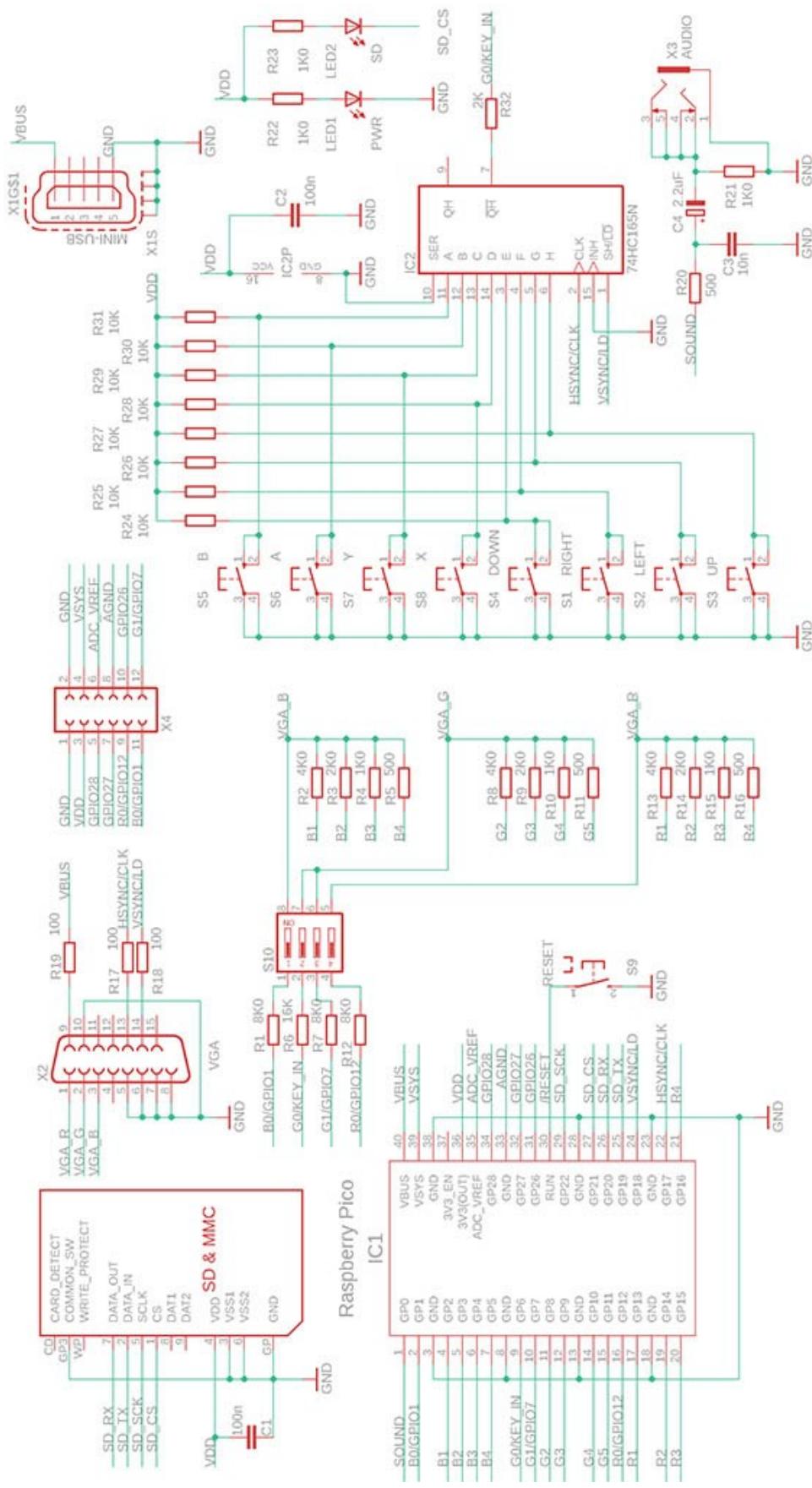
Prototype board of the PicoPadVGA:



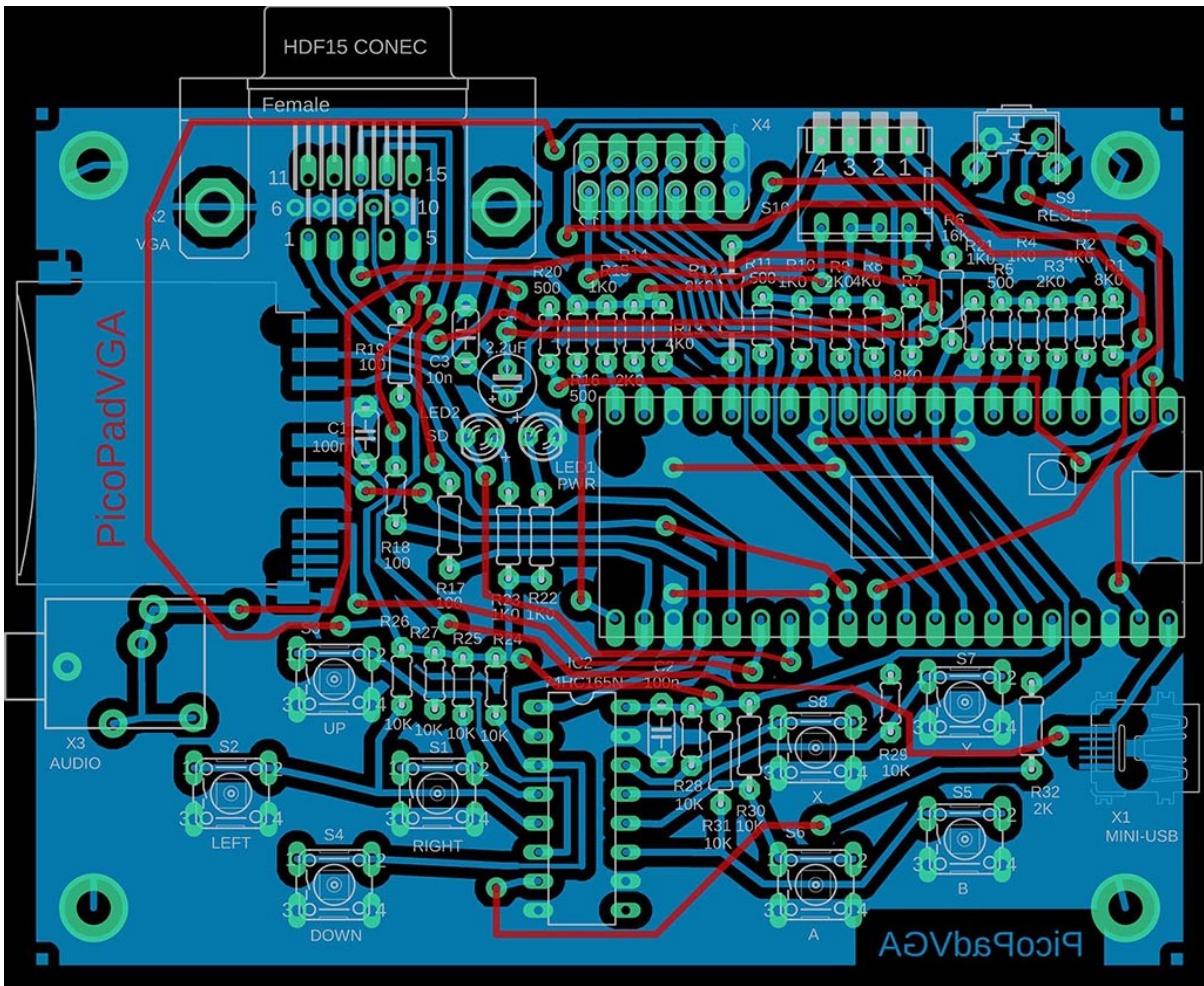
Most programs and games on the PicoPadVGA can also be controlled from the USB keyboard. PicoPadVGA must be powered from an external +5V power supply via power USB connector. Key mapping:

- A confirmation of options, the main action button (USB keyboard: **Ctrl**),
- B secondary action button (USB keyboard: **Alt**),
- X help, scene resolution (USB keyboard: **Shift**),
- Y interrupt function or exit program (USB keyboard: **Esc**).

Schematic diagram of the PicoPadVGA:



Fitting components on the PicoPadVGA PCB:



Device

void DevicelInit();

Device init. This function is automatically called during application startup from the internal RuntimeInit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Battery

void Batinit();

Init battery measurement. This function is automatically called during application startup from the internal DevicelInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Screen shot

void ScreenShot();

Do one screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 320x240 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option.

void SmallScreenShot();

Do one small screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 160x120 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option. It is recommended to rather use the ScreenShot() function and reduce the image in the graphic editor. The resulting image will be of better quality.

LEDs

LED1 0 LED1 index (SD card LED)

LED2 1 LED2 internal index (green, on Pico board)

void LedOn(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED ON.

void LedOff(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED OFF.

void LedFlip(u8 inx);

inx ... LED index (LED1 or LED2)

Flip LED.

void LedSet(u8 inx, u8 val);

inx ... LED index (LED1 or LED2)

val ... value 0 or 1

Set LED.

void LedInit();

Initialize LEDs. This function is automatically called during application startup from the internal DeviceInit() function.

void LedTerm();

Terminate LEDs. This function is automatically called during application termination from the internal DeviceTerm() function.

Keys

Key codes:

NOKEY	0	no key from keyboard
KEY_UP	1	up (remapped to CH_UP character)
KEY_LEFT	2	left (remapped to CH_LEFT character)
KEY_RIGHT	3	right (remapped to CH_RIGHT character)
KEY_DOWN	4	down (remapped to CH_DOWN character)
KEY_X	5	X (help, solver) (remapped to CH_TAB character)
KEY_Y	6	Y (cancel, quit) (remapped to CH_ESC character)
KEY_A	7	A (confirm, main action) (remapped to CH_CR character)
KEY_B	8	B (2nd action) (remapped to CH_SPC character)

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(char key);

key ... key index **KEY_***

Check if key **KEY_*** is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

char KeyGet();

Get button from keyboard buffer. Returns **NOKEY** if no scan code.

char KeyChar();

Get character from local keyboard. Returns **NOCHAR** if no character. Keys from the **KeyGet()** function are remapped to the **CH_*** codes.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(char key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

Check no pressed key.

void KeyWaitNoPressed();

Wait for no key pressed.

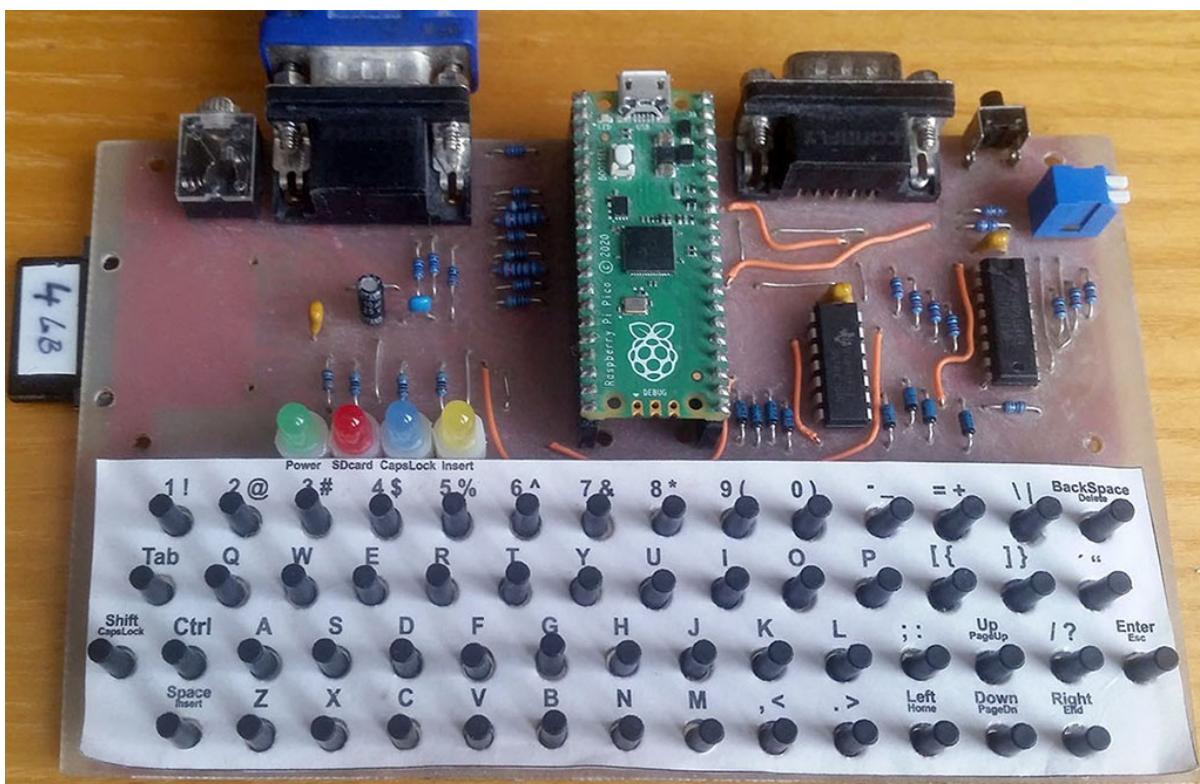
8.4. Picoino

Folder: _devices\picoino

Config: USE_PICOINO, USE_PICOINO10

The Picoino microcomputer is the predecessor of the PicoPad console. It uses output to an external VGA monitor in QVGA mode with a resolution of 320x240 pixels. Pixels are in 8-bit RGB332 format (3 bits red component, 3 bits green component and 2 bits blue component). The Picoino is equipped with an internal keyboard of 56 micro-switches, an SD card and a Joystick connector for connecting external devices. Compilation of Picoino sample programs are done using compilation parameter 'picoino10'.

Picoino 1.0:



Device

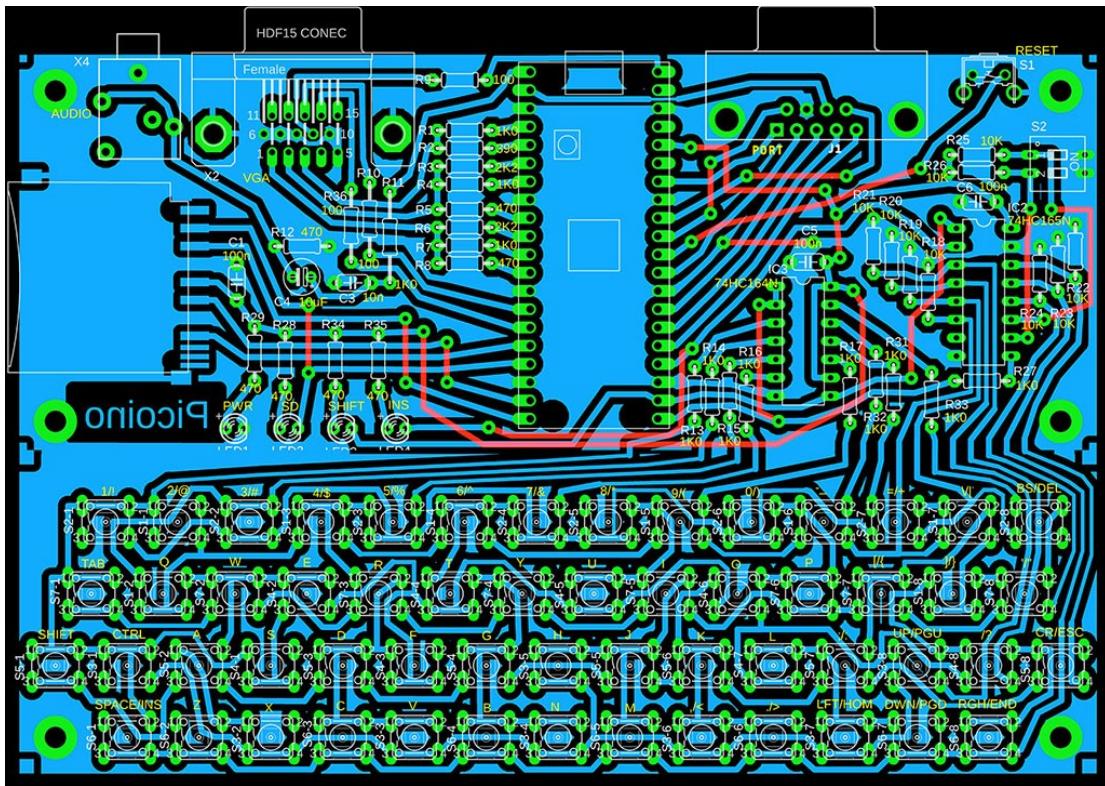
void DeviceInit();

Device init. This function is automatically called during application startup from the internal RuntimeInit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Fitting components on the Picoino PCB:



Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal DeviceInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

float GetBat();

Get battery voltage in V.

int GetBatInt();

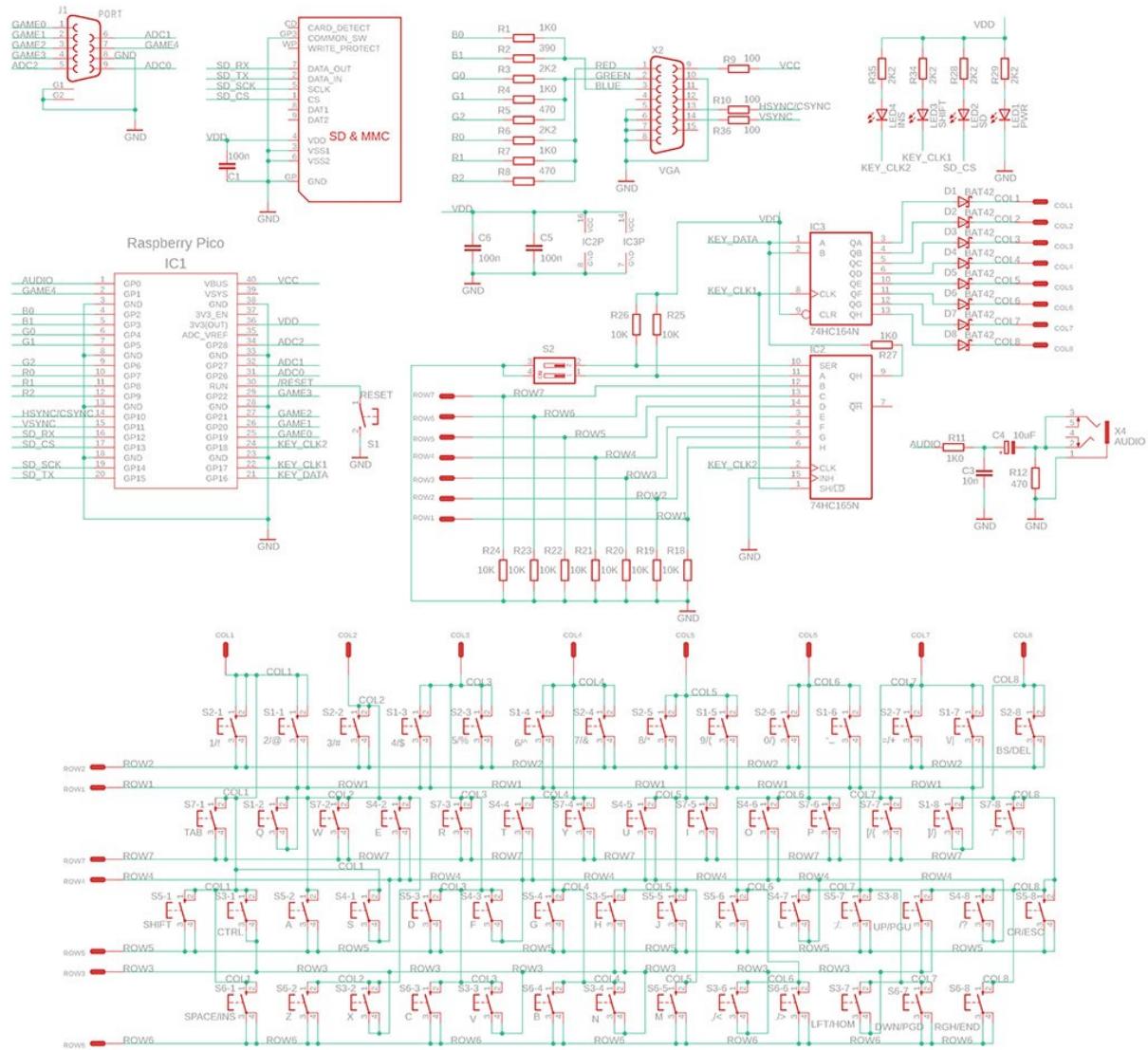
Get battery voltage, integer in mV.

Screen shot

void ScreenShot();

Do one screen shot. Function captures screen in BMP RGB32 8-bit format (resolution 320x240 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option.

Schematic diagram of the Picino 1.0:



LEDs

LED1 0 LED1 internal index (green, on Pico board)

void LedOn(u8 inx);

inx ... LED index (LED1)

Set LED ON.

void LedOff(u8 inx);

inx ... LED index (LED1)

Set LED OFF.

void LedFlip(u8 inx);

inx ... LED index (LED1)

Flip LED.

void LedSet(u8 inx, u8 val);

inx ... LED index (LED1)

val ... value 0 or 1

Set LED.

Keys

Virtual key codes:

VK_2	1	// 2/@
VK_1	2	// 1/!
VK_CTRL	3	// Ctrl
VK_S	4	// S
VK_SHIFT	5	// Shift
VK_SPACE	6	// Space/Ins
VK_TAB	7	// Tab
VK_Q	8	// Q
VK_3	9	// 3/#
VK_X	10	// X
VK_E	11	// E
VK_A	12	// A
VK_Z	13	// Z
VK_W	14	// W
VK_4	15	// 4/\$
VK_5	16	// 5/%
VK_V	17	// V
VK_F	18	// F
VK_D	19	// D
VK_C	20	// C
VK_R	21	// R
VK_6	22	// 6/^
VK_7	23	// 7/&
VK_N	24	// N
VK_T	25	// T
VK_G	26	// G
VK_B	27	// B
VK_Y	28	// Y
VK_9	29	// 9/(

VK_8	30	// 8/*
VK_H	31	// H
VK_U	32	// U
VK_J	33	// J
VK_M	34	// M
VK_I	35	// I
VK_RSLASH	36	// \'
VK_EQU	37	// =/+
VK_LEFT	38	// Left/Home
VK_L	39	// L
VK_SEMI	40	// ;:
VK_DOWN	41	// Down/PgDn
VK_LPAR	42	// [{
VK_MINUS	43	// -_-
VK_0	44	// 0/)
VK_COMMA	45	// ,<
VK_O	46	// O
VK_K	47	// K
VK_DOT	48	// ./>
VK_P	49	// P
VK_RPAR	50	//]/}
VK_BS	51	// Backspace/Del
VK_UP	52	// Up/PgUp
VK_SLASH	53	// /?
VK_CR	54	// CR/Esc
VK_RIGHT	55	// Right/End
VK_QUOT	56	// '"

Remapping PicoPad keys to Picoino virtual keys:

KEY_UP	VK_UP	// up
KEY_LEFT	VK_LEFT	// left
KEY_RIGHT	VK_RIGHT	// right
KEY_DOWN	VK_DOWN	// down
KEY_X	VK_CTRL	// X
KEY_Y	VK_BS	// Y
KEY_A	VK_SPACE	// A
KEY_B	VK_Z	// B

```
Bool KeyCapsLock; // flag - caps lock is ON  
Bool KeyInsert; // flag - Insert is ON
```

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(char key);

key ... key index **VK_*** or **KEY_***

Check if key **VK_*** or **KEY_*** is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

char KeyGet();

Get button from keyboard buffer. Returns **NOKEY** if no scan code.

char KeyChar();

Get character from local keyboard. Returns **NOCHAR** if no character. Keys from the KeyGet() function are remapped to the **CH_*** and ASCII codes.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(char key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

Check no pressed key.

void KeyWaitNoPressed();

Wait for no key pressed.

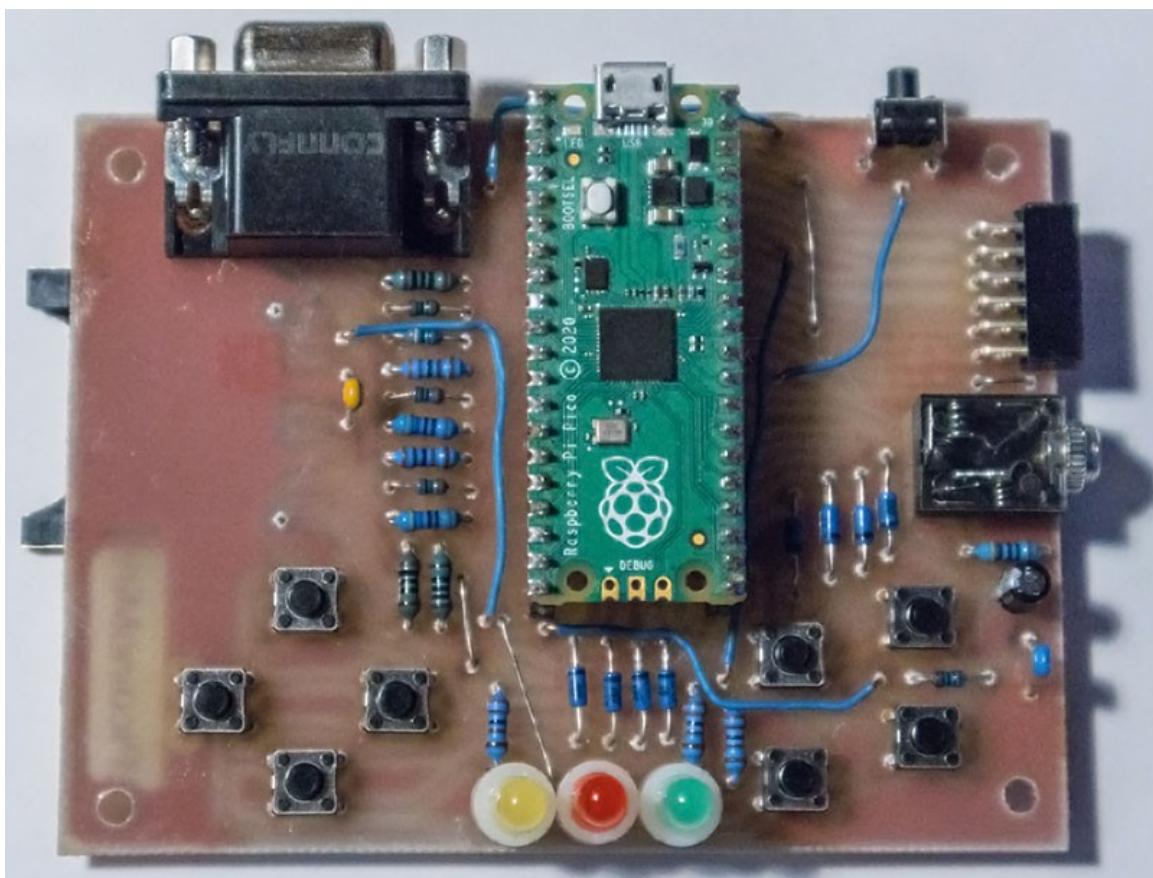
8.5. PicoinoMini

Folder: _devices\picoinomini

Config: USE_PICOINO, USE_PICOINOMINI

PicoinoMini is a lighter version of Picoino. It also uses VGA monitor output in QVGA mode with 320x240 pixel/8-bit RGB332 color resolution, has an SD card and PWM audio output. Instead of an internal keyboard, it has a keypad with 8 keys, similar to the PicoPad. Compilation of PicoinoMini sample programs are done using compilation parameter 'picoinomini'.

PicoinoMini:



Device

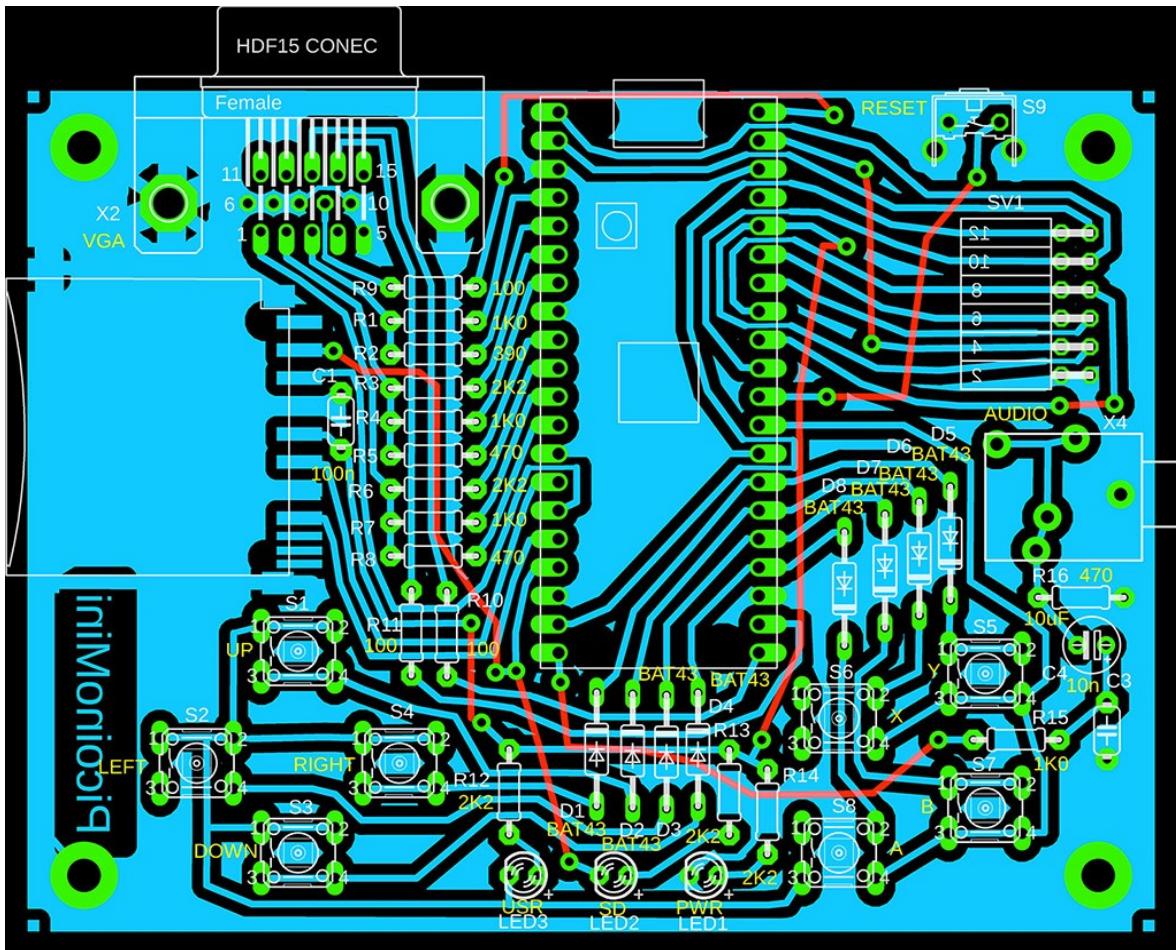
void DeviceInit();

Device init. This function is automatically called during application startup from the internal RuntimeInit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Fitting components on the Picoino PCB:



Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal DeviceInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

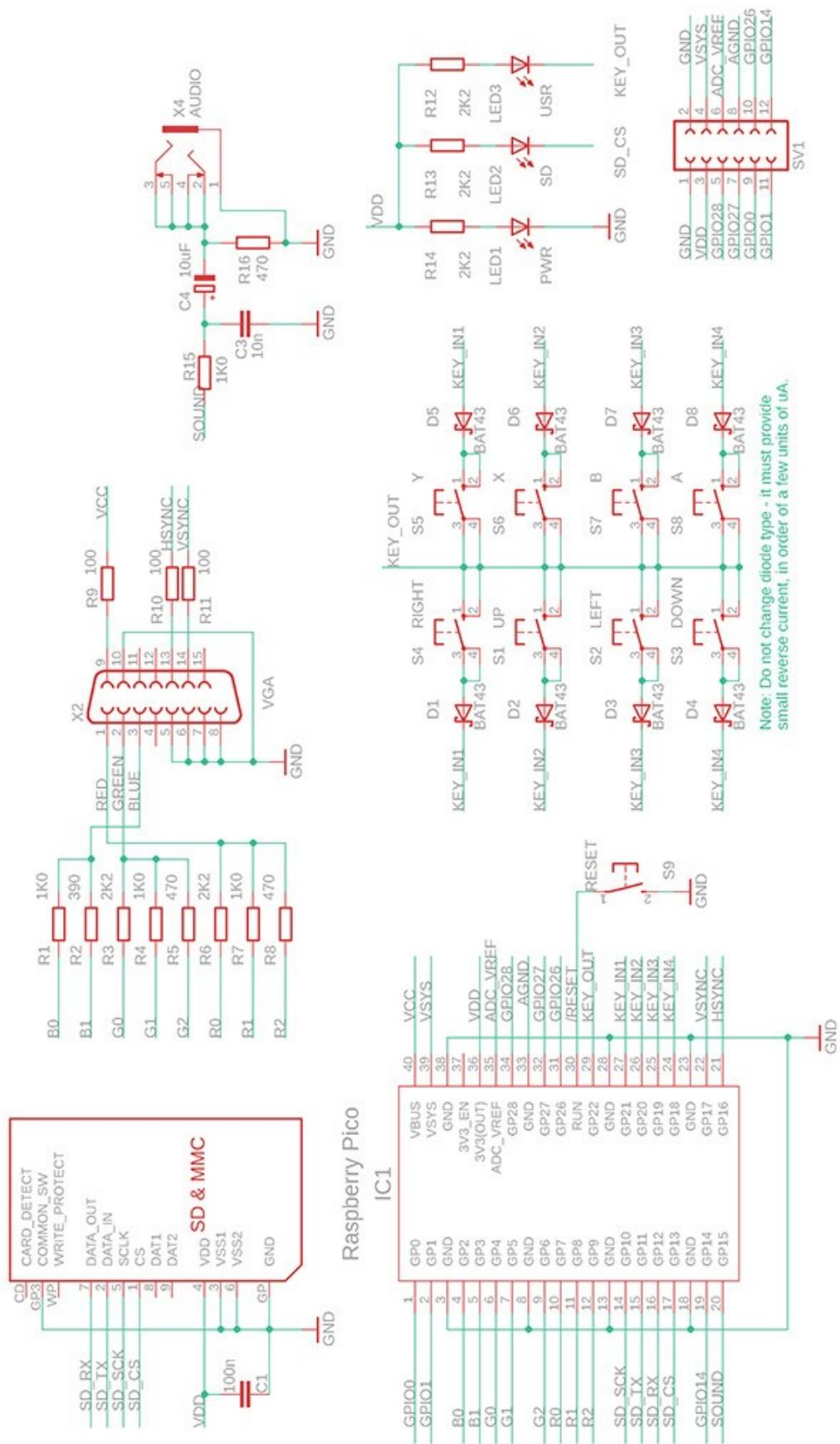
float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Schematic diagram of the PicoinoMini:



Screen shot

void ScreenShot();

Do one screen shot. Function captures screen in BMP RGB32 8-bit format (resolution 320x240 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option.

LEDs

LED1 0 LED1 index (yellow, user LED)

LED2 1 LED2 internal index (green, on Pico board)

void LedOn(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED ON.

void LedOff(u8 inx);

inx ... LED index (LED1 or LED2)

Set LED OFF.

void LedFlip(u8 inx);

inx ... LED index (LED1 or LED2)

Flip LED.

void LedSet(u8 inx, u8 val);

inx ... LED index (LED1 or LED2)

val ... value 0 or 1

Set LED.

PicoinoMini keys:

KEY_UP // up

KEY_LEFT / left

KEY_RIGHT // right

KEY_DOWN // down

KEY_X // X

KEY_Y // Y

KEY_A // A

KEY_B // B

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(char key);

key ... key index **KEY_***

Check if key **KEY_*** is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

char KeyGet();

Get button from keyboard buffer. Returns **NOKEY** if no scan code.

char KeyChar();

Get character from local keyboard. Returns **NOCHAR** if no character. Keys from the KeyGet() function are remapped to the **CH_*** and ASCII codes.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(char key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

Check no pressed key.

void KeyWaitNoPressed();

Wait for no key pressed.

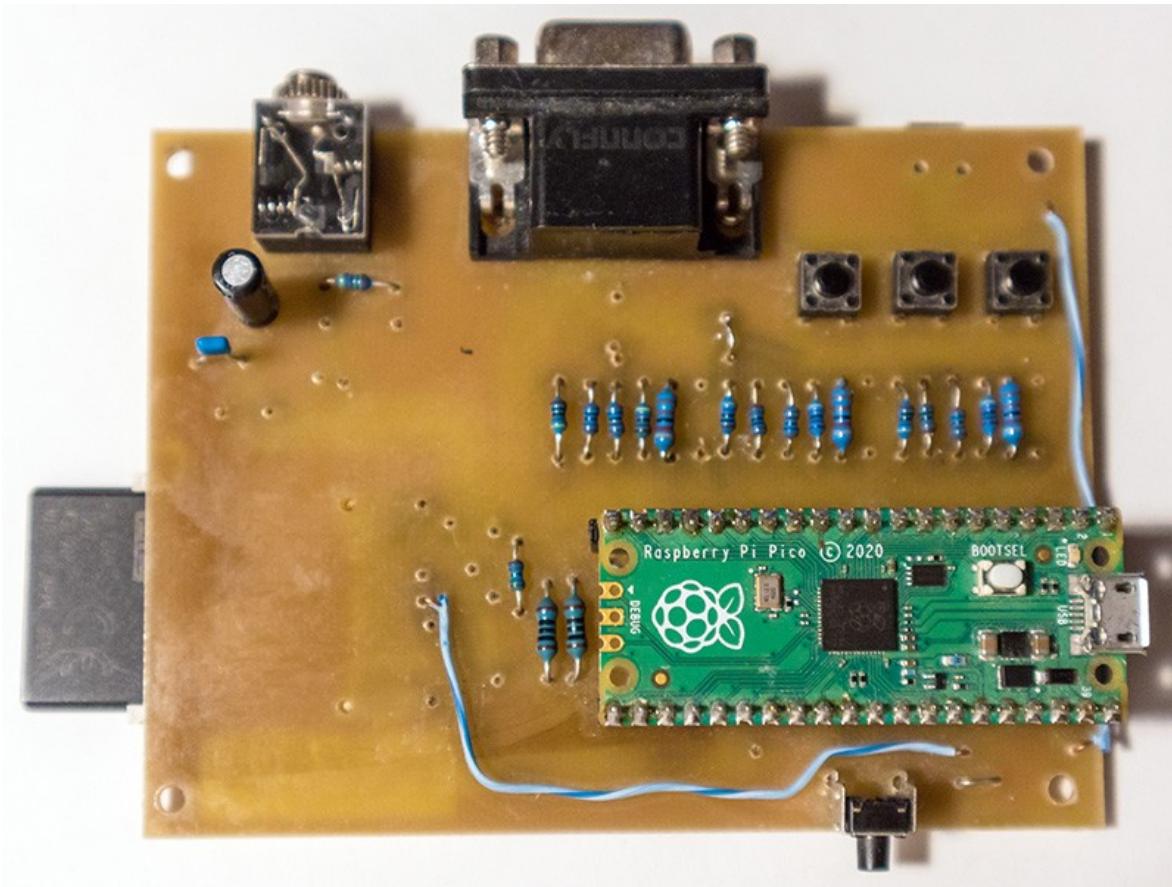
8.6. DemoVGA

Folder: _devices\picoinomini

Config: USE_PICOINO, USE_PICOINOMINI

DemoVGA is a demo board with output to a VGA monitor in 320x240 pixel resolution, 16 bits, RGB565 color format. Includes SD card, additional power USB connector, headphone output and 3 buttons. Compilation of DemoVGA sample programs are done using compilation parameter 'demovga'. If the program requires more buttons (typically games), an external USB keyboard connected to the USB connector of the Pico module can be used.

DemoVGA:



Device

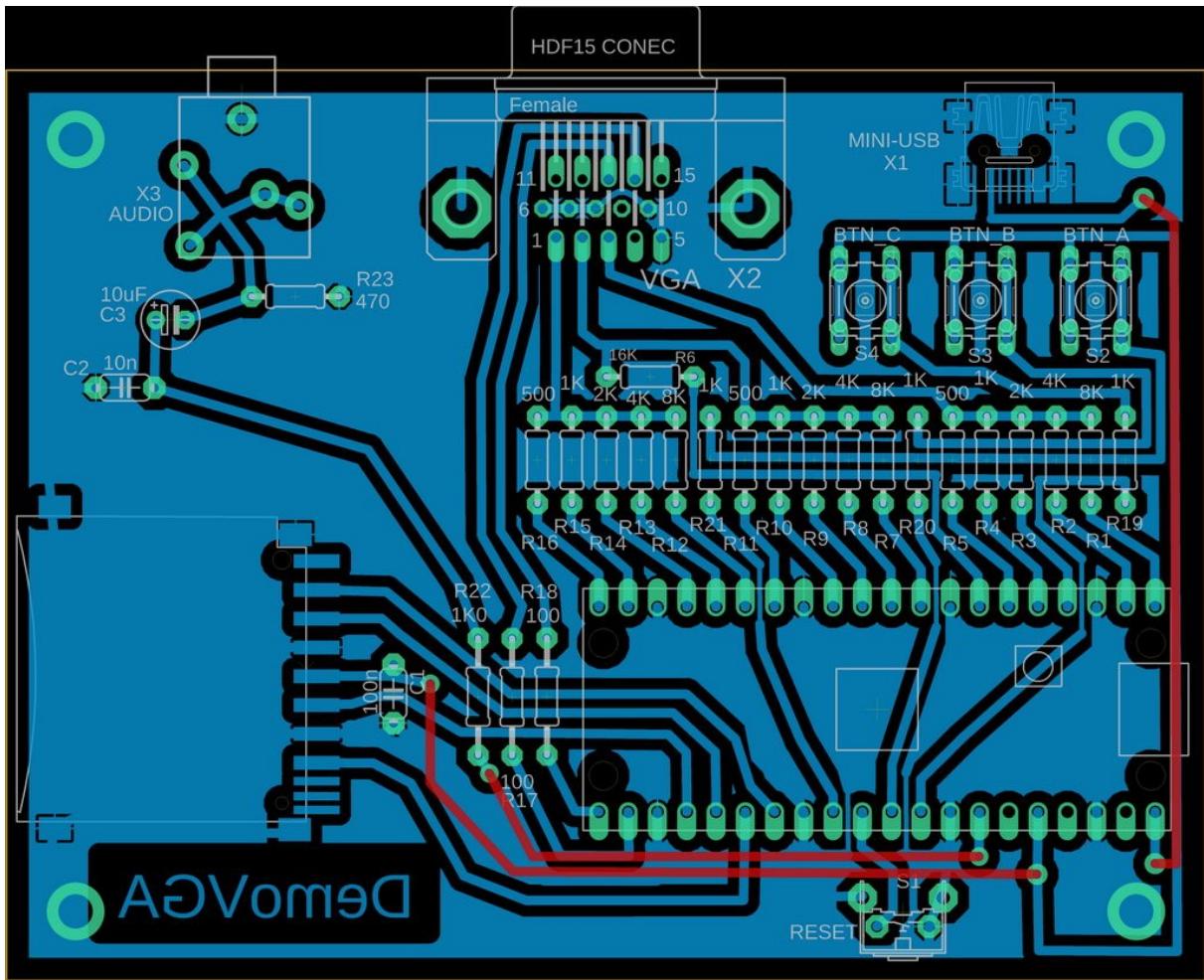
void DeviceInit();

Device init. This function is automatically called during application startup from the internal RuntimeInit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Fitting components on the DemoVGA PCB:



Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal DeviceInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

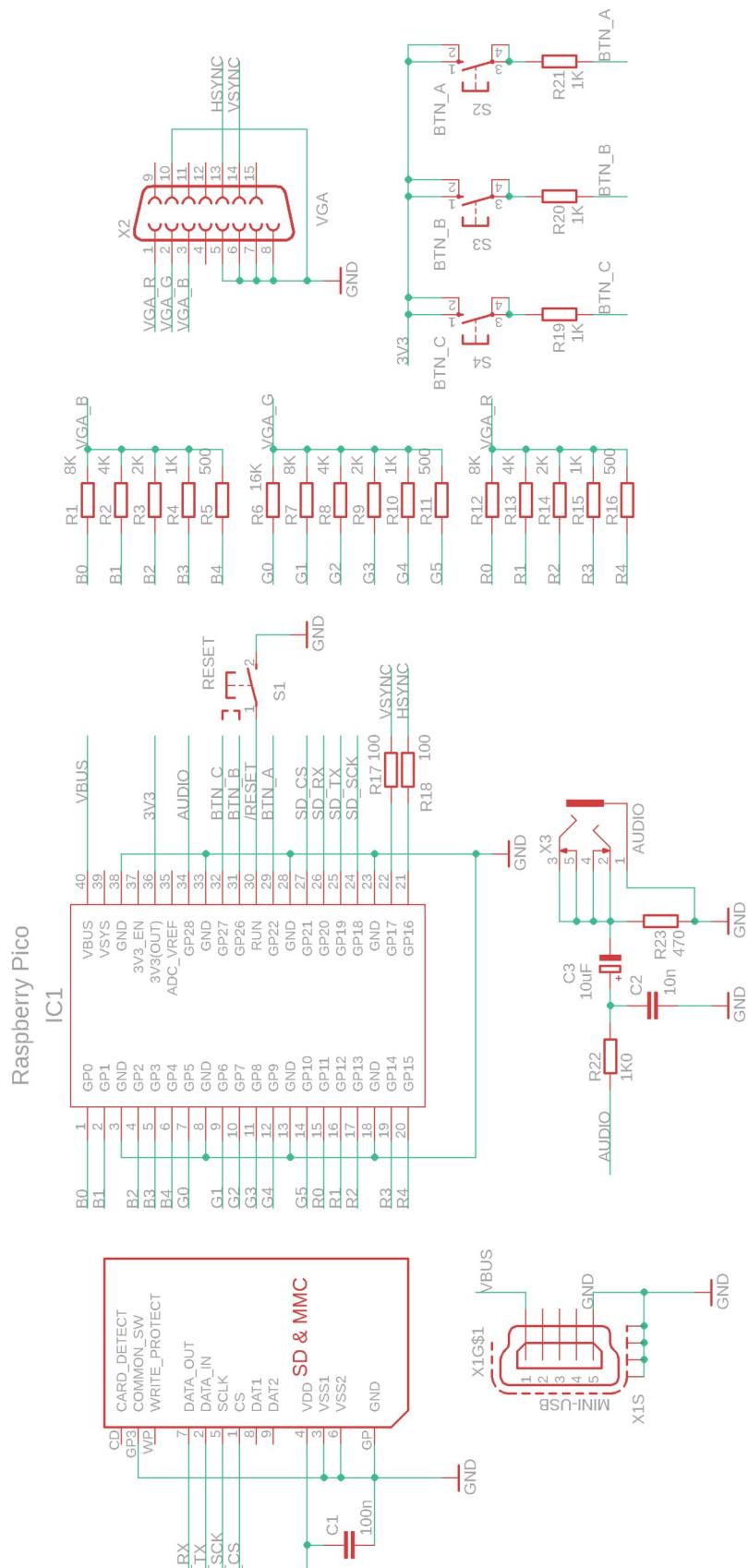
float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Schematic diagram of the DemoVGA:



Screen shot

void ScreenShot();

Do one screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 320x240 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option.

void SmallScreenShot();

Do one small screen shot. Function captures screen in BMP RGB565 16-bit format (resolution 160x120 pixels) and generates file SCRxxxx.BMP in root of SD card. Requires to be compiled with the **USE_SCREENSHOT** configuration option. It is recommended to rather use the ScreenShot() function and reduce the image in the graphic editor. The resulting image will be of better quality.

LEDs

LED1 0 LED1 index (green, on Pico board)

void LedOn(u8 inx);

inx ... LED index (LED1)

Set LED ON.

void LedOff(u8 inx);

inx ... LED index (LED1)

Set LED OFF.

void LedFlip(u8 inx);

inx ... LED index (LED1)

Flip LED.

void LedSet(u8 inx, u8 val);

inx ... LED index (LED1)

val ... value 0 or 1

Set LED.

DemoVGA keys:

BTN_A // A

BTN_B // B

BTN_C // C

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(u8 key);

key ... key index **BTN_***

Check if key **BTN_*** is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

u8 KeyGet();

Get button from keyboard buffer. Returns **NOKEY** if no scan code.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(u8 key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

Check no pressed key.

void KeyWaitNoPressed();

Wait for no key pressed.

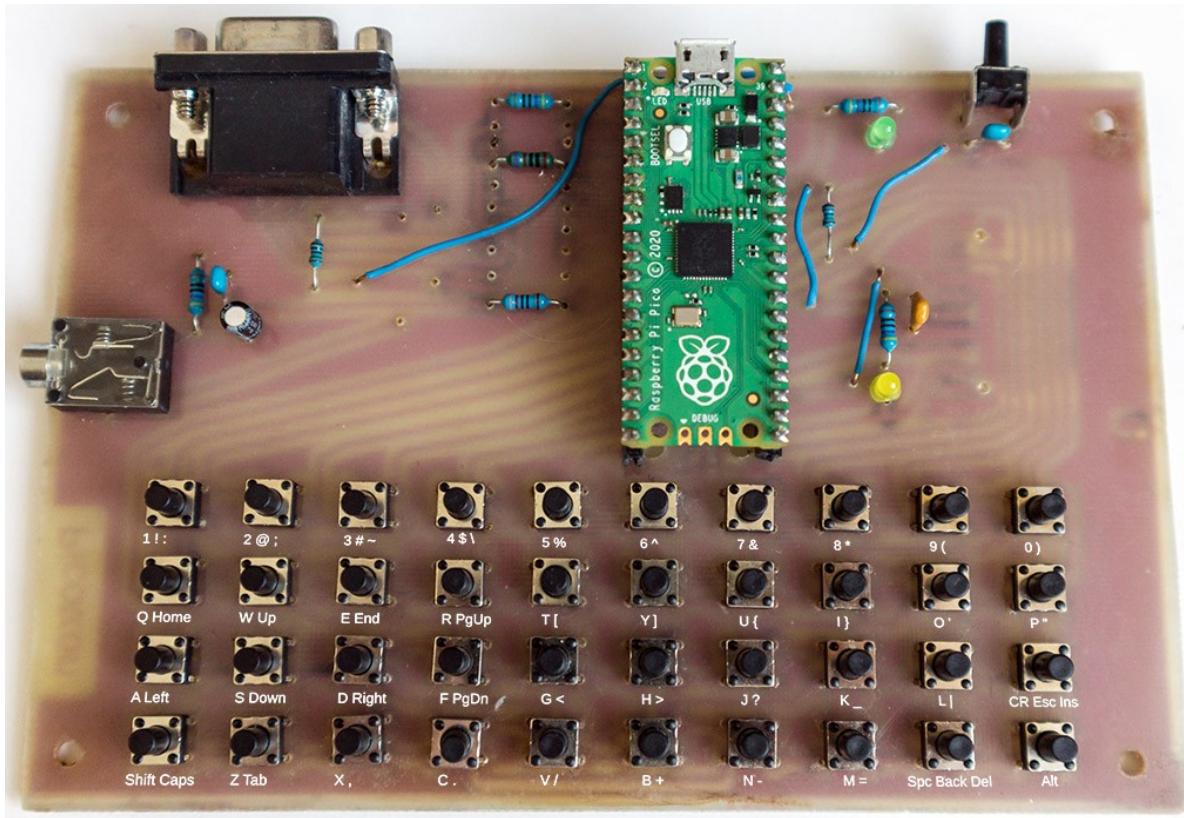
8.7. Picotron

Folder: _devices\picotron

Config: USE_PICOTRON

Picotron is simple computer with output to VGA monitor with resolution up to 800x600 pixels, with 4-bit color output (16 colors) in YRGB1111 format. Includes SD card, 40-key keyboard and headphone output. Compilation of Picotron sample programs are done using compilation parameter 'picotron'. This board is not suitable for graphics applications. Its use is mainly in industrial areas where higher resolution displays are required.

Picotron:



Device

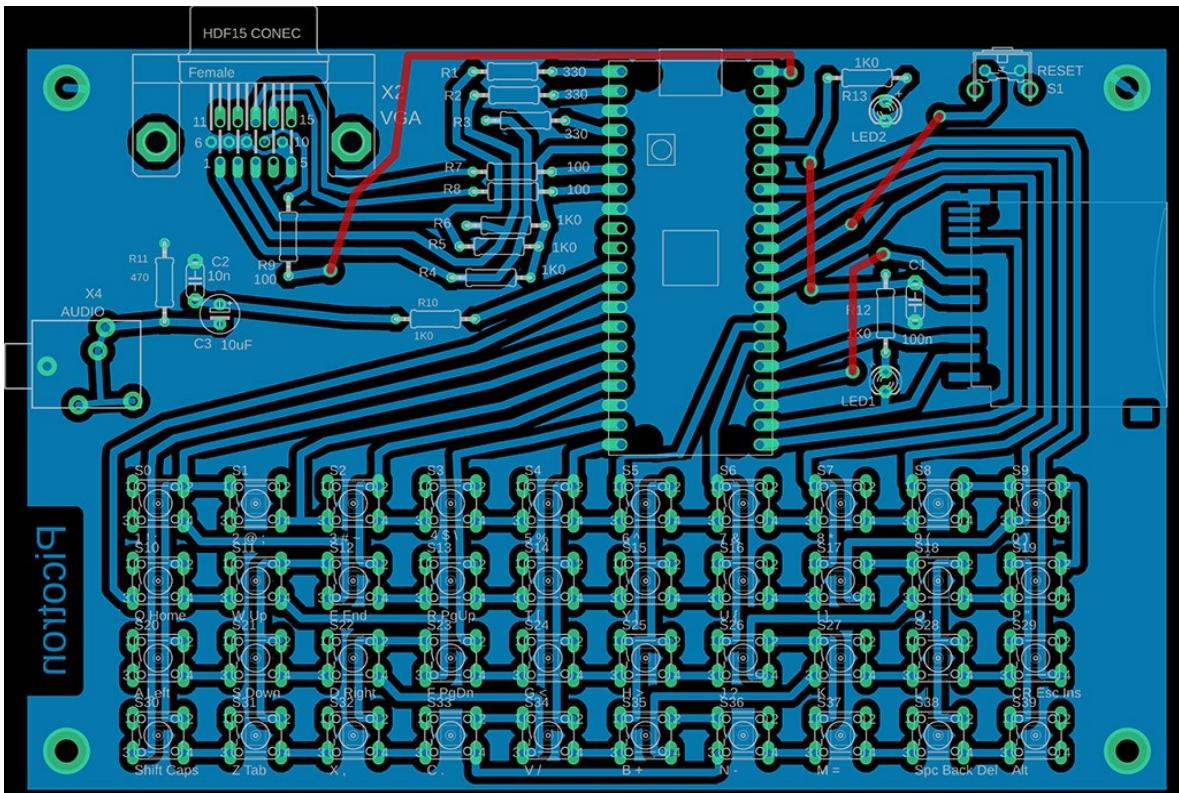
void DeviceInit();

Device init. This function is automatically called during application startup from the internal RuntimeInit() function.

void DeviceTerm();

Device terminate. This function is automatically called during application termination from the internal RuntimeTerm() function.

Fitting components on the Picotron PCB:



Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal DeviceInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

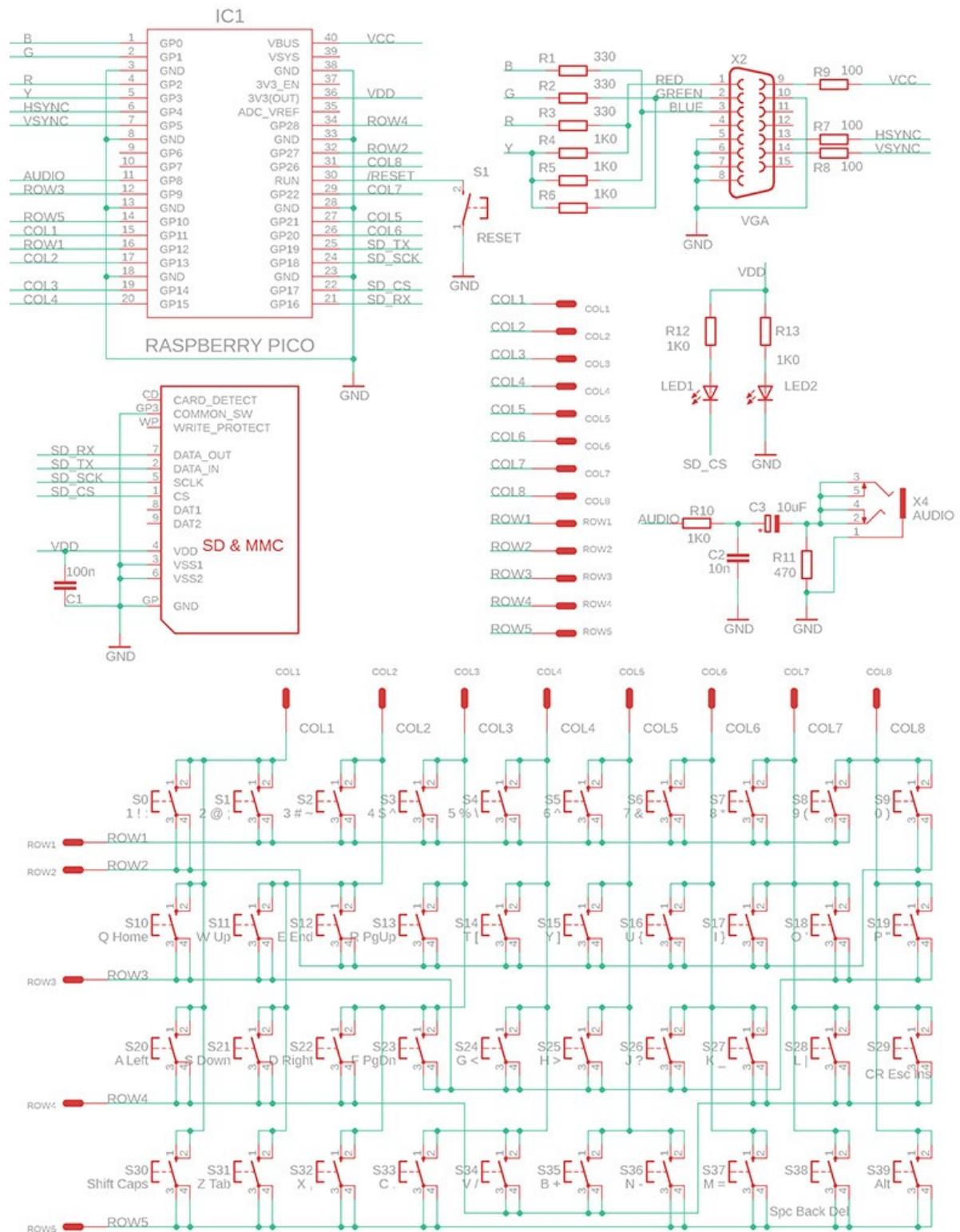
float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Schematic diagram of the Picotron:



LEDs

LED1 0 LED1 index (green, on Pico board)

void LedOn(u8 inx);
inx ... LED index (LED1)

Set LED ON.

void LedOff(u8 inx);
inx ... LED index (LED1)

Set LED OFF.

void LedFlip(u8 inx);
inx ... LED index (LED1)

Flip LED.

void LedSet(u8 inx, u8 val);
inx ... LED index (LED1)
val ... value 0 or 1

Set LED.

Picotron keys:

#define VK_2	1	// 2 @ ;
#define VK_3	2	// 3 # ~
#define VK_4	3	// 4 \$ ^
#define VK_5	4	// 5 % \'
#define VK_6	5	// 6 ^
#define VK_7	6	// 7 &
#define VK_8	7	// 8 *
#define VK_9	8	// 9 (
#define VK_1	9	// 1 ! :
#define VK_E	10	// E End
#define VK_R	11	// R PgUp
#define VK_T	12	// T [
#define VK_Y	13	// Y]
#define VK_U	14	// U {
#define VK_I	15	// I }
#define VK_O	16	// O)
#define VK_Q	17	// Q Home

```

#define VK_W      18    // W Up
#define VK_F      19    // F PgDn
#define VK_G      20    // G <
#define VK_H      21    // H >
#define VK_J      22    // J ?
#define VK_O      23    // O '
#define VK_P      24    // P "
#define VK_A      25    // A Left
#define VK_S      26    // S Down
#define VK_D      27    // D Right
#define VK_V      28    // V /
#define VK_B      29    // B +
#define VK_K      30    // K _
#define VK_L      31    // L |
#define VK_CR     32    // CR Esc Ins
#define VK_SHIFT   33    // Shift Caps
#define VK_Z      34    // Z Tab
#define VK_X      35    // X ,
#define VK_C      36    // C .
#define VK_N      37    // N -
#define VK_M      38    // M =
#define VK_SPACE   39    // Spc Back Del
#define VK_ALT     40    // Alt (alternate key code)

```

// remapped codes

```

#define KEY_UP    VK_W    // up
#define KEY_LEFT   VK_A    // left
#define KEY_RIGHT  VK_D    // right
#define KEY_DOWN   VK_S    // down
#define KEY_X     VK_L    // X
#define KEY_Y     VK_CR   // Y
#define KEY_A     VK_SPACE// A
#define KEY_B     VK_M    // B

```

void KeyInit();

Initialize keys. Key handling requires timed operation from SysTick interrupt. This function is automatically called during application startup from the internal DeviceInit() function.

void KeyTerm();

Terminate keys. This function is automatically called during application termination from the internal DeviceTerm() function.

Bool KeyPressed(u8 key);

key ... key index KEY_*

Check if key KEY_* is currently pressed.

void KeyScan();

Scan keyboard. Called from SysTick. If SysTick is not configured, KeyScan() function must be called from the application repeatedly.

u8 KeyGet();

Get button from keyboard buffer. Returns NOKEY if no scan code.

void KeyFlush();

Flush keyboard buffer.

void KeyRet(u8 key);

Return key to keyboard buffer (can hold only 1 key).

Bool KeyNoPressed();

Check no pressed key.

void KeyWaitNoPressed();

Wait for no key pressed.

8.8. Raspberry Pico

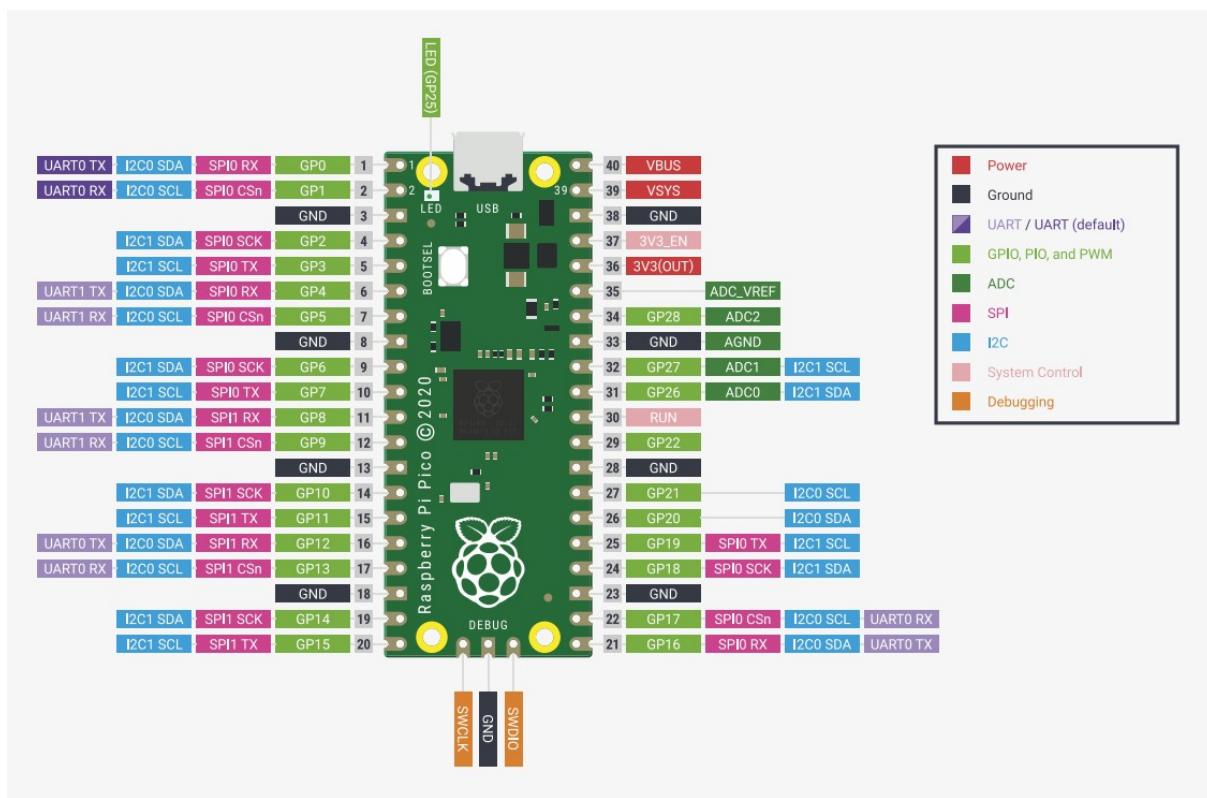
Folder: _devices\pico

Config: USE_PICO

The PicoLibSDK library is also ready for the basic Raspberry Pico module itself, without additional hardware. This configuration is especially useful for basic testing of the Pico module and for basic tutorial lessons.

Most of the sample applications for Raspberry Pico require a console output to a COM terminal. The output can be either to a UART serial port or to a virtual USB COM port.

In the case of the UART serial port, connect a UART-to-USB converter (e.g., PL2303TA adapter) to pins GPIO0 (TX output from the Pico) and GPIO1 (RX input to the Pico). Set COM port to 115200 Baud, 8 bits, no parity, 1 stop bit, no flow control.



Battery

void BatInit();

Init battery measurement. This function is automatically called during application startup from the internal DeviceInit() function.

void BatTerm();

Terminate battery measurement. This function is automatically called during application termination from the internal DeviceTerm() function.

float GetBat();

Get battery voltage in V.

int GetBatInt();

Get battery voltage, integer in mV.

Sample programs for Raspberry Pico

Led - blinking LED on Pico board.

Hello - print "Hello World!" on the console.

Pi - calculating number Pi to 1180 digits.

Temp - measure CPU temperature.

Vsys - measure Vsys supply voltage.

9. Sample Applications

Almost all sample applications and games are available for PicoPad, PicoPadVGA, Picoino and DemoVGA. The main difference is that the output from Picoino is only in 8-bit RGB332 color, while the output from PicoPad and DemoVGA is in 16-bit RGB565 color. Some programs (mainly games) for DemoVGA uses external USB keyboard.

The control is marked according to the PicoPad. In Picoino the keys are mapped in the following meaning: the **Spc** (spacebar) is as 'A' main action key, **Z** is as 'B' secondary action key, **Ctrl** is in the meaning of 'X' help and solution, **BackSpace** is instead of the 'Y' key, i.e. interrupt operation and exit the program.

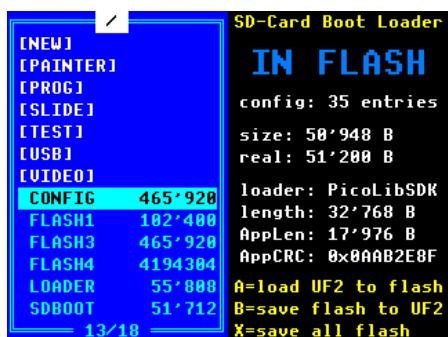
Boot3 Loader



Boot3 Loader - loader to run programs from the SD card. The 'X' key adjusts the volume and brightness of the display. The Boot3 loader can be changed either via the USB interface (BOOTSEL) or by loading from the SD card using the SDBOOT program.



Config - Setup device configuration: volume, backlight, ADC reference, battery, temperature, crystal.



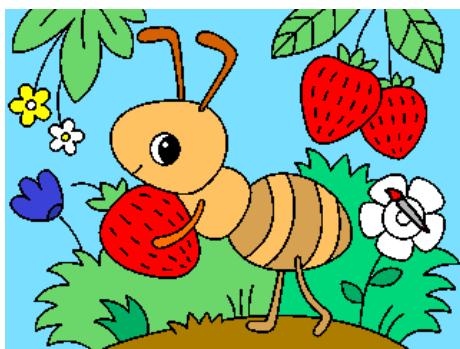
SD-card boot loader - the program is loaded into RAM memory and allows writing a new boot3 loader from the SD card to flash memory (possibly with the application). But it also saves the flash memory contents to a UF2 file without modifying the flash memory contents by running this program.

Books



Gingerbread House - a fairy tale book about a gingerbread house.

Colouring Book



Colouring Book - Children's colouring pages of images 320*240/16-bit in folder /COLBOOK.

Demos



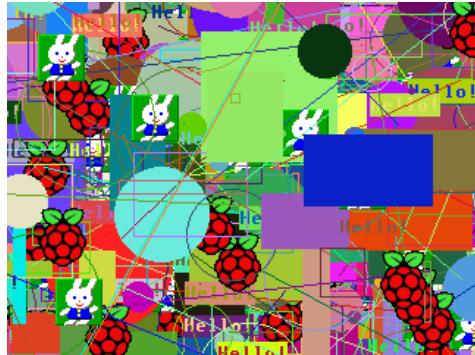
Balloons - flying balloons.

```

123456789! = 2.8535125219127859811439096
289626250005509472059934076823834073667
408136827054726611211860420557450762446
2657437181764363577796747711488238492633
3322648270001764571923736892904582466989
6595382399367382383892414333735355859626
66371253419285996252482185802702497335
8118065675547130592878118971462776174866
8851922891530855537291002687801914203
361892783245082665329102286307110453693
33815048245082665329102286307110453693
338140560898724185645942868842511974716
62619883763032802624598228885841529026
7626075544943693610749848228693553784568
99821371773386226718910372510779519971446
948018065320846525263235292614605650873092
57952235515342285483191863976289565288608
038443948038467073967952435899376247316
8191366483536222910723178356908766197252
197915164443862673847900056930778053474
996893916828972318433928928961538652478
8459465728266504158948286720346556212451372
0727592283383221657871136081620224393847
389543958372860993267668984226584476020
21465577826018059358631723751925674472994
5112028359816324818956939674751885603826
5017e+9453335859 Result OK

```

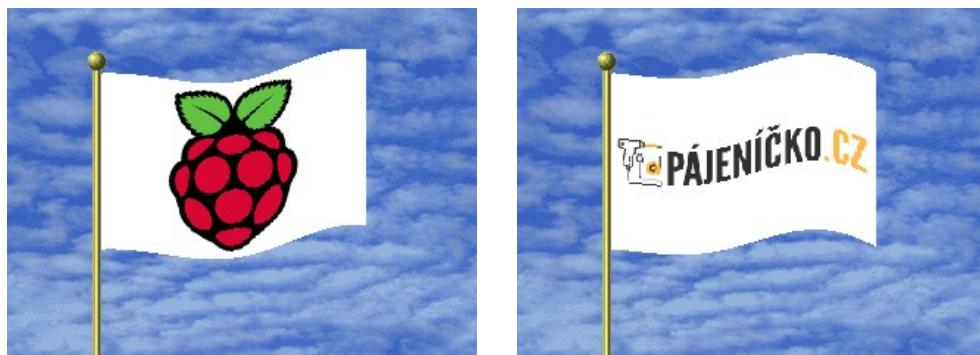
BigFact - calculating big factorial of number 123456789 to 1150 digits, using real4096.



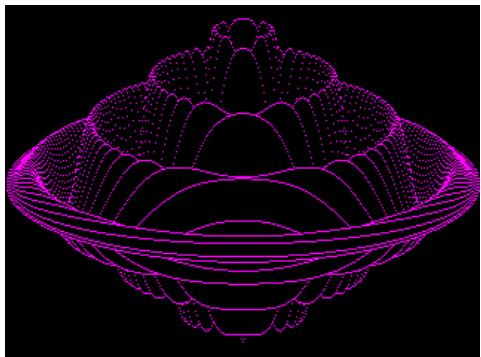
Draw - demonstration of drawing graphic elements. For the demonstration, alternate between slow rendering and drawing at maximum speed.



Earth - rotating globe. Software spherical image transformation.



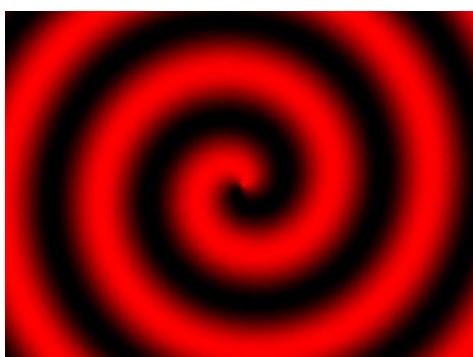
Flag - fluttering flag.



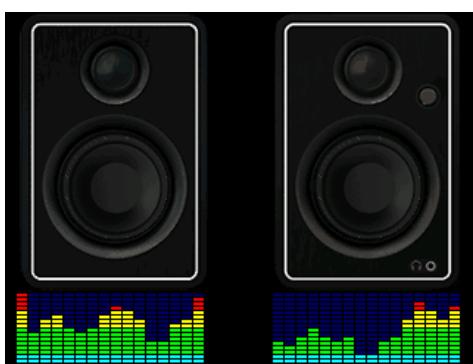
Fountain - draw 3D graph.



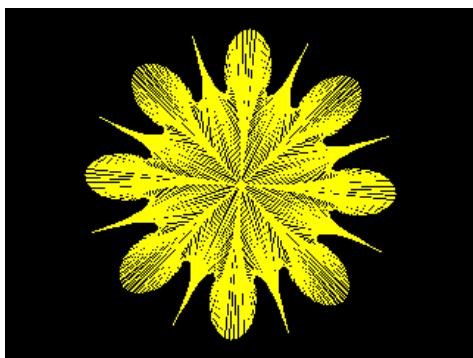
Hello World - the simplest example.



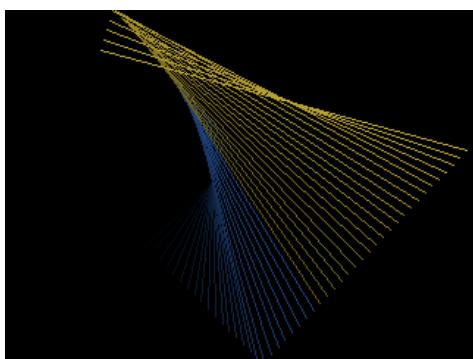
Hypno - a hypnotic rotating pattern. Example of matrix image transformation.



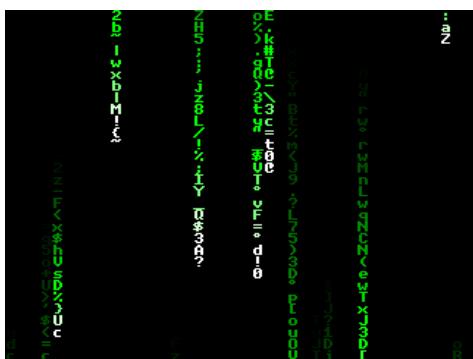
Level Meter - music spectrum indicator simulation (sound).



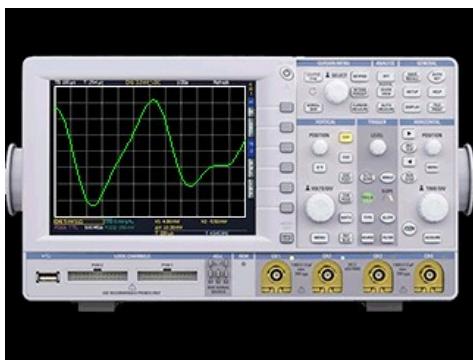
LineArt - draw line flower.



Lines - relaxation line pattern generator.



Matrix Rain - "matrix code rain" simulation.



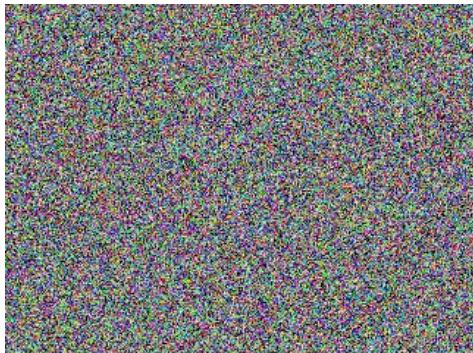
Oscilloscope - simulation of oscilloscope signal display.



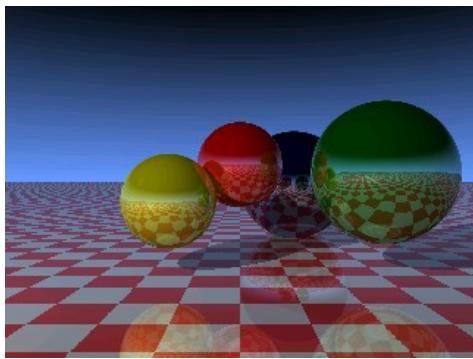
PF2024 - Christmas animation, playing song Silent Night.

```
Result OK: 3.141592653589793238462643383  
2795028841971693993751058289749445923078  
1640628620899862803482534211706798214888  
6513282306647893844609550582231725359488  
1284811174502841027819385211055596446229  
48954930838196442881897566593344612847564  
8233786783165271201989145648566923468348  
6104543266482133936872682491412737245870  
0660631558817488152892096282925489171536  
4367892590360011330530548820466521384146  
95194151160943305727036575919530921861  
1738193261179310511854807446237996274956  
7351885752724891227938183011949129833673  
3624406566430860213949463952247371907021  
7986094370277053921717629317675238467
```

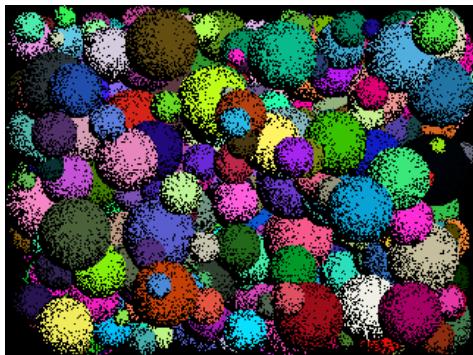
Pi - calculating number Pi to 1180 digits.



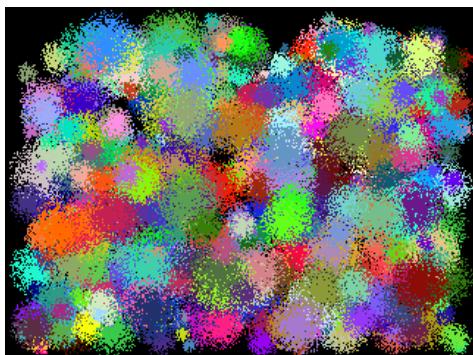
Pixels - random generation of colored pixels.



Raytracing - 3D pattern generation by ray tracing method, using both CPU cores.



Spheres - random spheres generation.



Spots - random generation of spots.



Twister - twisting of the textured block.



Water Surface - simulation of rippling water surface (sound).

Emulators



141PF - Busicom 141-PF calculator emulator with Intel 4004 processor and printer drum.



Test emulators of many processors.

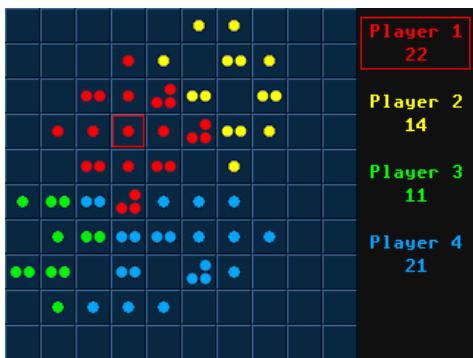


PC DOS emulator.

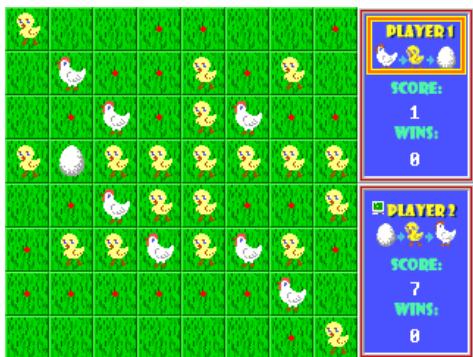
Games



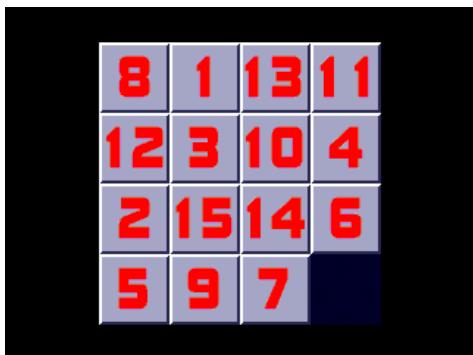
Ants - card game (sound). Supporting multiplayer over USB cable.



Exploding Atoms - game of exploding atoms (sound).



Eggs - logic game (sound). Based on the game Reversi.



Fifteen - logic game (sound). Sorting tiles from 1 to 15.



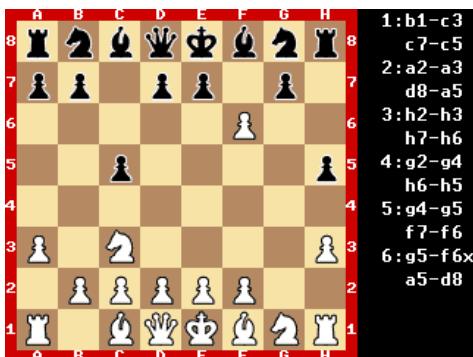
Fifteen Puzzle - logic game (sound). Sorting image tiles from 1 to 15.



Flappy - logic game for MZ-800 (sound). Move blue ball to blue wall.



Ghost Racing - car racing with your ghost from best lap (sound).



Chess - Chess game.



Space Invaders - Shooting game.



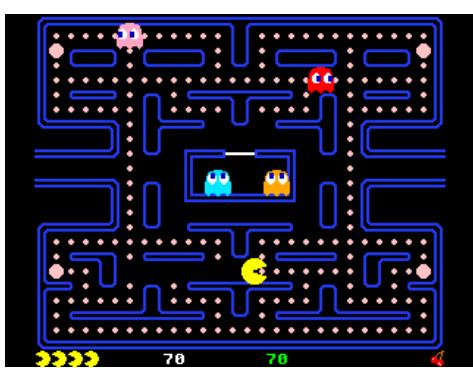
Life - cell life simulator (cellular automaton).



Maze - the goal is to find a way out of the maze.



Game Multiset - Multiset of 16 games (only PicoPad device).



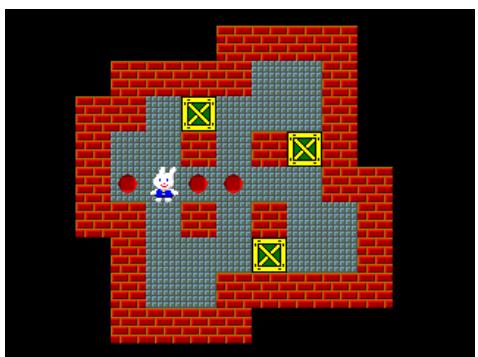
PacMan - Classic action game (sound).



Pictor - Shooting game (sound)



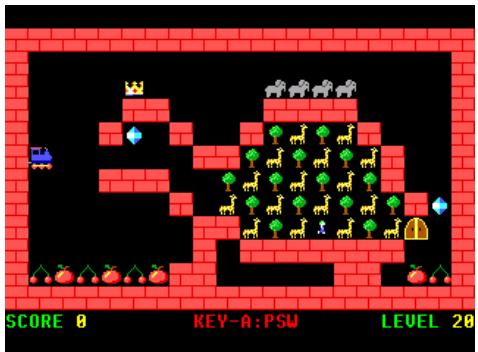
Raptor - Shooting game (sound).



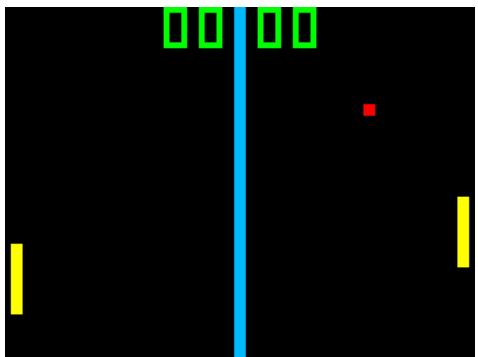
Sokoban - Logic game (sound). **3000 levels** from different authors, with their solutions.



Tetris - Game stacking blocks (sound).



Train - logic game based on the principle of the Snake (sound). 50 levels with solutions.



TV Tennis - Classic TV game (sound).



Vegaslot - Winning slot machine, based on a real slot machine (sound).



Zoom - Buck Rogers-Planet of Zoom

Measure

```
KEY_A/KEY_B mode: 5) Notes PWM Sine  
KEY_X output: GPIO14  
  
UP/DOWN select:  
F=466.16376Hz A#4  
T=2.1451689ms  
  
Real frequency:  
F=466.16376Hz  
T=2.1451689ms  
  
crystal=12'000'000Hz div=429033.77849  
sys_clk=200'000'000Hz mode=Slow_Precise
```

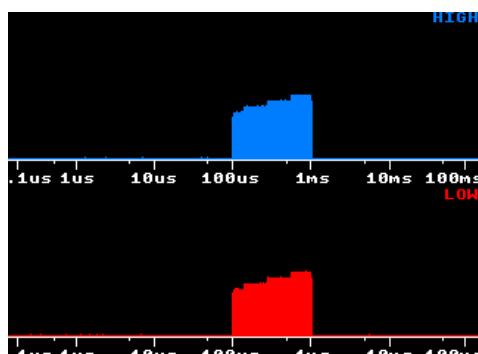
Frequency Generator - output to GPIO14 or GPIO15 (speaker), range 0.01Hz to 100 MHz.

```
GPIO1  
86.40002 MHz  
11.57407ns  
  
GPIO14  
1.929241 Hz  
518.3387ms
```

Frequency Meter - input from GPIO1, GPIO14, precision 6 digits in range up to 100 MHz.



Mini-Analyzer - Simple logic analyzer on GPIO0 with 125 MHz sampling rate and test noise generator on GPIO1 with sampling rate 15.75 MHz.

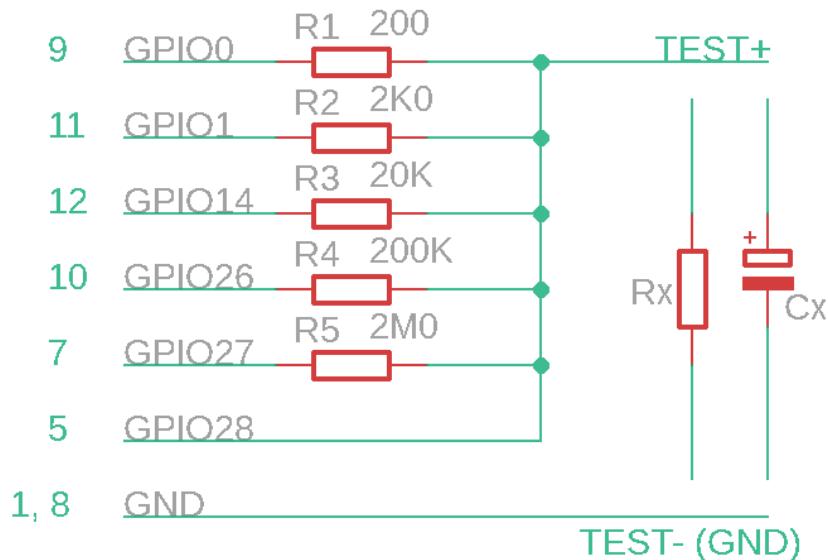


Pulsegram - spectrum histogram of HIGH and LOW pulses. Input from GPIO1, range 4 Hz to 7 MHz, logarithmic scale and logarithmic values.

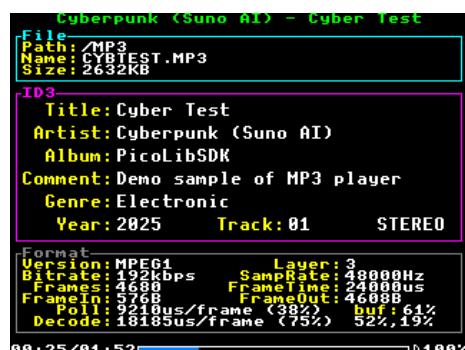
ESR:
8.34 ohm

Capacitance:
189.6 uF

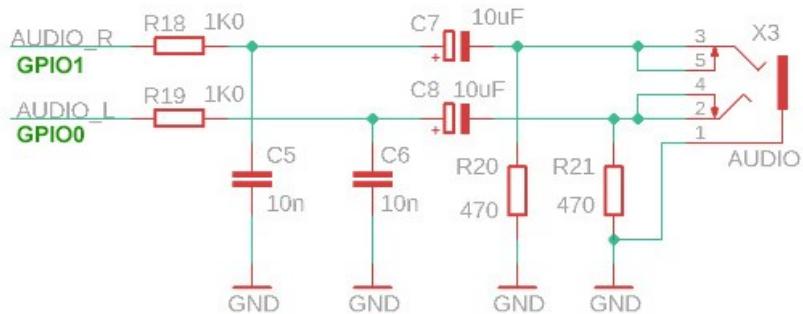
RC meter - measures the resistance of resistors, capacitance and ESR of capacitors. An adapter with 5 resistors, connected to an external connector, is needed according to the following diagram. GPIO0 ... resistor 200 ohm, GPIO1 ... resistor 2 Kohm, GPIO14 ... resistor 20 Kohm, GPIO26 ... resistor 200 Kohm, GPIO27 ... resistor 2 Mohm, GPIO28 ... test input. In case of PicoPadVGA it is necessary to disable the VGA output pins using the DIL switch. Resistors: 0.1 ohm up to 100 Mohm, precision 5% (lower on ends of range). Capacitors: 5 pF up to 5 mF, precision 10%. The capacitor measurement method uses a logarithmic regression of the capacitor charging and discharging curves.



MP3



MP3 Player - Player of MP3 files from the SD card, to PWMSnd output. MP3_GP01 is a variant with stereo audio output on GPIO0 and GPIO1. The following wiring is recommended:



New



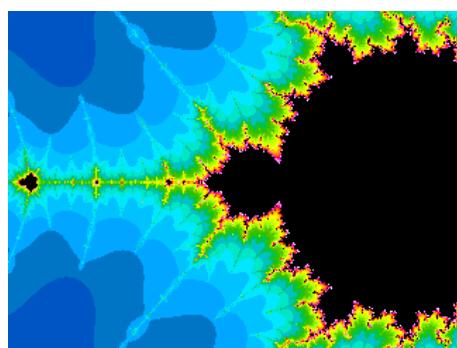
New project - Base of new project.

Painter



Painter - Graphic editor using USB mouse, folder /PAINTER.

Programs



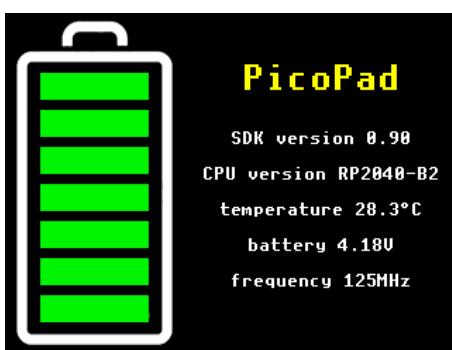
Mandelbrot - fractal pattern generator of Mandelbrot set. Using both CPU cores.

SlideShow



SlideShow - Slide show BMP image player from folder /SLIDE.

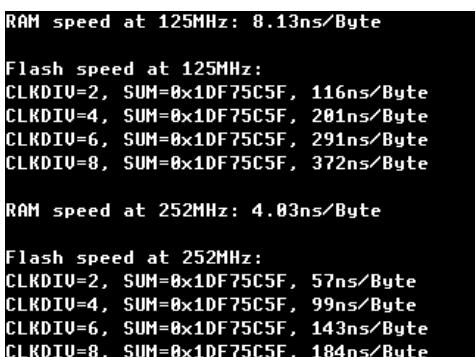
Tests



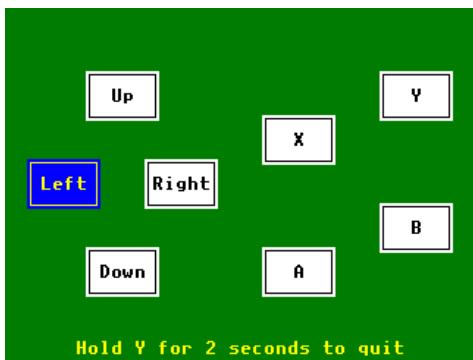
Battery - Display battery status.



BootSel - Resets Pico to mode of loading program via USB.



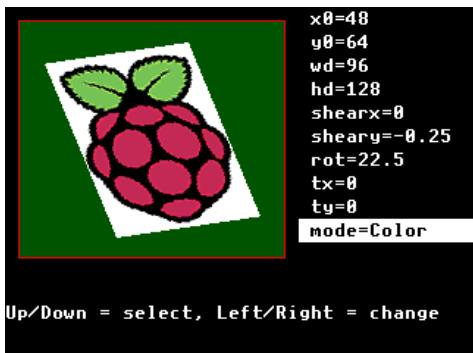
Flash Test - Flash Memory Speed Test.



KeyPad - test of keys.



Led - Test LED indicators.



Mat2D - Test Matrix 2D transformations.



Monoscope - Display video test pattern.

```
freq 300'000'000 Hz
freq 300'400'000 Hz
freq 302'400'000 Hz
```

OverClock - Check overclock limit of the CPU.



RGB Test - Color test pattern.

```
SD Card
SDType: SDHC
Media size: 7460 MB
File System: FAT32
Cluster size: 64 sectors (32768 bytes)
Root entries: 0 (0 on FAT32)
Disk size: 7452 MB
Used space: 50 MB
Free space: 7401 MB
Press A to list files in ROOT
Press B to HEX view file LOADER.UF2
Press X to write file TEST.TXT
```

SD Card - Test SD card and files.

```
its, lost on normal 0.00 digits (total 2
31.19)
real1024ext: 303.14 digits, lost 4.82 di
gits, lost on normal 0.00 digits (total
308.25)
real1536ext: 456.36 digits, lost 5.72 di
gits, lost on normal 0.00 digits (total
462.38)
real2048ext: 610.19 digits, lost 6.02 di
gits, lost on normal 0.00 digits (total
616.51)
real3072ext: 919.84 digits, lost 5.42 di
gits, lost on normal 0.00 digits (total
924.76)
OK
```

Test9 - Trigonometric Test or real numbers “The Forensics Evaluation Algorithm”.

USB

```
connecting USB keyboard...connected:  
key=09 (9) modi=00 char=f  
key=07 (?) modi=00 char=d  
key=0A (10) modi=00 char=g  
key=07 (?) modi=00 char=d  
key=08 (8) modi=00 char=e  
key=15 (21) modi=00 char=r  
key=0A (10) modi=00 char=g  
key=06 (6) modi=00 char=c  
key=07 (?) modi=00 char=d  
key=15 (21) modi=00 char=r  
key=15 (21) modi=00 char=r  
key=07 (?) modi=00 char=d
```

ExtKeyb - test external USB keyboard.

```
btn=00001 X=-401 Y=-121  
btn=00001 X=-404 Y=-121  
btn=00011 X=-412 Y=-119  
btn=00011 X=-420 Y=-116  
btn=00011 X=-422 Y=-116  
btn=00000 X=-422 Y=-116  
btn=00010 X=-422 Y=-116  
btn=00011 X=-422 Y=-116  
btn=00011 X=-422 Y=-115  
btn=00011 X=-421 Y=-114  
btn=00001 X=-421 Y=-114  
btn=00001 X=-411 Y=-120  
btn=00001 X=-408 Y=-123  
btn=00001 X=-407 Y=-123
```

ExtMouse - test external USB mouse.

```
connecting to the host...connected  
arrows ... move mouse  
A ..... left button  
B ..... right button  
X ..... keyboard text  
Y ..... quit
```

UsbDev - test USB device, simulate keyboard and mouse.

```
Speed: Low Speed 1.5 Mbps  
Reset signal 10 ms...OK  
Wait 400 ms...OK  
Get MaxPkt...8 B  
Set address...OK  
Get DevDesc...cls=0 sub=0 prot=0  
Get CfgDesc len...59 B  
Get CfgDesc...C-I-CC-E-I-CC-E  
Parsing...OK  
Set Cfg 1...OK  
Cfg Complete...OK  
Devices: Keyb Hid2  
  
A=save to SD, B=restart, Y=quit
```

UsbHost - diagnostic USB host enumeration.

```
Select port A=1 or B=2, Y=Quit
Connecting as port 2 (Slave)...OK
send=424KB/s, recv=424KB/s, err=0/0
send=437KB/s, recv=437KB/s, err=0/0
```

UsbPort - check speed and reliability of the USB mini-port transfer.

```
Connecting as Device...OK
Press A/B/X to send text, Y restart:
Nice blue text
Full row of nice colors
Double-width text
Very BIG text
Don't worry!
This text is inverted
How are you?
PicoPad is cool!
PicoPad is cool!
How are you?
Double-width text
```

UsbUart - connect USB host and device serial port.

Video



Video - video player to play video files *.VID from the SD card from folder /VIDEO.

10. Instruction Set

RP2040 Processor

The RP2040 processor used in the Raspberry Pico is a dual-core Cortex M0+ in ARMv6-M version, running at 125 MHz as standard (guaranteed frequency 133 MHz). The instruction set is Thumb-1, with a few added instructions from Thumb-2. So you won't find 32-bit ARM instructions here, and unfortunately not even the if-then IT conditional block instruction (this is only found in the ARMv6T2 version).

The original ARM processors contain a 32-bit ARM instruction set (each instruction is 32 bits). A limited set of 16-bit instructions, Thumb-1, was derived from the ARM instruction set, which is a subset of the ARM instructions and was designed for smaller processors. If a processor supports both instruction sets, it is possible to switch between them by setting bit 0 of the PC register. Setting bit 0 interprets the Thumb instructions, resetting bit 0 uses the ARM instruction set.

The Raspberry Pico processor RP2040 supports only the Thumb-1 instruction set. Therefore, bit 0 of the PC register must always be set. Attempting to switch to the ARM instruction set (by jumping to an address with bit 0 reset) will result in a hard-fail.

RP2040 Registers

All registers are 32-bit.

R0 to **R12** ... registers for general use. In most Thumb-1 instructions, only registers R0..R7 are supported.

Some registers are used by the C compiler for special purposes. Register R12 is the Intra Procedure Call Scratch Register (IPC). Register R11 is used as the Frame Pointer FP. Register R10 is the Stack Limit SL. Register R9 is used as Static Base SB.

R13 (SP) ... Stack Pointer. The processor has two stack pointers. MSP is Main stack. It serves as the main stack pointer when the processor starts and during interrupts. PSP is the Process stack, used for processes in a multitasking system. The current stack pointer, assigned to R13 (SP), is selected in the CONTROL register. The stack pointer points to the bottom edge of the stack. When the register is loaded into the stack, the SP pointer value is decremented.

R14 (LR) ... Link Register. During subroutine calls in ARM processors, the return address is not stored in the stack, but in the link register LR. When returning from a subroutine, a jump is made back to the contents of the link register. This has the advantage of faster execution of the subroutine because the stack does not have to be manipulated. However, if subroutines are called in greater depth, it is necessary to ensure that the LR register is preserved and restored. In most cases, a free register can be used, and so even in this case the subroutine call can be fast.

R15 (PC) ... Program Counter. The PC program counter points to the current address in program memory. Instructions in Thumb mode must be aligned to a multiple of 2 (i.e., an even address) because each instruction takes up 2 or 4 bytes. Therefore, the lowest unused bit of the PC register is used to indicate whether the ARM instruction set (bit 0 is cleared) or the Thumb instruction set (bit 0 is set) is being used in program processing. If you read the PC register in RP2040, bit 0 will always be set.

PSR ... Program Status Register is a special register containing the processor state. It is accessed through 3 subregisters: the APSR (Application Program Status Register, contains flags), the IPSR (Interrupt Program Status Register) and the EPSR (Execution Program Status Register).

PRIMASK ... is the interrupt masking register. In RP2040 it contains only the lowest bit 0, indicating global interrupt disable.

CONTROL ... is the processor control register. It sets the privileged mode and stack pointer mode.

When a function is called in gcc, registers R0 to R3 contain the input arguments to the function and also the return value from the function. If more input arguments are needed, additional arguments are passed via the stack.

The called function must preserve the contents of registers R4 to R11 and register R13 (SP). Conversely, the function may not preserve registers R0 to R3, R12 (IPC) and R14 (LR). In particular, register R14 (LR) must be watched and its contents preserved before calling the function so that a return from the subroutine is possible.

RP2040 Flags

Register APSR (Application Program Status Register) contains operation result flags in its highest 4 bits.

N, bit 31 ... Negation. The flag is a copy of the high bit of the 31 operation result register and indicates a negative number.

Z, bit 30 ... Zero. If flag Z is set to 1, it indicates a null result of the operation (or a match in the comparison operation).

C, bit 29 ... Carry. Flag C set to 1 indicates an overflow of the result of a non-sign operation.

V, bit 28 ... Overflow. Flag V set to 1 indicates an overflow of the result of the sign operation.

Let us consider flags C and V in more detail. Think of the flags as extending the registers by 1 high bit. The C transfer is set when the extension bit is set. When added together with the input C transfer, only the final result is critical, not the intermediate steps. Example:

[0] 0xFFFFFFFFE ... first operand

+ [0] 0x00000001 ... the second operand, adding it would be

the intermediate result [0] 0xFFFFFFFF

+ carry C=1 ... addition of input carry

Result is [1] 0x00000000. The expansion bit is set, so the output flag C is 1.

When subtracting, the situation is different. In ARM processors, subtraction is implemented as adding the inverted value (NOT) plus 1, i.e., adding a negative operand. Thus, when flag C is set, it does not mean an underflow of the result below 0 (i.e., the "borrow" flag), but an inverted underflow flag. Example:

[0] 0x00000002 ... first operand

- [0] 0x00000001 ... second operand

When subtracting, we first bitwise invert the second operand and change the operation to addition.

+ [0] 0xFFFFFFFFE ... the inverted second operand, adding it would be

the intermediate result [1] 0x00000000.

We still have to add 1 to get the negation of the second operand.

+ [0] 0x00000001 ... result is [1] 0x00000001

As you can see, the result of the operation is the correct value 1, but the C flag has been set. We have to look at C as an inverted Borrow flag for subtraction. If we add the input transfer C in the operation, we do it the same way as for addition and it doesn't matter that it has the meaning of an inverted underflow.

In understanding the overflow flag V, we can again help ourselves by extending the bit to extend (duplicate) the sign of the operand. The overflow flag V will be set if the result overflows out of range - which it will if the expansion bit is different from the sign bit. Example:

[1] 0x80000000 ... the first operand, this is the maximum negative number -2147483648

+ [1] 0xFFFFFFFF ... add the second operand, the number -1

The result is [1] 0x7FFFFFFF, i.e. the number -2147483649. From the difference between the expansion and sign bits, we can see that the result overflows outside the sign number range and therefore the overflow flag V will be set.

RP2040 Instructions

For each instruction, the number of clock cycles T are listed. For the Load and Store instructions, the time depends on the memory being accessed. When accessing regular RAM, the time is typically 2 clock cycles (16 ns on 125 MHz). When accessing the SIO registers (GPIO), the access time is 1 clock (8 ns on 125 MHz). When accessing some registers and external Flash, the time may be longer due to additional wait-state.

If an "S" appears at the end of the instruction name, it means that the instruction affects flags (does not apply to CMP and TST instructions, which always affect flags).

Unless otherwise specified, the instruction supports only registers R0..R7. The extended register set R0..R15 is supported only by the instructions MOV Rd,Rm, MOV PC,Rm, ADD Rd,Rm, ADD PC,Rm, CMP Rn,Rm, BX Rm, BLX Rm, MRS Rd,<reg>, and MSR <reg>,Rd.

Subtract instructions use Carry to mean the inverse of the Borrow underflow.

When writing assembler instructions, the # character precedes a numeric constant.

Instructions must not access an unaligned address (otherwise they will raise hard-fault) - that is, 16-bit access instructions must not access an odd address, and 32-bit access instructions must only access a 32-bit aligned address.

In the instructions, 32 bits is referred to as a word, 16 bits is a half-word, and 8 bits is a byte.

Conditional jump conditions b<cc>:

- EQ ... equal
- NE ... not equal
- CS (HS) ... carry set (unsigned higher or same)
- CC (LO) ... carry clear (unsigned lower)
- MI ... negative (minus)
- PL ... positive or zero (plus)
- VS ... overflow
- VC ... no overflow

- HI ... unsigned higher
- LS ... unsigned lower or same
- GE ... signed greater or equal
- LT ... signed less than
- GT ... signed greater than
- LE ... signed less or equal

Instruction	T	Flags	Meaning	Note
MOVS Rd,#<imm>	1	N Z --	Rd <- imm	imm is a constant 0..255
MOVS Rd,Rm	1	N Z --	Rd <- Rm	
MOV Rd,Rm	1	-----	Rd <- Rm	Registers R0..R15. If Rd=PC, Rm must not be PC. If Rd=SP, Rm must not be SP.
MOV PC,Rm	2	-----	PC <- Rm	Registers R0..R14. Performs a jump to the address in register Rm. Bit 0 of the address is ignored.
ADDS Rd,Rn,#<imm>	1	N Z C V	Rd <- Rn + imm	imm is a constant 0..7
ADDS Rd,Rn,Rm	1	N Z C V	Rd <- Rn + Rm	
ADD Rd,Rm	1	-----	Rd <- Rd + Rm	Registers R0..R14
ADD PC,Rm	2	-----	PC <- PC + Rm	Registers R0..R14. Performs a jump relative to the next address + 2 (e.g. at #0 skips 1 following instruction). Bit 0 of the resulting address is ignored.
ADDS Rd,#<imm>	1	N Z C V	Rd <- Rd + imm	imm is a constant 0..255
ADCS Rd,Rm	1	N Z C V	Rd <- Rd + Rm + C	sum with carry
ADD SP,#<imm>	1	-----	SP <- SP + imm	imm is a constant 0..508, must be a multiple of 4
ADD Rd,Sp,#<imm>	1	-----	Rd <- SP + imm	imm is a constant 0..1020, must be a multiple of 4
ADR Rd,<label>	1	-----	Rd <- label	label is in range PC..PC+1020. Address must be aligned to 32 bits (i.e. a multiple of 4). It is replaced by the ADD Rd,PC,#<imm> instruction during compilation.
ADD Rd,PC,#<imm>	1	-----	Rd <- PC + imm	imm is a constant 0..1020, it must be a multiple of 4. PC has the value of the following address + 2 aligned down to a multiple of 4.
SUBS Rd,Rn,Rm	1	N Z C V	Rd <- Rn - Rm	
SUBS Rd,Rn,#<imm>	1	N Z C V	Rd <- Rn - imm	imm is a constant 0..7
SUBS Rd,#<imm>	1	N Z C V	Rd <- Rd - imm	imm is a constant 0..255
SBCS Rd,Rm	1	N Z C V	Rd <- Rd - Rm - not C	
SUB SP,#<imm>	1	-----	SP <- SP - imm	imm is a constant 0..508, must be a multiple of 4
NEGS Rd,Rn	1	N Z C V	Rd <- 0 - Rn	negation, synonym of RSBS Rd,Rn,#0

				instruction
RSBS Rd,Rn,#0	1	N Z C V	Rd <- 0 - Rn	negation, synonym of NEGS Rd,Rn instruction
MULS Rd,Rm	1	N Z --	Rd <- Rd * Rm	C and V flags remain unchanged (they are undefined in older ARMs versions)
CMP Rn,Rm	1	N Z C V	Rn - Rm ?	Registers R0..R15.
CMPN Rn,Rm	1	N Z C V	Rn + Rm ?	comparison with negation of Rm
CMP Rn,#<imm>	1	N Z C V	Rn - imm ?	imm is a constant 0..255
ANDS Rd,Rm	1	N Z --	Rd <- Rd and Rm	
EORS Rd,Rm	1	N Z --	Rd <- Rd xor Rm	exclusive or
ORRS Rd,Rm	1	N Z --	Rd <- Rd or Rm	
BICS Rd,Rm	1	N Z --	Rd <- Rd and not Rm	bit clear
MVNS Rd,Rm	1	N Z --	Rd <- not Rm	move not
TST Rn,Rm	1	N Z --	Rn and Rm ?	AND test
LSLS Rd,Rm,#<shift>	1	N Z C -	Rd <- Rm << shift	A logical shift to the left. shift is a constant 0..31. 0 is inserted into the lower bits. Carry is the carry from the last high bit.
LSLS Rd,Rs	1	N Z C -	Rd <- Rd << Rs	A logical shift to the left. Bits 0..7 in Rs contain the number of shifts. 0 is inserted in the lower bits. Carry is the carry from the last high bit.
LSRS Rd,Rm,#<shift>	1	N Z C -	Rd <- Rm >> shift	Logical shift to the right. shift is a constant 0..31. 0 is inserted into the upper bits. Carry is the carry from the last lowest bit.
LSRS Rd,Rs	1	N Z C -	Rd <- Rd >> Rs	A logical shift to the right. Bits 0..7 in Rs contain the number of shifts. 0 is inserted in the upper bits. Carry is the carry from the last lowest bit.
ASRS Rd,Rm,#<shift>	1	N Z C -	Rd <- Rm >> shift	Arithmetic shift to the right. shift is a constant 0..31. Bit 31 (sign) is multiplied into the upper bits. Carry is the carry from the last lowest bit.
ASRS Rd,Rs	1	N Z C -	Rd <- Rd >> Rs	Arithmetic shift to the right. Bits 0..7 in Rs contain the number of shifts. Bit 31 (sign) is multiplied into the upper bits. Carry is the carry from the last lowest bit.
RORS Rd,Rs	1	N Z C -	Rd <- Rd >> Rs, wrap	Rotation to the right. Bits 0..7 in Rs contain the number of shifts. The released lower bits are transferred to the upper bits. Carry is the carry from the last lowest bit.
LDR Rd,[Rn,#<imm>]	1-2	----	Rd <- [Rn + imm]	imm is offset 0..124, must be a multiple of 4.
LDRH Rd,[Rn,#<imm>]	1-2	----	Rd <- [Rn + imm] [16]	imm is offset 0..62, multiple of 2. Reads 16 bits, fills the upper 16 bits with 0.

LDRB Rd,[Rn,#<imm>]	1-2	----	Rd <- [Rn + imm] [8]	imm je offset 0..31. Reads 8 bits, fills the upper 24 bits with 0.
LDR Rd,[Rn,Rm]	1-2	----	Rd <- [Rn + Rm]	
LDRH Rd,[Rn,Rm]	1-2	----	Rd <- [Rn + Rm] [16]	It reads 16 bits. Fills the upper 16 bits with 0.
LDRSH Rd,[Rn,Rm]	1-2	----	Rd <- [Rn + Rm] [16]	It reads 16 bits. The upper 16 bits are expanded from the signed 15th bit.
LDRB Rd,[Rn,Rm]	1-2	----	Rd <- [Rn + Rm] [8]	It reads 8 bits. Fills the upper 24 bits with 0.
LDRSB Rd,[Rn,Rm]	1-2	----	Rd <- [Rn + Rm] [8]	It reads 8 bits. The upper 24 bits are expanded from the signed 7th bit.
LDR Rd,<label>	1-2	----	Rd <- [label]	Loads 32-bits from the specified address. The address must be in the range PC..PC+1020 and must be aligned to 4 bytes.
LDR Rd,[SP,#<imm>]	1-2	----	Rd <- [SP + imm]	imm is a constant 0..1020, it must be a multiple of 4.
LDM Rn!,{<reglist>}	1+N	----	load <reglist>	Reads N registers according to reglist, from base address Rn, excluding register Rn. Register Rn will be incremented after the operation. The registers must be listed in ascending order in the list. Another name for the instruction is LDMIA (Load Multiply and Increment After).
LDM Rn,{<reglist>}	1+N	----	load <reglist>	Reads N registers according to reglist, from base address Rn, including register Rn. Register Rn must be specified in the register list. The registers must be listed in ascending order. Another name for the instruction is LDMIA (Load Multiply and Increment After).
STR Rd,[Rn,#<imm>]	1-2	----	Rd -> [Rn + imm]	imm is offset 0..124, it must be a multiple of 4.
STRH Rd,[Rn,#<imm>]	1-2	----	Rd -> [Rn + imm] [16]	imm is offset 0..62, multiple of 2. Stores 16 bits.
STRB Rd,[Rn,#<imm>]	1-2	----	Rd -> [Rn + imm] [8]	imm is offset 0..31. It stores 8 bits.
STR Rd,[Rn,Rm]	1-2	----	Rd -> [Rn + Rm]	
STRH Rd,[Rn,Rm]	1-2	----	Rd -> [Rn + Rm] [16]	Stores 16 bits.
STRB Rd,[Rn,Rm]	1-2	----	Rd -> [Rn + Rm] [8]	Stores 8 bits.
STR Rd,[SP,#<imm>]	1-2	----	Rd -> [SP + imm]	imm is a constant 0..1020, it must be a multiple of 4.
STM Rn!,{<reglist>}	1+N	----	store <reglist>	Stores N registers according to reglist, from base address Rn, excluding register Rn. Register Rn will be incremented after the operation. The registers must be listed in ascending order in the list. Another name for the instruction is STMIA (Store Multiply

				and Increment After).
PUSH {<reglist>}	1+N	----	push <reglist>	Stores the registers into the stack according to the list.
PUSH {<reglist>,LR}	1+N	----	push <reglist,LR>	It stores the registers in the tray according to the list, including the LR register.
POP {<reglist>}	1+N	----	pop <reglist>	Restores registers from the stack according to the list.
POP {<reglist>,PC}	1+N	----	pop <reglist,PC>	Restores registers from the stack according to the list, including the PC registry. This jumps to the original LR address. The PC address must have its lowest bit set to 0 (otherwise hardfault).
B<cc> <label>	1-2	----	if (cc) then PC <- label	Jump to the given label if the condition is met. The label must be max. -256..+254 distant from the next instruction + 2. When the condition is met (and the jump is executed) the instruction takes 2 clock cycles. If not met (and continued), it will take 1 clock cycle.
B <label>	2	----	PC <- label	Unconditional jump. Label must be within a range of max. +- 2 KB.
BL <label>	3	----	LR <- PC, PC <- label	Save the following address into LR and jump to the label. Must be in the range +- 4 MB.
BX Rm	2	----	PC <- Rm	Indirect jump to an address from the Rm register. Bit 0 of the address must be set (otherwise hardfault). Registers R0..R15.
BLX Rm	2	----	LR <- PC, PC <- Rm	Storage of the following address into LR and indirect jump to the address from the Rm register. Bit 0 of the address must be set (otherwise hardfault). Registers R0..R15.
SXTH Rd,Rm	1	----	Rd <- Rm [16]	Extends the lower 16 bits from Rm to Rd, fills the upper 16 bits with a sign bit.
SXTB Rd,Rm	1	----	Rd <- Rm [8]	Extends the lower 8 bits from Rm to Rd, fills the upper 24 bits with a sign bit.
UXTH Rd,Rm	1	----	Rd <- Rm [16]	Extends the lower 16 bits from Rm to Rd, clears the upper 16 bits.
UXTB Rd,Rm	1	----	Rd <- Rm [8]	Extends the lower 8 bits from Rm to Rd, clears the upper 24 bits
REV Rd,Rm	1	----	Rd <- revb Rm	Reverses the byte order of the Rm register
REV16 Rd,Rm	1	----	Rd <- revh Rm	Swaps the byte order of the halves Rm (b[0] <-> b[1], b[2] <-> b[3])
REVSH Rd,Rm	1	----	Rd <- revsh Rm	Reverses the byte order of the lower half of Rm and extends the sign to the upper half.

SVC #<imm>	-	----		Supervisor Call, imm is a constant 0..255
CPSID i	1	----		Global Interruption Disable
CPSIE i	1	----		Global Interruption Enable
MRS Rd,<reg>	3	----		Read the system register. Registers R0..R15
MSR <reg>,Rd	3	----		Write to the system register. Registers R0..R15
BKPT #<imm>	-	----		Breakpoint. imm is a constant 0..255.
SEV	1	----		Send event.
WFE	2	----		Waiting for the event. The time can be extended until the event arrives.
WFI	2	----		Waiting for an interrupt. The time can be extended until the interrupt arrives.
YIELD	1	----		Indication in multithread that a task is being executed.
NOP	1	----		No operation
ISB	3	----		Instruction synchronization
DMB	3	----		Data memory barrier
DSB	3	----		Data synchronization barrier