hsk-libs-user

257

Generated by Doxygen 1.8.12

Contents

Chapter 1

HSK XC878 µC Library Users' Manual

1.1 Preface

Welcome to the High Speed Karlsruhe (HSK) XC878 microcontroller (µC) users' manual. This document is intended for those who want to use the libraries for their own µC applications.

This document contains all the availabe library header documentation.

See also

PDF Version

1.2 About This Document

This document is work in progress, so far the documentation for the libraries is mostly complete. Documentation of implemented applications is less so and like the applications still subject to a lot of change.

1.2.1 Project Layout

- `LICENSE.md`
  - ISCL and 3rd party licensing
- `Makefile`
  - Makefile to invoke the SDCC and doxygen toolchain
- `Makefile.local`
  - Local non-revisioned Makefile for overriding default parameters
- `README.md`
  - Repository README
- `uVisionupdate.sh`
  - Updates the µVision project's overlaying instructions

- `bin.c51/`
  - C51 toolchain output produced by Keil µVision (safe to delete)
- `bin.sdcc/`
  - SDCC compiler output (safe to delete)
- `conf/`
  - Project configuration files
- `conf/doxygen.common`
  - Basic doxygen settings
- `conf/doxygen.dbc`
  - Doxygen setting changes to create documentation from DBC headers
- `conf/doxygen.dev`
  - Doxygen setting changes to create the developer documentation
- `conf/doxygen.scripts`
  - Doxygen setting changes to create the scripts documentation
- `conf/doxygen.user`
  - Doxygen setting changes to create the user documentation
- `conf/sdcc`
  - SDCC configuration, contains basic `CFLAGS` and invokes version specific platform hacks
- `doc/`
  - Documentation build directory
- `gen/`
  - Generated code e.g. the `.mk` files with build instructions
- `gen/dbc/`
  - C headers generated from Vector DBCs (via `scripts/dbc2c.awk`)
- `gh-pages/`
  - Project documentation, published at https://lonkamikaze.github.io/hsk-libs
- `gh-pages/contrib/`
  - This directory contains 3rd party documentation
- `gh-pages/contrib/ICM7228.pdf`
  - Intersil ICM7228 8-Digit, LED Display Decoder Driver data sheet
- `gh-pages/contrib/Microcontroller-XC87x-Data-Sheet-V15-infineon.pdf`
  - Data sheet for the Infineon XC87x series
- `gh-pages/contrib/XC878_um_v1_1.pdf`
  - Infineon XC878 User Manual Version 1.1
- `hacks/`
  - Storage directory for hacks that are pulled in depending on platform parameters like the SDCC version

- `img/`
  - Pictures included in this documentation
- `inc/`
  - 3rd party headers
- `scripts/`
  - Contains build scripts used by the `Makefile`, this folder is documented in the a dedicated document
- `src/`
  - The project source code
- `src/doc/`
  - This directory contains general documentation that is not specific to a library, application or a file, i.e. this chapter of the documentation
- `src/hsk_.../`
  - Directories with this prefix contain library code
- `uVision/`
  - ARM Keil µVision project files

Chapter 2

The XC878 8-Bit Microcontroller Platform

The XC878 is an Intel 8051/8052 compatible µC architecture. This entails strong memory limitations with severe implications to writing code.

The strength of the architecture is that the controller contains many specialized modules that, once set up, perform many tasks without intense interaction.

Critical for this project are the following kinds of modules:

- 10-Bit AD conversion channels
- Timers that can be triggered by external signals or perform PWM
- CAN controller

See also

XC878 Reference Manual: XC878_um_v1_1.pdf
ARM Keil Infineon XC878-16FF page: http://www.keil.com/dd/chip/4480.htm
Infineon XC87x Series Overview: http://www.infineon.com/cms/en/product/microcontrollers/8-bit/html?channel=db3a304323b87bc20123dcee653f7007&tab=2
8051 Basics Tutorial: http://www.8052.com/tut8051

2.1 Registers and Paging

The XC878 functions and modules are controlled through so called Special Function Registers (SFRs).

Due to the number of modules and functions of the controller a lot more registers are present than the 128 that can be addressed. These 128 register addressed in the upper dirctly addressable address range from `0x80` to `0xFF`.

To circumvent the 128 register limit each functional block of registers has a paging register that can be used to access different Pages of registers. In C code this is done using the SFR_PAGE() macro defined in the Infineon/↩ XC878.h header file that is provided by Keil µVision, the IDE used for this project or the headers can directly be downloaded from ARM.

Paging only affects code that directly interacts with the hardware. One of the benefits of *using* these libraries is that paging is not an issue in the logical code.

Each section of the XC878 Reference Manual has a Register Overview that contains a table of pages and registers.

2.2 Memory Limitations

The 8051 platform offers 128 bytes of `data` memory in the address range 0x00-0x7F in front of the SFR address range. Because 128 bytes are insufficient, the 8051 architecture knows several kinds of memory that are accessed in different manners and thus quicker or slower to access.

The 8052 has an additional 128 bytes of indirectly addressable memory. This memory is accessed through the key word `idata`. Access to `idata` is slower than to `data`. The syntax for declaring a variable in `idtata` memory is:

```
<type> idata <identifier> [= <value>];
```

The additional `idata` memory is located in the upper half of the address range. The lower half accesses `data` memory. Any `data` access to a pointer actually is `idata` access. This is why SFRs cannot be accessed with pointers. They are masked by `idata` memory.

The slowest kind of memory used by this library is the `xdata` memory. The `xdata` memory makes 3kb of additional memory available and the libraries place all large data structures in them.

Variables are declared in `xdata` with the following syntax:

```
<type> xdata <identifier> [= <value>];
```

The first 256 bytes of `xdata` memory are also accessible as 8 bit addressed `pdata`. Using `pdata` is faster than `xdata`. The `p` in `pdata` stands for paged. Historically the 8052 family of µCs used register `P2` for paging. The XC878 instead provides an SFR named XADDRH.

However current 8051 C compilers don't support paging. I.e. one would have to ensure that structs and buffers do not cross page borders and update `XADDRH` manually. So instead of making the code more complicated and messing with the linkers `XADDRH` is fixed to the first `xdata` page and `pdata` is simply used as an additional 256 bytes of relatively fast memory.

There also is a 128 bits wide memory range of `bit` variables, which is used by single bit variables of the type `bool`.

In contrast to the small amount of available RAM, 64k of ROM are available to hold executable code. Thus a program well designed to the XC878 is one that produces a lot of static code to reduce the required amount of runtime memory use.

Code resides in its own address range, the `code` block. The µC can be run from code residing in `xdata` as well, to bootstrap the µC. Constants can also be placed in the `code` block.

2.2.1 Overlaying

In order to mitigate the memory limitations of the platform the C51 and SDCC compilers perform an optimisation called overlaying.

The compilers build a call graph, much like the one in the documentation of main(). The call graph is a directed graph and functions (i.e. their local variables and parameters) may occupy the same space in memory, provided they cannot reach each other in the graph.

E.g. main() can reach both init() and run(), thus main() may not occupy the same memory as init() and run(). However run() and init() cannot reach each other, so they may store their data in the same memory.

This reduces the use of the stack, which is expensive in terms of runtime (this statement is not generally true, it just applies to the 8051 family of µCs).

Functions may not be called more than once at a time. I.e. they may not be recursive or called from regular code and interrupts both. Both compilers provide a `reentrant` keyword to make functions operate on the stack. Its use should be avoided if possible.

Both C51 and SDCC do not track function pointers. Thus creating a function pointer results in a false call in the call tree. A call through a function pointer is not added to the call tree.

The C51 tool chain offers call tree manipulations and SDCC provides the nooverlay pragma to mitigate this.

The section about Implications of Overlaying lists best practices to optimize code for overlaying.

2.3 Pointers

Due to the existence of different kinds of memory, pointers come in two variations, generic pointers and memory-specific pointers. Generic pointers take 3 bytes of memory and are by far the slowest to process. Memory-specific pointers take 1 byte for `data`, `idata` or `pdata` and 2 bytes for `xdata` or `code`. They are also faster to process.

Note that the `data` keyword needs to be explicitely specified for `data` pointers. Pointers declared without explicit mention of the memory type always result in generic pointers.

Pointers can be stored in different kinds of memory than the memory they point to:

```
<type> <ptr_target_mem> * <ptr_mem> <identifier>;
```

The following example creates an `idata` pointer to a struct in `xdata` memory:

```
struct foo xdata * idata p_foo;
```

Chapter 3

C51 Compiler Toolchain Setup

This section describes the necessary compiler toolchain setup based on the Keil µVision IDE.

3.1 Device

It is critical for device flashing and programming to select the correct version of the µC. This dialogue also allows you to select the extended linker and assembler. Doing so is imperative to perform the necessary link time optimisations to fit the libraries into the limited memory of the device.

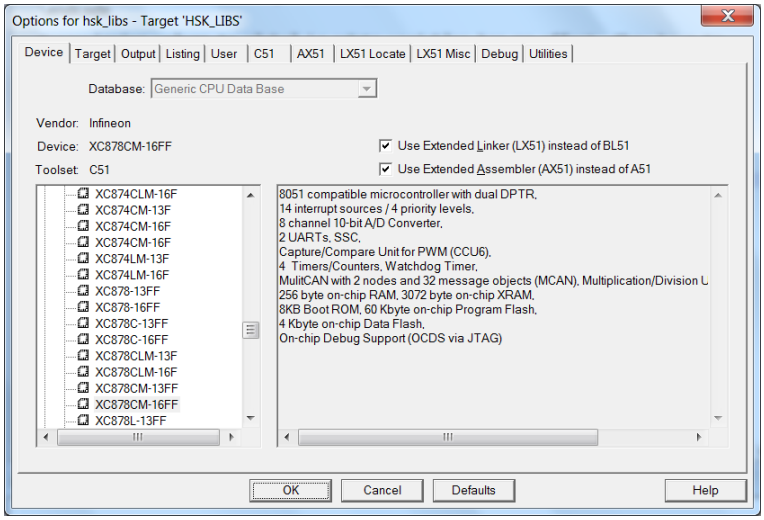


Figure 3.1 Keil µVision Device options dialogue

3.2 Target

The target dialogue lets you select several CPU architecture and memory layout settings.

The following options need to be set:

- Xtal (MHz):

  - This needs to be set to your external oscillator frequency, otherwise flashing and debugging might be unreliable

- Memory Model: Small
  - This setting means that variables are by default assigned to the first 128 bytes of directly addressable RAM, variables can still be mapped to different memory sections manaully as described in Memory Limitations
- Code ROM Size: Large: 64K program
  - This setting allows up to 64k of program data to be written to the device
- Use On-chip ROM
- Use On-chip XRAM
- Use multiple DPTR registers
  - This allows the compiler to reduce address writes of reoccuring pointer targets by using multiple pointer registers
- Safe address extension SFR in interrupts
  - XRAM/xdata access is not atomic. Thus interrupts using XRAM can interrupt and corrupt XRAM access of functions. This setting preserves the XRAM address registers and thus protects them from corruption

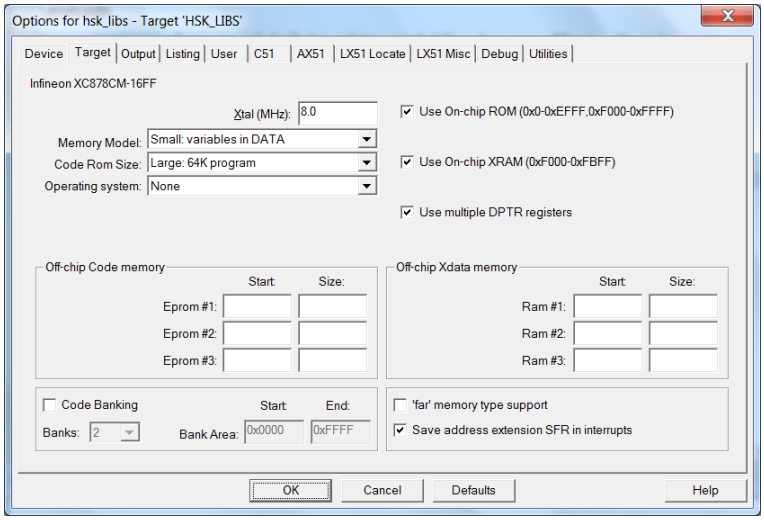


Figure 3.2 Keil µVision Target options dialogue

3.3 C51

The C51 is the C compiler configuration dialogue. The following settings are not obligatory for use of the HSK libraries, but recommended.

- Preprocessor Symbols
  - Define: `__xdata`, `__pdata`, `__idata`
    - This input field allows passing on preprocessor definitions to the preprocessor
    - The empty `__xdata`, `__pdata`, `__idata` defines allow C51 to ignore SDCC style memory assignments, this is useful to make such assignments where C51 does not support them

- Code Optimization
  - Level: 11: Reuse Common Exit Code
    - The highest level of optimisation, allowing the compiler the largest reduction of memory use
    - Select `4` or lower for debugging, all the common code eliminations prevent the debugger from mapping large chunks of C code to assembler code, making the program flow difficult to follow
  - Emphasis: Favor speed
    - Surprisingly this often produces smaller code than the `favor size` setting
  - Global Register Coloring
    - This setting allows the compiler to optimise register use throughout the entire application, reducing memory use and improving performance
  - Linker Code Packing
    - Activates a link time optimisation, after linking the application, the linker will replace long distance jumps with short jumps where applicable
- Warnings: Warninglevel 2
- Enable ANSI integer promotion rules

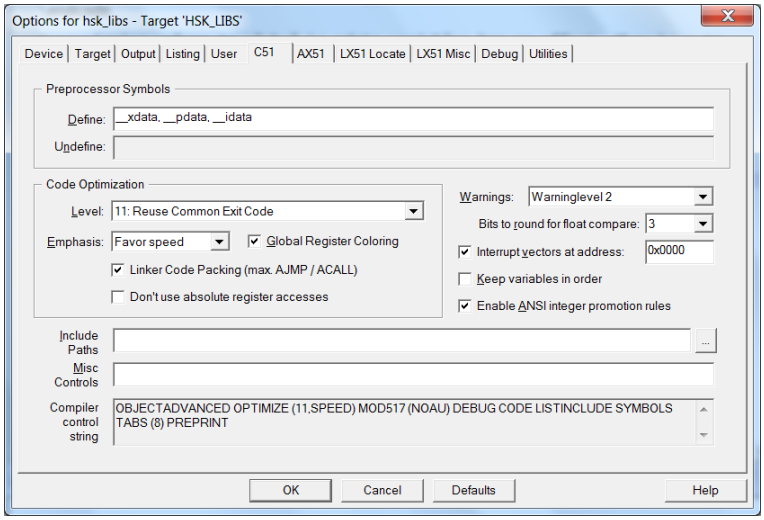


Figure 3.3 Keil µVision C51 options dialogue

3.4 LX51 Locate

LX51 is the extended linker of the C51 compiler tool chain, the Locate dialogue is used to map memory ranges. The form can also be used to assign portions of code to fixed addresses.

- User Memory Layout from Target Dialog
  - This option assigns the XC878 memory types to the appropriate address ranges
- User Classes: PDATA (X:0xF000-X:0xF0FF)
  - This option maps the `pdata` memory into the first 256 bytes of `xdata`

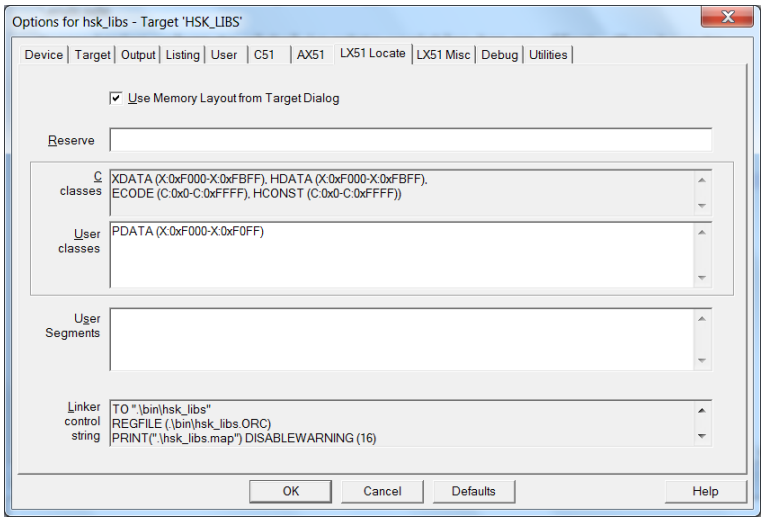


Figure 3.4 Keil µVision LX51 Locate options dialogue

3.5 LX51 Misc

The Misc dialogue holds the remaining linker settings.

- Overlay
  - This field can be used to add calls through function pointers to the call tree as is necessary for callback functions, the syntax is described in the µVision Help section OVERLAY Linker Directive
  - Manually filling this field can be avoided by running the `uVisionupdate`.sh script
- Misc controls: REMOVEUNUSED
  - This linker flag saves memory by discarding unused functions

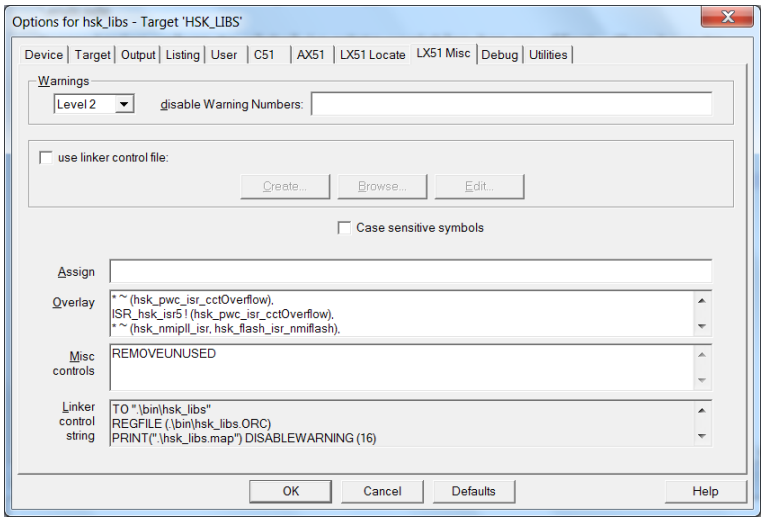


Figure 3.5 Keil µVision LX51 Misc options dialogue

3.6 Inline Assembler

Inline assembler has to be activated for Groups or single files individually. The Group Options can be found in the context menu of a Group.

To activate an option in this menu it needs to be unchecked and checked again.

- Generate Assembler SRC File
  - This option causes the compiler to generate assembler code instead of an object file
- Assemble SRC File
  - This option causes the compiler to assemble the generated assembler code

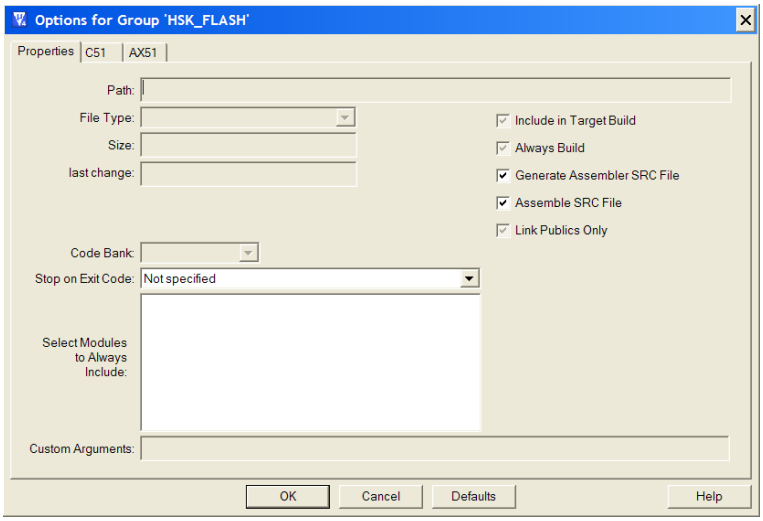


Figure 3.6 Keil µVision Group options dialogue

3.7 Device Programming and Debugging

To program or debug the device via On Chip Debug Support (OCDS) the Infineon Direct Access Server (DAS), a hardware access middleware, is required. The programming and debugging options can also be selected from the target options. Use the following settings in the Utilities tab:

- Use Target Driver for Flash Programming
  - Infineon DAS Client for XC800

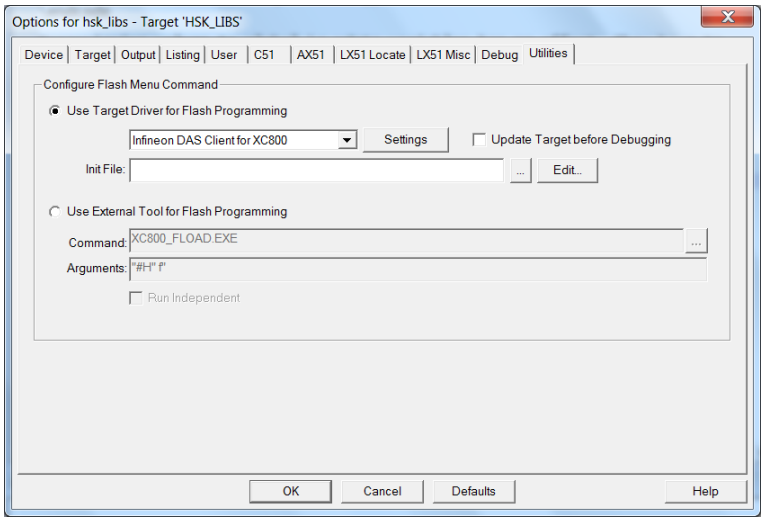


Figure 3.7 Keil µVision LX51 Utilities options dialogue

See also

Infineon DAS Tool Interface website: http://www.infineon.com/das/

Select `Settings` to enter the "Infineon XC800 DAS Driver Setup". The default options are mostly fine. Make sure of the following options:

- DAS Server: UDAS
- Target Debug Options
  - Miscellaneous Options
    - Disable Interrupts during Steps
      - Frequently occuring interrupts like timer interrupts make step by step debugging completely useless

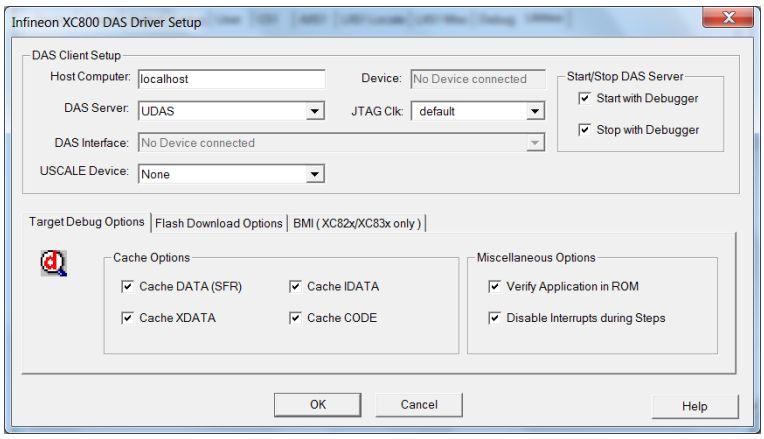


Figure 3.8 Keil µVision Infineon XC800 DAS Driver Setup

The Debug tab of the target options can be used to choose between in-simulator debugging and on-chip debugging.

Chapter 4

Using the Small Device C Compiler (SDCC)

This section describes how to compile code for the XC878 with the Small Device C Compiler. SDCC is an open source compiler supporting several 8 bit architectures.

This section is about using the compiler and maintaining C51 compatibility. Refer to The Project Makefile to build this project using SDCC.

See also

SDCC project: `http://sdcc.sf.net`
Small Device C Compiler Manual: `sdccman.pdf`

4.1 Processor Architecture

The 8051 architecture is selected with the parameter `-mmcs51`. Additionally the compiler needs to be invoked with the correct memory architecture, XRAM starts at addres 0xF000 and is 3kb wide. For this the parameters `--xram-loc` and `--xram-size` are provided:

```
-mmcs51 --xram-loc 0xF000 --xram-size 3072
```

4.2 SDCC Header File for XC878

This projects includes the file Infineon/XC878.h in the inc/ directory, which contains the SFR definitions for the XC878. It is a modified version of the XC878.h file proivided by Keil µVision, which in turn is a Dave generated file.

The modification is an `#ifdef SDCC` block, with some compatibility glue to allow using the C51 code mostly unmodified.

To make the header available to `sdcc` the inc/ should be added to the include search path with the `-I` parameter:

```
-mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/
```

4.3 Compiling Code

Code is compiled using the `-c` parameter:

```
sdcc -mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/ -o builddir/ -c example.c
```

The compiler will generate a number of files in builddir, among them `example.asm` and `example.rel`, the object file.

Instead of the build dir the output parameter `-o` can also take the name of the object file as a parameter.

SDCC can only compile one .c file at a time, thus every .c file must be compiled separately and linked in a separate step later.

4.4 Linking

Linking can be done by giving all required object files as parameters. The output file name will be based on the first input file or can explicitely be stated:

```
sdcc -mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/ -o builddir/ -c example.rel lib1.rel lib2.rel
```

The output file would be `builddir/example.ihx` (Intel HEX). `-o builddir/example.hex` can be used to change the filename suffix to `.hex`, which is more convenient when using XC800_FLOAD to flash the µC.

4.5 Programming

Whereas µVision as an IDE covers flashing, SDCC users need a separate tool to do so. One such tool is Infineon's XC800_FLOAD. FLOAD also requires DAS.

The use of FLOAD is very straightforward, make the following settings:

- Protocol: JTAG/SPD
- Physical Interface: UDAS/JTAG over USB
- Target Device: XC87x-16FF

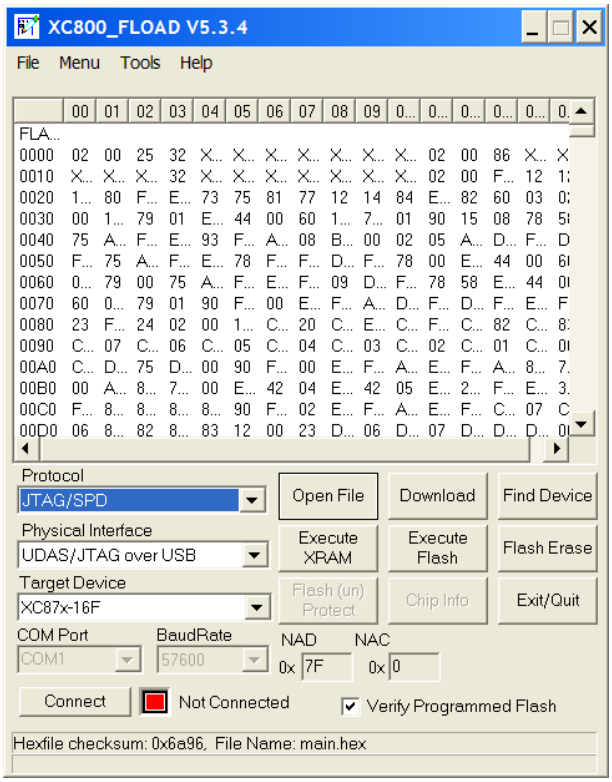


Figure 4.1 XC800_FLOAD Flash Programming Tool

The important functions of FLOAD are:

- Open: Open a HEX file for programming
- Download: Download the program to the µC flash
- Flash Erase: Clear the µC flash memory

See also

FLOAD download: http://www.infineon.com/cms/en/product/microcontrollers/development-tool
html?channel=db3a304319c6f18c011a0b54923431e5
Infineon DAS Tool Interface website: http://www.infineon.com/das/

4.6 Memory Usage Compatibility

In order to write compatible code, the compatiblity glue in the Infineon/XC878.h header maps keywords like `data`, `idata`, `xdata` etc. to their SDCC equivalents `__data`, `__idata` and `__xdata`.

Starting with SDCC 3.x C51 `code` and SDCC `__code` are largely interchangeable. This wasn't always the case, which results in two different styles for using `code`.

E.g. putting a variable into `code` space, C51 style:

```
ubyte code foo = 0x2A;
```

SDCC style:

```
const ubyte foo = 0x2A;
```

The C51 style is consistent with other variable space assignments and portable (it did not work with SDCC 2.x, but works with 3.x). The SDCC style is more logical in terms of writing code that communicates what one intends to do. The recommendation within this project is to use a redundant style, which also worked with SDCC 2.x with some C macro magic:

```
const ubyte code foo = 0x2A;
```

Function pointers are a special case. To SDCC all function pointers refer to `code`, C51 uses (slower, larger) generic pointers if the `code` keyword is missing. Unfortunately there is no compatible syntax to place `code` in a function pointer declaration. This can be circumvented with preprocessor instructions:

```
/*
 * SDCC does not like the \c code keyword for function pointers, C51 needs it
 * or it will use generic pointers.
 */
#ifdef SDCC
        #undef code
        #define code
#endif /* SDCC */
```

A function pointer declaration may look like this:

```
void (code *foo)(void);
```

Take care to restore `code` before the end of a `.h` file:

```
/*
 * Restore the usual meaning of \c code.
 */
#ifdef SDCC
        #undef code
        #define code    __code
#endif /* SDCC */
```

4.7 Interrupts

The most significant difference between interrupt handling in SDCC and C51 is that prototypes for the interrupts must be visible in the context of the main() function.

These prototypes can be enclosed in an `#ifdef`:

```
#ifndef _HSK_ISR_ISR_
#define _HSK_ISR_ISR_
void ISR_hsk_isr5(void) interrupt 5 using 1;
void ISR_hsk_isr6(void) interrupt 6 using 1;
void ISR_hsk_isr8(void) interrupt 8 using 1;
void ISR_hsk_isr9(void) interrupt 9 using 1;
void ISR_hsk_isr14(void) interrupt 14 using 2;
#endif /* _HSK_ISR_ISR_ */
```

According to the SDCC manual functions called by ISRs must be reentrant or protected from memory overlay. This is done with a compiler instruction:

```
#pragma save
#pragma nooverlay
void isr_callback(void) using 1 {
        [...]
}
#pragma restore
```

Unfortunately this causes a compiler warning when using C51:

```
..\src\main.c(49): warning C245: unknown #pragma, line ignored
```

This can be avoided by making nooverlay conditional:

```
#pragma save
#ifdef SDCC
#pragma nooverlay
#endif
void isr_callback(void) using 1 {
        [...]
}
#pragma restore
```

Chapter 5

The Project Makefile

The project Makefile offers access to all the UNIX command line facilities of the project. The file is written for the FreeBSD make, which is a descendant of PMake. Some convenience and elegance was sacrificed to make the Makefile GNU Make compatible.

5.1 Generating the Documentation

The Makefile can invoke Doxygen with the `make` targets `html` and `pdf`:

```
# make html pdf
Searching for include files...
Searching for example files...
Searching for images...
[...]
```

The `html` target creates the directories html/user/ and html/dev/, which contain the HTML version of this documentation.

The `pdf` target creates the directory pdf/ with the PDF versions of this documentation.

The targets create a Users' and a Developers' Manual. The first only includes documentation for public interfaces (i.e. headers). The second also includes the documentation of the implementation and some additional tidbits in this chapter that are only of interest when developing the libraries instead of building applications with them.

5.1.1 Dependencies

In order to build the documentation the following tools need to be installed on the system:

- Doxygen
- GraphViz (for creating dependency graphs)
- teTeX (for pdflatex)

5.2 Building

The Makefile uses SDCC to build. This can be changed in the first lines of the Makefile. The default target `build` builds all the .c files. Each .c file containing a main() function will also be linked, resulting in a .hex file:

```
# make
sdcc -mmcs51 [...] -o bin.sdcc/hsk_adc/hsk_adc.rel -c src/hsk_adc/hsk_adc.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_boot/hsk_boot.rel -c src/hsk_boot/hsk_boot.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_can/hsk_can.rel -c src/hsk_can/hsk_can.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_wdt/hsk_wdt.rel -c src/hsk_wdt/hsk_wdt.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_icm7228/hsk_icm7228.rel -c src/hsk_icm7228/hsk_icm7228.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_isr/hsk_isr.rel -c src/hsk_isr/hsk_isr.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_pwc/hsk_pwc.rel -c src/hsk_pwc/hsk_pwc.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_pwm/hsk_pwm.rel -c src/hsk_pwm/hsk_pwm.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_timers/hsk_timer01.rel -c src/hsk_timers/hsk_timer01.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_flash/hsk_flash.rel -c src/hsk_flash/hsk_flash.c
sdcc -mmcs51 [...] -o bin.sdcc/main.rel -c src/main.c
sdcc -mmcs51 [...] -o bin.sdcc/main.hex bin.sdcc/hsk_timers/hsk_timer01.rel [...]
```

All compiler output is dumped into the bin.sdcc/ directory. All the .c files are built, independent of whether they are linked into a .hex file.

5.3 Cygwin

Using a combination of native Binaries and Cygwin, the complete set of build and generator facilities can be used from Microsoft Windows.

The followin downloads are required:

- GIT: https://git-scm.com/downloads
- SDCC: https://sourceforge.net/projects/sdcc/files/
- Cygwin installer: http://cygwin.com/install.html

Additionally to the defaults the following Cygwin packages have to be installed:

- Devel: gcc-core
- Devel: libiconv
- Devel: make

After the installation the PATH variable should reference SDCC and Cygwin:

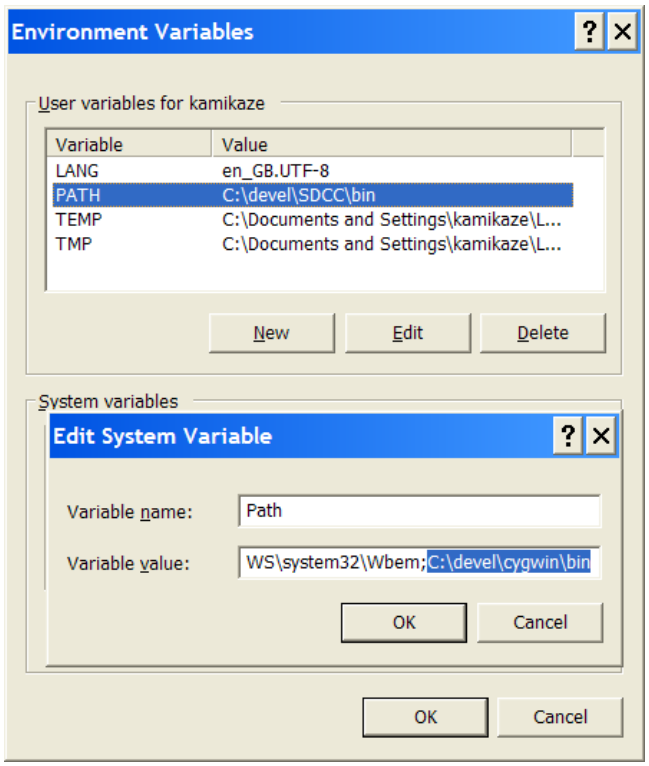


Figure 5.1 Cygwin and SDCC PATH environment

The libraries provide simple batch files to execute `uVisionupdate.sh` and call `make`. For ease of use the table of `make` targets is displayed:

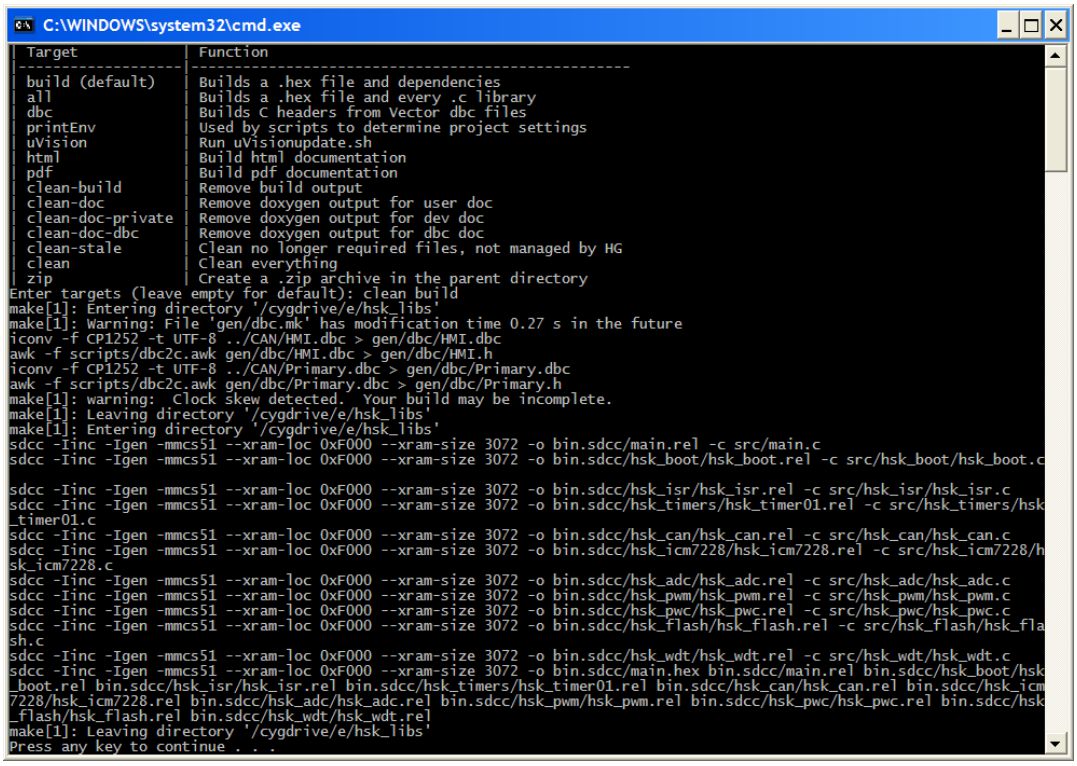


Figure 5.2 Make and available targets in Cygwin

Chapter 6

Code Requirements

To use these libraries the utilizing code must meet a small number of requirements.

6.1 SFR Pages

All public functions expect all pages set to 0 and reset all pages they touch to 0 before they exit.

All public functions also expect RMAP 0.

6.2 ISRs

The hsk_isr.h documentation lists the rules that need to be obeyed when implementing ISRs and callback functions:

- SFR Pages
- Register Banks

6.3 Memory

In order to access `pdata` and `xdata` the `hsk_boot` library must be linked.

Chapter 7

Variables and Memory

The Infineon/XC878.h header defines some unsigned data types:

- bool (1 bit)
- ulong (32 bits)
- uword (16 bits)
- ubyte (8 bits)

Signed types should only be used when necessary, floating point arithmetic should be avoided if in any way possible.

The correct place to store a variable depends on four factors:

- Size
- Lifetime
- Frequency of use during lifetime
- Overlay possibility

Size, and frequency are the most obvious, considering Memory Limitations. It is desireable to use fast memory for frequently accessed variables. Large data structures like buffers, arrays and structs simply use too much of the precious memory space to put them anywhere but `xdata`.

Both the C51 and SDCC compilers use a technique called overlay to fit all variables into memory. A stack is only used in reentrant functions. Only `data/idata` variables can be stacked with `push/pop` instructions. Stacking `xdata` is emulated in software and thus very slow.

The overlay approach is to build a call graph and thus decide which variables are never used at the same time. These variables are mapped to the same fixed memory addresses.

Variables with a long lifetime are locals in the main() function, static variables and global variables. These variables have to keep their state during the entire runtime. Thus they use memory space that cannot be shared with other variables.

ISRs and functions called by ISRs also cannot share memory, because there is no sensible way to make sure that a given function is not running when an interrupt occurs. In technical terms, each ISR is the root node of its own call graph.

The following table lists recommended memory types:

Context	Size	Critical	Memory
*	bool	*	bit
parameter	*	*	data
const	*	*	code
local	byte, word	*	data, idata
	>= long	no	xdata
		ISR/blocking	pdata, xdata
static/global	*	no	xdata
		ISR/blocking	pdata, xdata

`ISR/blocking` refers to memory accessed by ISRs, functions called back by ISRs and sections of code that block an interrupt.

7.1 Implications of Overlaying

First and foremost, Overlaying is only performed for the default memory. This project is built around the small memory model, i.e. only `data` memory is overlaid by SDCC and C51.

The `data` memory is only 128 bytes minus the used register banks large. With three register banks that means only 104 bytes of `data` memory are available. Thus non-overlayable variables should be placed in `idata` in order to use `data` memory for well overlayable variables.

Furthermore SDCC does not build a complete call tree, so it cannot eliminate unused functions like C51/LX51 and only overlays leaf functions. I.e. only functions that call no other functions are overlaid.

For SDCC overlaying ISRs is not possible, that is why locals of ISRs should in most cases be placed in `idata`, despite the performance impact.

A limitation of C51/LX51 is that it ignores explicit memory assignments in function parameters and always places them in the default memory, if they cannot be passed in registers. This limitation does not appear to be documented, but it was reported by an ARM support employee in case #530915.

SDCC does not share this limitation, it can place function parameters in all kinds of memory. Because such assignments are ignored by C51/LX51, parameter memory type should be optimised for SDCC. Explicit memory assignments prevent parameters from being passed in registers. SDCC only passes the first non-`bool` parameter in registers, so optimizations should be performed on the non-overlayable arguments following it.

For single use functions like `init` and `enable` functions, it might make sense to pass parameters in `xdata`. In such cases the SDCC memory assignment style should be used, to give the C51 preprocessor the chance to remove the assignments.

```
void hsk_can_init(const ubyte pins, const ulong __xdata baud);
```

If an application runs out of `data` space locals should be put into `idata` memory. Variables accessed by ISRs/ISR callbacks should be placed in `pdata` or `idata`. If that suffices the code should be checked for frequently accessed variables. The most frequent ones should be assigned to the default memory type if possible.

Both SDCC and C51 provide detailed information about memory use. The effects of relocating variables are often counter-intuitive, because it may interfere with several compiler and linker optimizations. This it is necessary to make use of this information in order to make sure that changes have the desired effect.

For SDCC check the assembler output for the `DSEG` area (search regex `/\\\.area DSEG/`):

```
;--------------------------------------------------------
; internal ram data
;--------------------------------------------------------
        .area DSEG    (DATA)
_hsk_adc_init_resolution_1_71:
        .ds 1
_hsk_adc_init_ctc_1_72:
        .ds 1
_hsk_adc_init_sloc0_1_0:
        .ds 2
_hsk_adc_init_sloc1_1_0:
        .ds 2
```

The `.map` file lists the complete memory layout produced by the linker (search regex `/^DSEG/`):

```
Area                    Addr        Size        Decimal Bytes (Attributes)
----------------        ----        ----        ------- ----- ------------
DSEG                 00000000    00000080 =         128. bytes (REL,CON)

      Value  Global                             Global Defined In Module
      -----  --------------------------------   ------------------------
     00000018  _hsk_pwm_init_PARM_2               hsk_pwm
     0000001C  _hsk_pwm_channel_set_PARM_2        hsk_pwm
     0000001E  _hsk_pwm_channel_set_PARM_3        hsk_pwm
     00000025  _hsk_icm7228_writeDec_PARM_2       hsk_icm7228
     00000027  _hsk_icm7228_writeDec_PARM_3       hsk_icm7228
     00000028  _hsk_icm7228_writeDec_PARM_4       hsk_icm7228
...
```

In µVision the `.map` file can be accessed by double clicking the project in the project tree view (search string `"D A T A"`):

```
START     STOP      LENGTH    ALIGN  RELOC    MEMORY CLASS   SEGMENT NAME
============================================================================

* * * * * * * * * *   D A T A   M E M O R Y   * * * * * * * * * * * * * *
000000H   000007H   000008H   ---    AT..     DATA           "REG BANK 0"
000008H   00000FH   000008H   ---    AT..     DATA           "REG BANK 1"
000010H   000017H   000008H   ---    AT..     DATA           "REG BANK 2"
000018H   00001BH   000004H   BYTE   UNIT     IDATA          _IDATA_GROUP_
00001CH.0 00001FH.7 000004H.0 ---    ---      **GAP**
000020H.0 000020H.4 000000H.5 BIT    UNIT     BIT            _BIT_GROUP_
000020H.5 000020H.5 000000H.1 BIT    UNIT     BIT            ?BI?HSK_CAN
000020H.6 000020H   000000H.2 ---    ---      **GAP**
000021H   00003FH   00001FH   BYTE   UNIT     DATA           _DATA_GROUP_
000040H   000040H   000001H   BYTE   UNIT     IDATA          ?STACK
```

See also

Overlaying in the SDCC manual
Global Registers used for Parameter Passing in the SDCC manual

Chapter 8

Authors

Authors use short tags in the code, this is the complete list of authors and the aliases they use.

Author

kami
Dominic Fandrey dominic.fandrey@highspeed-karlsruhe.de
kamikaze kamikaze@bsdforen.de
Head of Electronics Development season 2010/2011, 2011/2012

Chapter 9

Deprecated List

Global CAN_ENDIAN_INTEL

In favour of shorter and cleaner code the hsk_can_data_getSignal() and hsk_can_data_setSignal() functions were switched to using boolean (motorola positive) logic

Global CAN_ENDIAN_MOTOROLA

In favour of shorter and cleaner code the hsk_can_data_getSignal() and hsk_can_data_setSignal() functions were switched to using boolean (motorola positive) logic

Global hsk_adc_open

Use hsk_adc_open10() or hsk_adc_open8() as appropriate

Global hsk_adc_warmup

Use hsk_adc_warmup10()

Chapter 10

Module Index

10.1 Modules

Here is a list of all modules:

Chapter 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 13

Module Documentation

13.1 CAN Node Status Fields

This group of defines specifies status fields that can be queried from hsk_can_status().

Macros

- #define CAN_STATUS_LEC 0

  The Last Error Code field provides the error triggered by the last message on the bus.
- #define CAN_STATUS_TXOK 1

  Message Transmitted Successfully.
- #define CAN_STATUS_RXOK 2

  Message Received Successfully.
- #define CAN_STATUS_ALERT 3

  Alert Warning.
- #define CAN_STATUS_EWRN 4

  Error Warning Status.
- #define CAN_STATUS_BOFF 5

  Bus-off Status.

13.1.1 Detailed Description

This group of defines specifies status fields that can be queried from hsk_can_status().

13.1.2 Macro Definition Documentation

13.1.2.1 CAN_STATUS_ALERT

```
#define CAN_STATUS_ALERT 3
```

Alert Warning.

Return values

0	No warnings
1	One of the following error conditions applies: CAN_STATUS_EWRN; CAN_STATUS_BOFF

13.1.2.2 CAN_STATUS_BOFF

```
#define CAN_STATUS_BOFF 5
```

Bus-off Status.

Return values

0	The bus is not off
1	The bus is turned off due to an error counter exceeding 256

13.1.2.3 CAN_STATUS_EWRN

```
#define CAN_STATUS_EWRN 4
```

Error Warning Status.

Return values

0	No error warnings exceeded
1	An error counter has exceeded the warning level of 96

13.1.2.4 CAN_STATUS_LEC

```
#define CAN_STATUS_LEC 0
```

The Last Error Code field provides the error triggered by the last message on the bus.

For details check table 16-8 from the User Manual 1.1.

Return values

0	No Error
1	Stuff Error, 5 consecutive bits of the same value are stuffed, this error is triggered when the stuff bit is missing
2	Form Error, the frame format was violated
3	Ack Error, the message was not acknowledged, maybe nobody else is on the bus
4	Bit1 Error, a recessive (1) bit was sent out of sync
5	Bit0 Error, a recessive (1) bit won against a dominant (0) bit
6	CRC Error, wrong checksum for a received message

13.1.2.5 CAN_STATUS_RXOK

```
#define CAN_STATUS_RXOK 2
```

Message Received Successfully.

Return values

0	No successful receptions since the last time this field was queried
1	A message was received successfully

13.1.2.6 CAN_STATUS_TXOK

```
#define CAN_STATUS_TXOK 1
```

Message Transmitted Successfully.

Return values

0	No successful transmission since TXOK was queried last time
1	A message was transmitted and acknowledged successfully

13.2 External Interrupt Channels

This group consists of defines representing external interrupt channels.

Macros

- #define EX_EXINT0 0
  External interrupt channel EXINT0.
- #define EX_EXINT1 1
  External interrupt channel EXINT1.
- #define EX_EXINT2 2
  External interrupt channel EXINT2.
- #define EX_EXINT3 3
  External interrupt channel EXINT3.
- #define EX_EXINT4 4
  External interrupt channel EXINT4.
- #define EX_EXINT5 5
  External interrupt channel EXINT5.
- #define EX_EXINT6 6
  External interrupt channel EXINT6.

13.2.1 Detailed Description

This group consists of defines representing external interrupt channels.

13.2.2 Macro Definition Documentation

13.2.2.1 EX_EXINT0

```
#define EX_EXINT0 0
```

External interrupt channel EXINT0.

Mask with EA, disable with EX0.

13.2.2.2 EX_EXINT1

```
#define EX_EXINT1 1
```

External interrupt channel EXINT1.

Mask with EA, disable with EX1.

13.2.2.3 EX_EXINT2

```
#define EX_EXINT2 2
```

External interrupt channel EXINT2.

Mask with EX2.

13.2.2.4 EX_EXINT3

```
#define EX_EXINT3 3
```

External interrupt channel EXINT3.

Mask with EXM.

13.2.2.5 EX_EXINT4

```
#define EX_EXINT4 4
```

External interrupt channel EXINT4.

Mask with EXM.

13.2.2.6 EX_EXINT5

```
#define EX_EXINT5 5
```

External interrupt channel EXINT5.

Mask with EXM.

13.2.2.7 EX_EXINT6

```
#define EX_EXINT6 6
```

External interrupt channel EXINT6.

Mask with EXM.

13.3 External Interrupt Triggers

This group contains defines representing the different edge triggers.

Macros

- #define EX_EDGE_RISING 0

  Trigger interrupt on rising edge.
- #define EX_EDGE_FALLING 1

  Trigger interrupt on falling edge.
- #define EX_EDGE_BOTH 2

  Trigger interrupt on both edges.

13.3.1 Detailed Description

This group contains defines representing the different edge triggers.

13.3.2 Macro Definition Documentation

13.3.2.1 EX_EDGE_BOTH

```
#define EX_EDGE_BOTH 2
```

Trigger interrupt on both edges.

13.3.2.2 EX_EDGE_FALLING

```
#define EX_EDGE_FALLING 1
```

Trigger interrupt on falling edge.

13.3.2.3 EX_EDGE_RISING

```
#define EX_EDGE_RISING 0
```

Trigger interrupt on rising edge.

13.4 External Interrupt Input Ports

Each define of this group represents an external interrupt port configuration.

Macros

- #define EX_EXINT0_P05 0

  External interrupt EXINT0 input port P0.5.
- #define EX_EXINT3_P11 1

  External interrupt EXINT3 input port P1.1.
- #define EX_EXINT0_P14 2

  External interrupt EXINT0 input port P1.4.
- #define EX_EXINT5_P15 3

  External interrupt EXINT5 input port P1.5.
- #define EX_EXINT6_P16 4

  External interrupt EXINT6 input port P1.6.
- #define EX_EXINT3_P30 5

  External interrupt EXINT3 input port P3.0.
- #define EX_EXINT4_P32 6

  External interrupt EXINT4 input port P3.2.
- #define EX_EXINT5_P33 7

  External interrupt EXINT5 input port P3.3.
- #define EX_EXINT6_P34 8

  External interrupt EXINT6 input port P3.4.
- #define EX_EXINT4_P37 9

  External interrupt EXINT4 input port P3.7.
- #define EX_EXINT3_P40 10

  External interrupt EXINT3 input port P4.0.
- #define EX_EXINT4_P41 11

  External interrupt EXINT4 input port P4.1.
- #define EX_EXINT6_P42 12

  External interrupt EXINT6 input port P4.2.
- #define EX_EXINT5_P44 13

  External interrupt EXINT5 input port P4.4.
- #define EX_EXINT6_P45 14

  External interrupt EXINT6 input port P4.5.
- #define EX_EXINT1_P50 15

  External interrupt EXINT1 input port P5.0.
- #define EX_EXINT2_P51 16

  External interrupt EXINT2 input port P5.1.
- #define EX_EXINT5_P52 17

  External interrupt EXINT5 input port P5.2.
- #define EX_EXINT1_P53 18

  External interrupt EXINT1 input port P5.3.
- #define EX_EXINT2_P54 19

  External interrupt EXINT2 input port P5.4.
- #define EX_EXINT3_P55 20

  External interrupt EXINT3 input port P5.5.
- #define EX_EXINT4_P56 21

  External interrupt EXINT4 input port P5.6.
- #define EX_EXINT6_P57 22

  External interrupt EXINT6 input port P5.7.

13.4.1 Detailed Description

Each define of this group represents an external interrupt port configuration.

13.4.2 Macro Definition Documentation

13.4.2.1 EX_EXINT0_P05

```
#define EX_EXINT0_P05 0
```

External interrupt EXINT0 input port P0.5.

13.4.2.2 EX_EXINT0_P14

```
#define EX_EXINT0_P14 2
```

External interrupt EXINT0 input port P1.4.

13.4.2.3 EX_EXINT1_P50

```
#define EX_EXINT1_P50 15
```

External interrupt EXINT1 input port P5.0.

13.4.2.4 EX_EXINT1_P53

```
#define EX_EXINT1_P53 18
```

External interrupt EXINT1 input port P5.3.

13.4.2.5 EX_EXINT2_P51

```
#define EX_EXINT2_P51 16
```

External interrupt EXINT2 input port P5.1.

13.4.2.6 EX_EXINT2_P54

```
#define EX_EXINT2_P54 19
```

External interrupt EXINT2 input port P5.4.

13.4.2.7 EX_EXINT3_P11

```
#define EX_EXINT3_P11 1
```

External interrupt EXINT3 input port P1.1.

13.4.2.8 EX_EXINT3_P30

```
#define EX_EXINT3_P30 5
```

External interrupt EXINT3 input port P3.0.

13.4.2.9 EX_EXINT3_P40

```
#define EX_EXINT3_P40 10
```

External interrupt EXINT3 input port P4.0.

13.4.2.10 EX_EXINT3_P55

```
#define EX_EXINT3_P55 20
```

External interrupt EXINT3 input port P5.5.

13.4.2.11 EX_EXINT4_P32

```
#define EX_EXINT4_P32 6
```

External interrupt EXINT4 input port P3.2.

13.4.2.12 EX_EXINT4_P37

```
#define EX_EXINT4_P37 9
```

External interrupt EXINT4 input port P3.7.

13.4.2.13 EX_EXINT4_P41

```
#define EX_EXINT4_P41 11
```

External interrupt EXINT4 input port P4.1.

13.4.2.14 EX_EXINT4_P56

```
#define EX_EXINT4_P56 21
```

External interrupt EXINT4 input port P5.6.

13.4.2.15 EX_EXINT5_P15

```
#define EX_EXINT5_P15 3
```

External interrupt EXINT5 input port P1.5.

13.4.2.16 EX_EXINT5_P33

```
#define EX_EXINT5_P33 7
```

External interrupt EXINT5 input port P3.3.

13.4.2.17 EX_EXINT5_P44

```
#define EX_EXINT5_P44 13
```

External interrupt EXINT5 input port P4.4.

13.4.2.18 EX_EXINT5_P52

```
#define EX_EXINT5_P52 17
```

External interrupt EXINT5 input port P5.2.

13.4.2.19 EX_EXINT6_P16

```
#define EX_EXINT6_P16 4
```

External interrupt EXINT6 input port P1.6.

13.4.2.20 EX_EXINT6_P34

```
#define EX_EXINT6_P34 8
```

External interrupt EXINT6 input port P3.4.

13.4.2.21 EX_EXINT6_P42

```
#define EX_EXINT6_P42 12
```

External interrupt EXINT6 input port P4.2.

13.4.2.22 EX_EXINT6_P45

```
#define EX_EXINT6_P45 14
```

External interrupt EXINT6 input port P4.5.

13.4.2.23 EX_EXINT6_P57

```
#define EX_EXINT6_P57 22
```

External interrupt EXINT6 input port P5.7.

13.5 Input Port Access

This group contains defines and macros to initialize port pins as inputs and read them.

Macros

- #define IO_PORT_IN_INIT(port, pins)

  Initializes a set of port pins as inputs.
- #define IO_PORT_ON_GND 0

  Bit mask to set the logical 1 to GND level for all selected pins.
- #define IO_PORT_ON_HIGH 0xff

  Bit mask to set the logical 1 to high level for all selected pins.
- #define IO_PORT_GET(port, pins, on)

  Evaluates to a bit mask of logical pin states of a port.

13.5.1 Detailed Description

This group contains defines and macros to initialize port pins as inputs and read them.

13.5.2 Macro Definition Documentation

13.5.2.1 IO_PORT_GET

```
#define IO_PORT_GET(
              port,
              pins,
              on )
```

Value:

```
( \
        (port##_DATA ^ ~(on)) & (pins) \
)
```

Evaluates to a bit mask of logical pin states of a port.

Note

Can also be used for Output Port Access

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

port	The parallel port to access
pins	A bit mask of the pins to select
on	A bit mask of pins that defines the states which represent on

13.5.2.2 IO_PORT_IN_INIT

```
#define IO_PORT_IN_INIT(
              port,
              pins )
```

Value:

```
{ \
        port##_DIR &= ~(pins); \
}
```

Initializes a set of port pins as inputs.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

port	The parallel port to configure
pins	A bit mask of the pins to select

13.5.2.3 IO_PORT_ON_GND

```
#define IO_PORT_ON_GND 0
```

Bit mask to set the logical 1 to GND level for all selected pins.

Note

Can also be used for Output Port Access

13.5.2.4 IO_PORT_ON_HIGH

```
#define IO_PORT_ON_HIGH 0xff
```

Bit mask to set the logical 1 to high level for all selected pins.

Note

Can also be used for Output Port Access

13.6 Output Port Access

This group contains macros and defines to initialize port pins for output and safely set output states.

Macros

- #define IO_PORT_STRENGTH_WEAK 0

  Bit mask to set weak drive strength for all selected pins.
- #define IO_PORT_STRENGTH_STRONG 0xff

  Bit mask to set strong drive strength for all selected pins.
- #define IO_PORT_DRAIN_DISABLE 0

  Bit mask to disable drain mode for all selected pins.
- #define IO_PORT_DRAIN_ENABLE 0xff

  Bit mask to enable drain mode for all selected pins.
- #define IO_PORT_OUT_INIT(port, pins, strength, drain, on, set)

  Initializes a set of port pins as outputs.
- #define IO_PORT_OUT_SET(port, pins, on, set)

  Set a set of output port pins.

13.6.1 Detailed Description

This group contains macros and defines to initialize port pins for output and safely set output states.

13.6.2 Macro Definition Documentation

13.6.2.1 IO_PORT_DRAIN_DISABLE

```
#define IO_PORT_DRAIN_DISABLE 0
```

Bit mask to disable drain mode for all selected pins.

13.6.2.2 IO_PORT_DRAIN_ENABLE

```
#define IO_PORT_DRAIN_ENABLE 0xff
```

Bit mask to enable drain mode for all selected pins.

13.6.2.3 IO_PORT_OUT_INIT

```
#define IO_PORT_OUT_INIT(
              port,
              pins,
              strength,
              drain,
              on,
              set )
```

Value:

```
{ \
        port##_DIR |= pins; \
        SFR_PAGE(_pp3, noSST); \
        port##_OD &= (drain) | ~(pins); \
        port##_OD |= (drain) & (pins); \
        port##_DS &= (strength) | ~(pins); \
        port##_DS |= (strength) & (pins); \
        SFR_PAGE(_pp0, noSST); \
        port##_DATA &= ((set) ^ ~(on)) | ~(pins); \
        port##_DATA |= ((set) ^ ~(on)) & (pins); \
}
```

Initializes a set of port pins as outputs.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

port	The parallel port to configure
pins	A bit mask of the pins to select
strength	A bit mask of pins with strong drive strength
drain	A bit mask of pins that only drive GND
on	A bit mask of pins that defines the states which represent on

See also

IO_PORT_ON_GND
IO_PORT_ON_HIGH

Parameters

set	Initial logical values for the defined outputs

13.6.2.4 IO_PORT_OUT_SET

```
#define IO_PORT_OUT_SET(
              port,
              pins,
```

```
                on,
                set )
```

Value:

```
{\
        port##_DATA &= ((set) ^ ~(on)) | ~(pins); \
        port##_DATA |= ((set) ^ ~(on)) & (pins); \
}
```

Set a set of output port pins.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

port	The parallel port to set
pins	A bit mask of the pins to select
on	A bit mask of pins that defines the states which represent on

See also

IO_PORT_ON_GND
IO_PORT_ON_HIGH

Parameters

set	Set logical values for the defined outputs

13.6.2.5 IO_PORT_STRENGTH_STRONG

```
#define IO_PORT_STRENGTH_STRONG 0xff
```

Bit mask to set strong drive strength for all selected pins.

13.6.2.6 IO_PORT_STRENGTH_WEAK

```
#define IO_PORT_STRENGTH_WEAK 0
```

Bit mask to set weak drive strength for all selected pins.

13.7 I/O Port Pull-Up/-Down Setup

This group contains macros and defines to initialize the pull-up/-down devices of port pins.

Macros

- #define IO_PORT_PULL_DISABLE 0

  Bit mask to disable pull up/down for all selected pins.
- #define IO_PORT_PULL_ENABLE 0xff

  Bit mask to enable pull up/down for all selected pins.
- #define IO_PORT_PULL_DOWN 0

  Bit mask to select pull down for all selected pins.
- #define IO_PORT_PULL_UP 0xff

  Bit mask to select pull up for all selected pins.
- #define IO_PORT_PULL_INIT(port, pins, pull, dir)

  Sets the pull-up/-down properties of port pins.

13.7.1 Detailed Description

This group contains macros and defines to initialize the pull-up/-down devices of port pins.

13.7.2 Macro Definition Documentation

13.7.2.1 IO_PORT_PULL_DISABLE

```
#define IO_PORT_PULL_DISABLE 0
```

Bit mask to disable pull up/down for all selected pins.

13.7.2.2 IO_PORT_PULL_DOWN

```
#define IO_PORT_PULL_DOWN 0
```

Bit mask to select pull down for all selected pins.

13.7.2.3 IO_PORT_PULL_ENABLE

```
#define IO_PORT_PULL_ENABLE 0xff
```

Bit mask to enable pull up/down for all selected pins.

13.7.2.4 IO_PORT_PULL_INIT

```
#define IO_PORT_PULL_INIT(
              port,
              pins,
              pull,
              dir )
```

Value:

```
{ \
        SFR_PAGE(_pp1, noSST); \
        port##_PUDSEL &= (dir) | ~(pins); \
        port##_PUDSEL |= (dir) & (pins); \
        port##_PUDEN &= (pull) | ~(pins); \
        port##_PUDEN |= (pull) & (pins); \
        SFR_PAGE(_pp0, noSST); \
}
```

Sets the pull-up/-down properties of port pins.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

port	The parallel port to configure
pins	A bit mask of the pins to select
pull	A bit mask of pins to activate the internal pull up/down device for
dir	A bit mask of pins to set the pull direction

13.7.2.5 IO_PORT_PULL_UP

```
#define IO_PORT_PULL_UP 0xff
```

Bit mask to select pull up for all selected pins.

13.8 Variable Access

This group specifies macros to access bits of a variable.

Macros

- #define IO_VAR_SET(var, bits, on, set)

  Set a set of variable bits.
- #define IO_VAR_GET(var, bits, on)

  Evaluates to a bit mask of logical states of a variable.

13.8.1 Detailed Description

This group specifies macros to access bits of a variable.

Their value lies in the seperation of encoded `on` state and logical `on` (1), as well as the safe bit masking.

13.8.2 Macro Definition Documentation

13.8.2.1 IO_VAR_GET

```
#define IO_VAR_GET(
              var,
              bits,
              on )
```

Value:

```
( \
        ((var) ^ ~(on)) & (bits) \
)
```

Evaluates to a bit mask of logical states of a variable.

Parameters

var	The variable to access
bits	A bit mask of the bits to select
on	A bit mask that defines the states which represent true

13.8.2.2 IO_VAR_SET

```
#define IO_VAR_SET(
              var,
              bits,
```

```
                on,
                set )
```

Value:

```
{\
        (var) &= ((set) ^ ~(on)) | ~(bits); \
        (var) |= ((set) ^ ~(on)) & (bits); \
}
```

Set a set of variable bits.

Parameters

var	The variable to set
bits	A bit mask of the bits to select
on	A bit mask that defines the states which represent true
set	Set logical values for the defined bits

13.9 Pulse Width Detection Units

This group of defines is used to select return format of hsk_pwc_channel_getValue().

Modules

- Pulse Width Times

  The defines are for returning average pulse width.
- Pulse Frequencies

  These defines are for returning average frequencies.
- Pulse Duty Times

  These defines are used for returning the duty time of the latest pulse.

Macros

- #define PWC_UNIT_SUM_RAW 0

  Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

13.9.1 Detailed Description

This group of defines is used to select return format of hsk_pwc_channel_getValue().

13.9.2 Macro Definition Documentation

13.9.2.1 PWC_UNIT_SUM_RAW

```
#define PWC_UNIT_SUM_RAW 0
```

Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

This is the sum of the buffered values, not the average.

Use this if precision is of the utmost importance.

13.10 Pulse Width Times

The defines are for returning average pulse width.

Macros

- #define PWC_UNIT_WIDTH_RAW 1

  Average of buffered pulse widths in multiples of $1/48 * 10^{-6}s$.
- #define PWC_UNIT_WIDTH_NS 2

  Average of buffered pulse widths in multiples of $10^{-9}s$.
- #define PWC_UNIT_WIDTH_US 3

  Average of buffered pulse widths in multiples of $10^{-6}s$.
- #define PWC_UNIT_WIDTH_MS 4

  Average of buffered pulse widths in multiples of $10^{-3}s$.

13.10.1 Detailed Description

The defines are for returning average pulse width.

13.10.2 Macro Definition Documentation

13.10.2.1 PWC_UNIT_WIDTH_MS

```
#define PWC_UNIT_WIDTH_MS 4
```

Average of buffered pulse widths in multiples of $10^{-3}s$.

13.10.2.2 PWC_UNIT_WIDTH_NS

```
#define PWC_UNIT_WIDTH_NS 2
```

Average of buffered pulse widths in multiples of $10^{-9}s$.

13.10.2.3 PWC_UNIT_WIDTH_RAW

```
#define PWC_UNIT_WIDTH_RAW 1
```

Average of buffered pulse widths in multiples of $1/48 * 10^{-6}s$.

13.10.2.4 PWC_UNIT_WIDTH_US

```
#define PWC_UNIT_WIDTH_US 3
```

Average of buffered pulse widths in multiples of $10^{-6}s$.

13.11 Pulse Frequencies

These defines are for returning average frequencies.

Macros

- #define PWC_UNIT_FREQ_S 5

  Average frequency of buffered pulses in multiples of $1/s$.
- #define PWC_UNIT_FREQ_M 6

  Average frequency of buffered pulses in multiples of $1/m$.
- #define PWC_UNIT_FREQ_H 7

  Average frequency of buffered pulses in multiples of $1/h$.

13.11.1 Detailed Description

These defines are for returning average frequencies.

13.11.2 Macro Definition Documentation

13.11.2.1 PWC_UNIT_FREQ_H

```
#define PWC_UNIT_FREQ_H 7
```

Average frequency of buffered pulses in multiples of $1/h$.

To prevent overflow issues this value is always a multiple of the number of averaged values $*$ 60.

This is just a convenience feature for quick testing, it is possible to achieve much better precision if the use case is known.

13.11.2.2 PWC_UNIT_FREQ_M

```
#define PWC_UNIT_FREQ_M 6
```

Average frequency of buffered pulses in multiples of $1/m$.

To prevent overflow issues this value is always a multiple of the number of averaged values.

13.11.2.3 PWC_UNIT_FREQ_S

```
#define PWC_UNIT_FREQ_S 5
```

Average frequency of buffered pulses in multiples of $1/s$.

13.12 Pulse Duty Times

These defines are used for returning the duty time of the latest pulse.

Macros

- #define PWC_UNIT_DUTYH_RAW 8
  Latest high pulse in multiples of $1/48 * 10^{-6} s$.
- #define PWC_UNIT_DUTYH_NS 9
  Latest high pulse in multiples of $1 * 10^{-9} s$.
- #define PWC_UNIT_DUTYH_US 10
  Latest high pulse in multiples of $1 * 10^{-6} s$.
- #define PWC_UNIT_DUTYH_MS 11
  Latest high pulse in multiples of $1 * 10^{-3} s$.
- #define PWC_UNIT_DUTYL_RAW 12
  Latest low pulse in multiples of $1/48 * 10^{-6} s$.
- #define PWC_UNIT_DUTYL_NS 13
  Latest low pulse in multiples of $1 * 10^{-9} s$.
- #define PWC_UNIT_DUTYL_US 14
  Latest low pulse in multiples of $1 * 10^{-6} s$.
- #define PWC_UNIT_DUTYL_MS 15
  Latest low pulse in multiples of $1 * 10^{-3} s$.

13.12.1 Detailed Description

These defines are used for returning the duty time of the latest pulse.

In order to use this return type, the channel buffer must hold at least 2 values. I.e. the averageOver argument of hsk_pwc_port_open() must be 2 or greater (there is no benefit to a value above 2).

To produce correct results the channel must also be in edge mode PWC_EDGE_BOTH.

13.12.2 Macro Definition Documentation

13.12.2.1 PWC_UNIT_DUTYH_MS

```
#define PWC_UNIT_DUTYH_MS 11
```

Latest high pulse in multiples of $1 * 10^{-3} s$.

13.12.2.2 PWC_UNIT_DUTYH_NS

```
#define PWC_UNIT_DUTYH_NS 9
```

Latest high pulse in multiples of $1 * 10^{-9} s$.

13.12.2.3 PWC_UNIT_DUTYH_RAW

```
#define PWC_UNIT_DUTYH_RAW 8
```

Latest high pulse in multiples of $1/48 * 10^{-6} s$.

13.12.2.4 PWC_UNIT_DUTYH_US

```
#define PWC_UNIT_DUTYH_US 10
```

Latest high pulse in multiples of $1 * 10^{-6} s$.

13.12.2.5 PWC_UNIT_DUTYL_MS

```
#define PWC_UNIT_DUTYL_MS 15
```

Latest low pulse in multiples of $1 * 10^{-3} s$.

13.12.2.6 PWC_UNIT_DUTYL_NS

```
#define PWC_UNIT_DUTYL_NS 13
```

Latest low pulse in multiples of $1 * 10^{-9} s$.

13.12.2.7 PWC_UNIT_DUTYL_RAW

```
#define PWC_UNIT_DUTYL_RAW 12
```

Latest low pulse in multiples of $1/48 * 10^{-6} s$.

13.12.2.8 PWC_UNIT_DUTYL_US

```
#define PWC_UNIT_DUTYL_US 14
```

Latest low pulse in multiples of $1 * 10^{-6} s$.

13.13 SSC I/O Ports

Used to create an I/O Port configuration, by unifying one of the SSC_MRST_P∗ with a SSC_MTSR_P∗ and a SSC_SCLK_P∗ ports.

Macros

- #define SSC_MRST_P05 1

  Master mode RX, slave mode TX port P0.5.
- #define SSC_MRST_P14 0

  Master mode RX, slave mode TX port P1.4.
- #define SSC_MRST_P15 2

  Master mode RX, slave mode TX port P1.5.
- #define SSC_MTSR_P04 (1 << 2)

  Master mode TX, slave mode RX port P0.4.
- #define SSC_MTSR_P13 (0 << 2)

  Master mode TX, slave mode RX port P1.3.
- #define SSC_MTSR_P14 (2 << 2)

  Master mode TX, slave mode RX port P1.4.
- #define SSC_SCLK_P03 (1 << 4)

  Synchronous clock port P0.3.
- #define SSC_SCLK_P12 (0 << 4)

  Synchronous clock port P1.2.
- #define SSC_SCLK_P13 (2 << 4)

  Synchronous clock port P1.3.

13.13.1 Detailed Description

Used to create an I/O Port configuration, by unifying one of the SSC_MRST_P∗ with a SSC_MTSR_P∗ and a SSC_SCLK_P∗ ports.

E.g.:

```
SSC_MRST_P05 | SSC_MTSR_P4 | SSC_SCLK_P03.
```

The ports have the following functions:

Type	Master Mode	Slave Mode
MRST	RX port	TX port
MTSR	TX port	RX port
SCLK	TX clock	RX clock

13.13.2 Macro Definition Documentation

13.13.2.1 SSC_MRST_P05

```
#define SSC_MRST_P05 1
```

Master mode RX, slave mode TX port P0.5.

13.13.2.2 SSC_MRST_P14

```
#define SSC_MRST_P14 0
```

Master mode RX, slave mode TX port P1.4.

13.13.2.3 SSC_MRST_P15

```
#define SSC_MRST_P15 2
```

Master mode RX, slave mode TX port P1.5.

13.13.2.4 SSC_MTSR_P04

```
#define SSC_MTSR_P04 (1 << 2)
```

Master mode TX, slave mode RX port P0.4.

13.13.2.5 SSC_MTSR_P13

```
#define SSC_MTSR_P13 (0 << 2)
```

Master mode TX, slave mode RX port P1.3.

13.13.2.6 SSC_MTSR_P14

```
#define SSC_MTSR_P14 (2 << 2)
```

Master mode TX, slave mode RX port P1.4.

13.13.2.7 SSC_SCLK_P03

```
#define SSC_SCLK_P03 (1 << 4)
```

Synchronous clock port P0.3.

13.13.2.8 SSC_SCLK_P12

```
#define SSC_SCLK_P12 (0 << 4)
```

Synchronous clock port P1.2.

13.13.2.9 SSC_SCLK_P13

```
#define SSC_SCLK_P13 (2 << 4)
```

Synchronous clock port P1.3.

Chapter 14

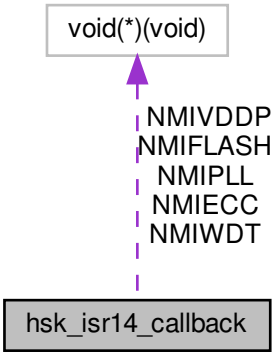
Data Structure Documentation

14.1 hsk_isr14_callback Struct Reference

Shared non-maskable interrupt routine.

`#include <hsk_isr.h>`

Collaboration diagram for hsk_isr14_callback:



Data Fields

- void(∗ NMIWDT)(void)
  Function to be called back when the NMIWDT interrupt event is triggered.
- void(∗ NMIPLL)(void)
  Function to be called back when the NMIPLL interrupt event is triggered.
- void(∗ NMIFLASH)(void)
  Function to be called back when the NMIFLASH interrupt event is triggered.
- void(∗ NMIVDDP)(void)
  Function to be called back when the NMIVDDP interrupt event is triggered.
- void(∗ NMIECC)(void)
  Function to be called back when the NMIECC interrupt event is triggered.

14.1.1 Detailed Description

Shared non-maskable interrupt routine.

This interrupt has the following sources:

- Watchdog Timer NMI (NMIWDT)
- PLL NMI (NMIPLL)
- Flash Timer NMI (NMIFLASH)
- VDDP Prewarning NMI (NMIVDDP)
- Flash ECC NMI (NMIECC)

14.1.2 Field Documentation

14.1.2.1 NMIECC

```
void( * hsk_isr14_callback::NMIECC) (void)
```

Function to be called back when the NMIECC interrupt event is triggered.

14.1.2.2 NMIFLASH

```
void( * hsk_isr14_callback::NMIFLASH) (void)
```

Function to be called back when the NMIFLASH interrupt event is triggered.

14.1.2.3 NMIPLL

```
void( * hsk_isr14_callback::NMIPLL) (void)
```

Function to be called back when the NMIPLL interrupt event is triggered.

14.1.2.4 NMIVDDP

```
void( * hsk_isr14_callback::NMIVDDP) (void)
```

Function to be called back when the NMIVDDP interrupt event is triggered.

14.1.2.5 NMIWDT

```
void( * hsk_isr14_callback::NMIWDT) (void)
```

Function to be called back when the NMIWDT interrupt event is triggered.

The documentation for this struct was generated from the following file:

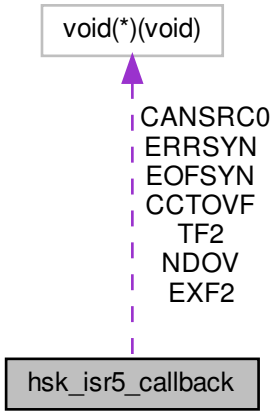
- hsk_isr/hsk_isr.h

14.2 hsk_isr5_callback Struct Reference

Shared interrupt 5 routine.

`#include <hsk_isr.h>`

Collaboration diagram for hsk_isr5_callback:



Data Fields

- void(∗ TF2)(void)

  Function to be called back when the TF2 interrupt event is triggered.
- void(∗ EXF2)(void)

  Function to be called back when the EXF2 interrupt event is triggered.
- void(∗ CCTOVF)(void)

  Function to be called back when the CCTOVF interrupt event is triggered.
- void(∗ NDOV)(void)

  Function to be called back when the NDOV interrupt event is triggered.
- void(∗ EOFSYN)(void)

  Function to be called back when the EOFSYN interrupt event is triggered.
- void(∗ ERRSYN)(void)

  Function to be called back when the ERRSYN interrupt event is triggered.
- void(∗ CANSRC0)(void)

  Function to be called back when the CANSRC0 interrupt event is triggered.

14.2.1 Detailed Description

Shared interrupt 5 routine.

Activate the interrupt by setting ET2 = 1.

This interrupt has the following sources:

- Timer 2 Overflow (TF2)
- Timer 2 External Event (EXF2)
- T2CCU CCT Overflow (CCTOVF)
- Normal Divider Overflow (NDOV)
- End of Syn Byte (EOFSYN)
- Syn Byte Error (ERRSYN)
- CAN Interrupt 0 (CANSRC0)

14.2.2 Field Documentation

14.2.2.1 CANSRC0

```
void( * hsk_isr5_callback::CANSRC0) (void)
```

Function to be called back when the CANSRC0 interrupt event is triggered.

14.2.2.2 CCTOVF

```
void( * hsk_isr5_callback::CCTOVF) (void)
```

Function to be called back when the CCTOVF interrupt event is triggered.

14.2.2.3 EOFSYN

```
void( * hsk_isr5_callback::EOFSYN) (void)
```

Function to be called back when the EOFSYN interrupt event is triggered.

14.2.2.4 ERRSYN

```
void( * hsk_isr5_callback::ERRSYN) (void)
```

Function to be called back when the ERRSYN interrupt event is triggered.

14.2.2.5 EXF2

```
void( * hsk_isr5_callback::EXF2) (void)
```

Function to be called back when the EXF2 interrupt event is triggered.

14.2.2.6 NDOV

```
void( * hsk_isr5_callback::NDOV) (void)
```

Function to be called back when the NDOV interrupt event is triggered.

14.2.2.7 TF2

```
void( * hsk_isr5_callback::TF2) (void)
```

Function to be called back when the TF2 interrupt event is triggered.

The documentation for this struct was generated from the following file:

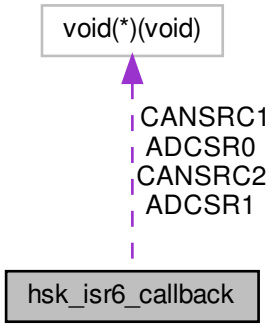
- hsk_isr/hsk_isr.h

14.3 hsk_isr6_callback Struct Reference

Shared interrupt 6 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr6_callback:



Data Fields

- void(∗ CANSRC1)(void)

  Function to be called back when the CANSRC1 interrupt event is triggered.
- void(∗ CANSRC2)(void)

  Function to be called back when the CANSRC2 interrupt event is triggered.
- void(∗ ADCSR0)(void)

  Function to be called back when the ADCSR0 interrupt event is triggered.
- void(∗ ADCSR1)(void)

  Function to be called back when the ADCSR1 interrupt event is triggered.

14.3.1 Detailed Description

Shared interrupt 6 routine.

Activate the interrupt by setting EADC = 1.

This interrupt has the following sources:

- CANSRC1
- CANSRC2
- ADCSR0
- ADCSR1

14.3.2 Field Documentation

14.3.2.1 ADCSR0

```
void( * hsk_isr6_callback::ADCSR0) (void)
```

Function to be called back when the ADCSR0 interrupt event is triggered.

14.3.2.2 ADCSR1

```
void( * hsk_isr6_callback::ADCSR1) (void)
```

Function to be called back when the ADCSR1 interrupt event is triggered.

14.3.2.3 CANSRC1

```
void( * hsk_isr6_callback::CANSRC1) (void)
```

Function to be called back when the CANSRC1 interrupt event is triggered.

14.3.2.4 CANSRC2

```
void( * hsk_isr6_callback::CANSRC2) (void)
```

Function to be called back when the CANSRC2 interrupt event is triggered.

The documentation for this struct was generated from the following file:

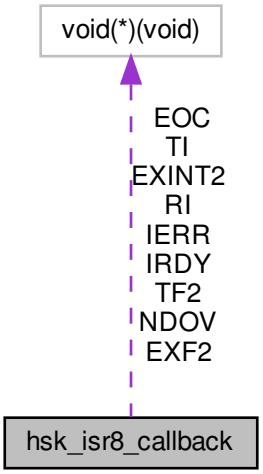
- hsk_isr/hsk_isr.h

14.4 hsk_isr8_callback Struct Reference

Shared interrupt 8 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr8_callback:



Data Fields

- void(∗ EXINT2)(void)

  Function to be called back when the EXINT2 interrupt event is triggered.
- void(∗ RI)(void)

  Function to be called back when the RI interrupt event is triggered.
- void(∗ TI)(void)

  Function to be called back when the TI interrupt event is triggered.
- void(∗ TF2)(void)

  Function to be called back when the TF2 interrupt event is triggered.
- void(∗ EXF2)(void)

Function to be called back when the EXF2 interrupt event is triggered.

- void(∗ NDOV)(void)

  Function to be called back when the NDOV interrupt event is triggered.

- void(∗ EOC)(void)

  Function to be called back when the EOC interrupt event is triggered.

- void(∗ IRDY)(void)

  Function to be called back when the IRDY interrupt event is triggered.

- void(∗ IERR)(void)

  Function to be called back when the IERR interrupt event is triggered.

14.4.1 Detailed Description

Shared interrupt 8 routine.

Activate the interrupt by setting EX2 = 1.

This interrupt has the following sources:

- External Interrupt 2 (EXINT2)
- UART1 (RI)
- UART1 (TI)
- Timer 21 Overflow (TF2)
- T21EX (EXF2)
- UART1 Fractional Divider (Normal Divider Overflow) (NDOV)
- CORDIC (EOC)
- MDU Result Ready (IRDY)
- MDU Error (IERR)

14.4.2 Field Documentation

14.4.2.1 EOC

```
void( * hsk_isr8_callback::EOC) (void)
```

Function to be called back when the EOC interrupt event is triggered.

14.4.2.2 EXF2

```
void( * hsk_isr8_callback::EXF2) (void)
```

Function to be called back when the EXF2 interrupt event is triggered.

14.4.2.3 EXINT2

```
void( * hsk_isr8_callback::EXINT2) (void)
```

Function to be called back when the EXINT2 interrupt event is triggered.

14.4.2.4 IERR

```
void( * hsk_isr8_callback::IERR) (void)
```

Function to be called back when the IERR interrupt event is triggered.

14.4.2.5 IRDY

```
void( * hsk_isr8_callback::IRDY) (void)
```

Function to be called back when the IRDY interrupt event is triggered.

14.4.2.6 NDOV

```
void( * hsk_isr8_callback::NDOV) (void)
```

Function to be called back when the NDOV interrupt event is triggered.

14.4.2.7 RI

```
void( * hsk_isr8_callback::RI) (void)
```

Function to be called back when the RI interrupt event is triggered.

14.4.2.8 TF2

```
void( * hsk_isr8_callback::TF2) (void)
```

Function to be called back when the TF2 interrupt event is triggered.

14.4.2.9 TI

```
void( * hsk_isr8_callback::TI) (void)
```

Function to be called back when the TI interrupt event is triggered.

The documentation for this struct was generated from the following file:

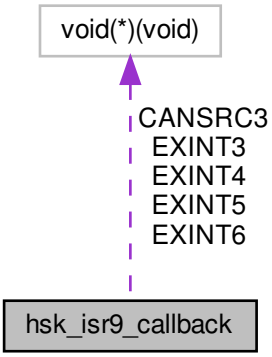
- hsk_isr/hsk_isr.h

14.5 hsk_isr9_callback Struct Reference

Shared interrupt 9 routine.

`#include <hsk_isr.h>`

Collaboration diagram for hsk_isr9_callback:



Data Fields

- void(∗ EXINT3)(void)
  Function to be called back when the EXINT3/T2CC0 interrupt event is triggered.
- void(∗ EXINT4)(void)
  Function to be called back when the EXINT4/T2CC1 interrupt event is triggered.
- void(∗ EXINT5)(void)
  Function to be called back when the EXINT5/T2CC2 interrupt event is triggered.
- void(∗ EXINT6)(void)
  Function to be called back when the EXINT6/T2CC3 interrupt event is triggered.
- void(∗ CANSRC3)(void)
  Function to be called back when the CANSRC3 interrupt event is triggered.

14.5.1 Detailed Description

Shared interrupt 9 routine.

Activate the interrupt by setting EXM = 1.

This interrupt has the following sources:

- EXINT3/T2CC0
- EXINT4/T2CC1
- EXINT5/T2CC2
- EXINT6/T2CC3
- CANSRC2

14.5.2 Field Documentation

14.5.2.1 CANSRC3

```
void( * hsk_isr9_callback::CANSRC3) (void)
```

Function to be called back when the CANSRC3 interrupt event is triggered.

14.5.2.2 EXINT3

```
void( * hsk_isr9_callback::EXINT3) (void)
```

Function to be called back when the EXINT3/T2CC0 interrupt event is triggered.

14.5.2.3 EXINT4

```
void( * hsk_isr9_callback::EXINT4) (void)
```

Function to be called back when the EXINT4/T2CC1 interrupt event is triggered.

14.5.2.4 EXINT5

```
void( * hsk_isr9_callback::EXINT5) (void)
```

Function to be called back when the EXINT5/T2CC2 interrupt event is triggered.

14.5.2.5 EXINT6

```
void( * hsk_isr9_callback::EXINT6) (void)
```

Function to be called back when the EXINT6/T2CC3 interrupt event is triggered.

The documentation for this struct was generated from the following file:

- hsk_isr/hsk_isr.h

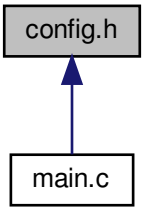
Chapter 15

File Documentation

15.1 config.h File Reference

Configuration for the Infineon XC800 Starter Kit.

This graph shows which files directly or indirectly include this file:



Macros

- #define CLK 8000000UL

  The external oscilator clock frequency.
- #define CAN0_BAUD 1000000

  The CAN0 baud rate in bits/s.
- #define CAN1_BAUD 1000000

  The CAN1 baud rate in bits/s.
- #define CAN0_IO CAN0_IO_P10_P11

  The CAN0 IO pin configuration RX P1.0 TX P1.1.
- #define CAN1_IO CAN1_IO_P14_P13

  The CAN1 IO pin configuration RX P1.4 TX P1.3.

15.1.1 Detailed Description

Configuration for the Infineon XC800 Starter Kit.

Author

kami

15.1.2 Macro Definition Documentation

15.1.2.1 CAN0_BAUD

```
#define CAN0_BAUD 1000000
```

The CAN0 baud rate in bits/s.

15.1.2.2 CAN0_IO

```
#define CAN0_IO CAN0_IO_P10_P11
```

The CAN0 IO pin configuration RX P1.0 TX P1.1.

15.1.2.3 CAN1_BAUD

```
#define CAN1_BAUD 1000000
```

The CAN1 baud rate in bits/s.

15.1.2.4 CAN1_IO

```
#define CAN1_IO CAN1_IO_P14_P13
```

The CAN1 IO pin configuration RX P1.4 TX P1.3.

15.1.2.5 CLK

```
#define CLK 8000000UL
```

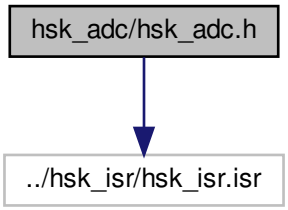
The external oscilator clock frequency.

15.2 hsk_adc/hsk_adc.h File Reference

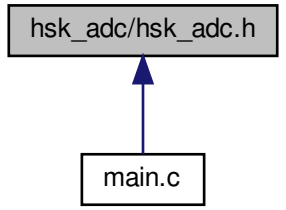
HSK Analog Digital Conversion headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_adc.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ADC_RESOLUTION_10 0

  10 bit ADC resolution.
- #define ADC_RESOLUTION_8 1

  8 bit ADC resultion.
- #define hsk_adc_open hsk_adc_open10

  Backwards compatibility hack.
- #define hsk_adc_warmup hsk_adc_warmup10

  Backwards compatibility hack.

Typedefs

- typedef ubyte hsk_adc_channel

  Typedef for ADC channel ids.

Functions

- void hsk_adc_init (ubyte resolution, uword convTime)

  Initialize the AD conversion.
- void hsk_adc_enable (void)

  Turns on ADC conversion, if previously deactivated.
- void hsk_adc_disable (void)

  Turns off ADC conversion unit to converse power.
- void hsk_adc_open10 (const hsk_adc_channel channel, uword ∗const target)

  Open the given ADC channel in 10 bit mode.
- void hsk_adc_open8 (const hsk_adc_channel channel, ubyte ∗const target)

  Open the given ADC channel in 8 bit mode.
- void hsk_adc_close (const hsk_adc_channel channel)

  Close the given ADC channel.
- bool hsk_adc_service (void)

  A maintenance function that takes care of keeping AD conversions going.
- bool hsk_adc_request (const hsk_adc_channel channel)

  Requests an ADC for a specific channel.
- void hsk_adc_warmup10 (void)

  Warm up 10 bit AD conversion.

15.2.1 Detailed Description

HSK Analog Digital Conversion headers.

This library provides access to all 8 ADC channels. Each channel can be provided with a pointer. Every completed conversion is written to the address provided by the pointer. The target memory can be protected for read access by msking the interrupts with EADC.

The conversion time can be freely configured in a wide range. Even short conversion times like 5µs yield good precission.

In order to keep the conversion going a service function hsk_adc_service() has to be called on a regular basis. This prevents locking up of the CPU due to an overload of interrupts, the ADC module can provide a new conversion result every 30 clock cycles.

Making the hsk_adc_service() call only as often as needed reduces the drain on the analogue input and reduces flickering.

Alternatively hsk_adc_request() can be used to request single just in time conversions.

Author

kami

15.2.2 Macro Definition Documentation

15.2.2.1 ADC_RESOLUTION_10

```
#define ADC_RESOLUTION_10 0
```

10 bit ADC resolution.

15.2.2.2 ADC_RESOLUTION_8

```
#define ADC_RESOLUTION_8 1
```

8 bit ADC resultion.

15.2.2.3 hsk_adc_open

```
#define hsk_adc_open hsk_adc_open10
```

Backwards compatibility hack.

Deprecated Use hsk_adc_open10() or hsk_adc_open8() as appropriate

15.2.2.4 hsk_adc_warmup

```
#define hsk_adc_warmup hsk_adc_warmup10
```

Backwards compatibility hack.

Deprecated Use hsk_adc_warmup10()

15.2.3 Typedef Documentation

15.2.3.1 hsk_adc_channel

```
typedef ubyte hsk_adc_channel
```

Typedef for ADC channel ids.

15.2.4 Function Documentation

15.2.4.1 hsk_adc_close()

```
void hsk_adc_close (
            const hsk_adc_channel channel )
```

Close the given ADC channel.

Stopp ADC if no more channels were left.

Parameters

channel	The channel id

15.2.4.2 hsk_adc_disable()

```
void hsk_adc_disable (
            void  )
```

Turns off ADC conversion unit to converse power.

15.2.4.3 hsk_adc_enable()

```
void hsk_adc_enable (
            void  )
```

Turns on ADC conversion, if previously deactivated.

15.2.4.4 hsk_adc_init()

```
void hsk_adc_init (
            ubyte resolution,
            uword convTime )
```

Initialize the AD conversion.

The shortest possible conversion time is 1.25µs, the longest is 714.75µs. The given value will be rounded down.

Note if hsk_adc_service() is not called in intervals shorter than convTime, there will be a waiting period between conversions. This prevents locking up of the controler with erratic interrupts.

There is a 4 entry queue, for starting conversions, so it suffices to average the interval below convTime.

All already open channels will be closed upon calling this function.

Parameters

resolution	The conversion resolution, any of ADC_RESOLUTION_∗
convTime	The desired conversion time in µs

15.2.4.5 hsk_adc_open10()

```
void hsk_adc_open10 (
            const hsk_adc_channel channel,
            uword *const target )
```

Open the given ADC channel in 10 bit mode.

Parameters

channel	The channel id
target	A pointer where to store conversion results

15.2.4.6 hsk_adc_open8()

```
void hsk_adc_open8 (
            const hsk_adc_channel channel,
            ubyte *const target )
```

Open the given ADC channel in 8 bit mode.

Parameters

channel	The channel id
target	A pointer where to store conversion results

15.2.4.7 hsk_adc_request()

```
bool hsk_adc_request (
            const hsk_adc_channel channel )
```

Requests an ADC for a specific channel.

This function is an alternative to hsk_adc_service(). Make requests in time before the updated value is required.

This function uses the same queue as hsk_adc_service(), if the queue is full it fails silently.

Parameters

channel	The channel id

Return values

0	The queue is full
1	A conversion request has been added to the queue

15.2.4.8 hsk_adc_service()

```
bool hsk_adc_service (
            void  )
```

A maintenance function that takes care of keeping AD conversions going.

This has to be called repeatedly.

There is a queue of up to 4 conversion jobs. One call of this function only adds one job to the queue.

Return values

0	No conversion request had been queued, either the queue is full or no channels have been configured
1	A conversion request has been added to the queue

15.2.4.9 hsk_adc_warmup10()

```
void hsk_adc_warmup10 (
            void  )
```

Warm up 10 bit AD conversion.

I.e. make sure all conversion targets have been initialized with a conversion result. This is a blocking function only intended for single use during the boot procedure.

This function will not terminate unless interrupts are enabled.

Note

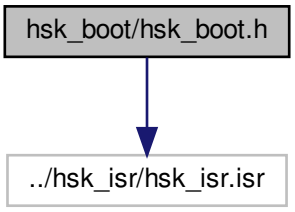
This function only works in 10 bit mode, because in 8 bit mode it is impossible to initialize targets with an invalid value.

15.3 hsk_boot/hsk_boot.h File Reference

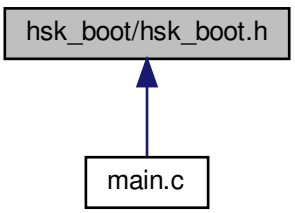
HSK Boot headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_boot.h:



This graph shows which files directly or indirectly include this file:



Functions

- void hsk_boot_extClock (const ulong clk)

  Switches to an external oscilator.

15.3.1 Detailed Description

HSK Boot headers.

This file contains the prototypes to put the µC into working condition.

Currently implemented:

- hsk_boot_extClock() Activates external clock input and sets up the PLL, this is important when communicating with other devices, the internal clock is not sufficiently precise

Linking this library also automatically causes the following boot actions:

- Deactivate all internal pullup devices
- Activate XDATA access
- Set the PDATA page to the first XDATA block

Author

kami

15.3.2 Function Documentation

15.3.2.1 hsk_boot_extClock()

```
void hsk_boot_extClock (
            const ulong clk )
```

Switches to an external oscilator.

This function requires xdata access.

The implemented process is named: "Select the External Oscillator as PLL input source"

The following is described in more detail in chapter 7.3 of the XC878 User Manual.

The XC878 can either use an internal 4MHz oscilator (default) or an external oscilator from 2 to 20MHz, normally referred to as FOSC. A phase-locked loop (PLL) converts it to a faster internal speed FSYS, 144MHz by default.

This implementation is currently limited to oscilators from 2MHz to 20MHz in 1MHz intervals.

The oscilator frequency is vital for external communication (e.g. CAN) and timer/counter speeds.

This implementation switches to an external clock ensuring that the PLL generates a 144MHz FSYS clock. The CLKREL divisor set to 6 generates the fast clock (FCLK) that runs at 48MHz. The remaining clocks, i.e. peripheral (PCLK), CPU (SCLK, CCLK), have a fixed divisor by 2, so they run at 24MHz.

After setting up the PLL, this function will register an ISR, that will attempt to reactivate the external oscillator in a PLL loss-of-clock event.

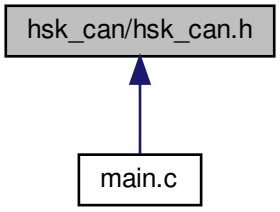
Parameters

clk	The frequency of the external oscilator in Hz.

15.4 hsk_can/hsk_can.h File Reference

HSK Controller Area Network headers.

This graph shows which files directly or indirectly include this file:



Macros

- #define CAN_ERROR 0xff

  Value returned by functions in case of an error.
- #define CAN0 0

  CAN node 0.
- #define CAN1 1

  CAN node 1.
- #define CAN0_IO_P10_P11 0

  CAN node 0 IO RX on P1.0, TX on P1.1.
- #define CAN0_IO_P16_P17 1

  CAN node 0 IO RX on P1.6, TX on P1.7.
- #define CAN0_IO_P34_P35 2

  CAN node 0 IO RX on P3.4, TX on P3.5.
- #define CAN0_IO_P40_P41 3

  CAN node 0 IO RX on P4.0, TX on P4.1.
- #define CAN1_IO_P01_P02 4

  CAN node 1 IO RX on P0.1, TX on P0.2.
- #define CAN1_IO_P14_P13 5

  CAN node 1 IO RX on P1.4, TX on P1.3.
- #define CAN1_IO_P32_P33 6

  CAN node 1 IO RX on P3.2, TX on P3.3.
- #define CAN_ENDIAN_INTEL 0

  Little endian signal encoding.
- #define CAN_ENDIAN_MOTOROLA 1

Big endian signal encoding.

- #define CAN_STATUS_LEC 0

  The Last Error Code field provides the error triggered by the last message on the bus.
- #define CAN_STATUS_TXOK 1

  Message Transmitted Successfully.
- #define CAN_STATUS_RXOK 2

  Message Received Successfully.
- #define CAN_STATUS_ALERT 3

  Alert Warning.
- #define CAN_STATUS_EWRN 4

  Error Warning Status.
- #define CAN_STATUS_BOFF 5

  Bus-off Status.

Typedefs

- typedef ubyte hsk_can_node

  CAN node identifiers.
- typedef ubyte hsk_can_msg

  CAN message object identifiers.
- typedef ubyte hsk_can_fifo

  CAN message FIFO identifiers.

Functions

- void hsk_can_init (const ubyte pins, const ulong baud)

  Setup CAN communication with the desired baud rate.
- void hsk_can_enable (const hsk_can_node node)

  Go live on the CAN bus.
- void hsk_can_disable (const hsk_can_node node)

  Disable a CAN node.
- ubyte hsk_can_status (const hsk_can_node node, const ubyte field)

  Returns a status field of a CAN node.
- hsk_can_msg hsk_can_msg_create (const ulong id, const bool extended, const ubyte dlc)

  Creates a new CAN message.
- ubyte hsk_can_msg_connect (const hsk_can_msg msg, const hsk_can_node node)

  Connect a message object to a CAN node.
- ubyte hsk_can_msg_disconnect (const hsk_can_msg msg)

  Disconnect a CAN message object from its CAN node.
- ubyte hsk_can_msg_delete (const hsk_can_msg msg)

  Delete a CAN message object.
- void hsk_can_msg_getData (const hsk_can_msg msg, ubyte ∗const msgdata)

  Gets the current data in the CAN message.
- void hsk_can_msg_setData (const hsk_can_msg msg, const ubyte ∗const msgdata)

  Sets the current data in the CAN message.
- void hsk_can_msg_send (const hsk_can_msg msg)

  Request transmission of a message.
- bool hsk_can_msg_sent (const hsk_can_msg msg)

  Return whether the message was successfully sent between this and the previous call of this method.

- void hsk_can_msg_receive (const hsk_can_msg msg)

  Return the message into RX mode after sending a message.
- bool hsk_can_msg_updated (const hsk_can_msg msg)

  Return whether the message was updated via CAN bus between this call and the previous call of this method.
- hsk_can_fifo hsk_can_fifo_create (ubyte size)

  Creates a message FIFO.
- void hsk_can_fifo_setupRx (hsk_can_fifo fifo, const ulong id, const bool extended, const ubyte dlc)

  Set the FIFO up for receiving messages.
- void hsk_can_fifo_setRxMask (const hsk_can_fifo fifo, ulong msk)

  Changes the ID matching mask of an RX FIFO.
- ubyte hsk_can_fifo_connect (const hsk_can_fifo fifo, const hsk_can_node node)

  Connect a FIFO to a CAN node.
- ubyte hsk_can_fifo_disconnect (const hsk_can_fifo fifo)

  Disconnect a FIFO from its CAN node.
- ubyte hsk_can_fifo_delete (const hsk_can_fifo fifo)

  Delete a FIFO.
- void hsk_can_fifo_next (const hsk_can_fifo fifo)

  Select the next FIFO entry.
- ulong hsk_can_fifo_getId (const hsk_can_fifo fifo)

  Returns the CAN ID of the selected FIFO entry.
- bool hsk_can_fifo_updated (const hsk_can_fifo fifo)

  Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.
- void hsk_can_fifo_getData (const hsk_can_fifo fifo, ubyte *const msgdata)

  Gets the data from the currently selected FIFO entry.
- void hsk_can_data_setSignal (ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount, const ulong value)

  Sets a signal value in a data field.
- ulong hsk_can_data_getSignal (const ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount)

  Get a signal value from a data field.

15.4.1 Detailed Description

HSK Controller Area Network headers.

This file contains the function prototypes to initialize and engage in CAN communication over the builtin CAN nodes 0 and 1.

Author

kami

15.4.2 CAN Message/Signal Tuples

The recommended way to use messages and signals is not to specify them inline, but to provide defines with a set of parameters.

These tupples should follow the following pattern:

```
#define MSG_<MSGNAME>    <id>, <extended>, <dlc>
#define SIG_<SIGNAME>    <motorola>, <signed>, <bitPos>, <bitCount>
```

The symbols have the following meaning:

- MSGNAME: The name of the message in capitals, e.g. AFB_CHANNELS
- id: The CAN id of the message, e.g. 0x403
- extended: Whether the CAN ID is extended or not, e.g. 0 for a regular ID
- dlc: The data length count of the message, e.g. 3
- SIGNAME: The name of the signal in capitals, e.g. AFB_CHANNEL0_CURRENT
- motorola: Whether the signal is in big endian (Motorola) format
- signed: Whether the signal is signed
- bitPos: The starting bit of the signal, e.g. 0
- bitCount: The length of the signal in bits, e.g. 10

Tuples using the specified format can directly be used as parameters for several functions in the library.

15.4.3 CAN Node Management

There are 7 port pairs available for CAN communication, check the CANn_IO_$*$ defines. Four for the node CAN0 and three for CAN1.

15.4.4 Message Object Management

The MultiCAN module offers up to 32 message objects. New messages are set up for receiving messages. Message object can be switched from RX to TX mode and back with the hsk_can_msg_send() and hsk_can_msg_receive() functions.

15.4.5 FIFOs

FIFOs are the weapon of choice when dealing with large numbers of individual messages or when receving multiplexed data. In most use cases only the latest version of a message is relevant and FIFOs are not required. But messages containing multiplexed signals may contain critical signals that would be overwritten by a message with the same ID, but a different multiplexor.

If more message IDs than available message objects are used to send and/or receive data, there is no choice but to use a FIFO.

Currently only RX FIFOs are supported.

A FIFO can act as a buffer the CAN module can store message data in until it can be dealt with. The following example illustrates how to read from a FIFO:

```
if (hsk_can_fifo_updated(fifo0)) {
        hsk_can_fifo_getData(fifo0, data0);
        hsk_can_fifo_next(fifo0);
        select = hsk_can_data_getSignal(data0, SIG_MULTIPLEXOR);
        [...]
}
```

When using a mask to accept several messages checking the ID becomes necessary:

```
if (hsk_can_fifo_updated(fifo0)) {
        switch (hsk_can_fifo_getId()) {
        case MSG_0_ID:
                hsk_can_fifo_getData(fifo0, data0);
                [...]
                break;
        case MSG_1_ID:
                hsk_can_fifo_getData(fifo0, data1);
                [...]
                break;
        [...]
        }
        hsk_can_fifo_next(fifo0);
}
```

FIFOs draw from the same message object pool regular message objects do.

15.4.6 Message Data

The hsk_can_data_setSignal() and hsk_can_data_getSignal() functions allow writing and reading signals across byte boundaries to and from a buffer.

For big endian signals the bit position of the most significant bit must be supplied (highest bit in the first byte). For little endian signals the least significant bit must be supplied (lowest bit in the first byte).

This conforms to the way signal positions are stored in Vector CANdb++ DBC files.

15.4.7 Macro Definition Documentation

15.4.7.1 CAN0

```
#define CAN0 0
```

CAN node 0.

15.4.7.2 CAN0_IO_P10_P11

```
#define CAN0_IO_P10_P11 0
```

CAN node 0 IO RX on P1.0, TX on P1.1.

15.4.7.3 CAN0_IO_P16_P17

```
#define CAN0_IO_P16_P17 1
```

CAN node 0 IO RX on P1.6, TX on P1.7.

15.4.7.4 CAN0_IO_P34_P35

```
#define CAN0_IO_P34_P35 2
```

CAN node 0 IO RX on P3.4, TX on P3.5.

15.4.7.5 CAN0_IO_P40_P41

```
#define CAN0_IO_P40_P41 3
```

CAN node 0 IO RX on P4.0, TX on P4.1.

15.4.7.6 CAN1

```
#define CAN1 1
```

CAN node 1.

15.4.7.7 CAN1_IO_P01_P02

```
#define CAN1_IO_P01_P02 4
```

CAN node 1 IO RX on P0.1, TX on P0.2.

15.4.7.8 CAN1_IO_P14_P13

```
#define CAN1_IO_P14_P13 5
```

CAN node 1 IO RX on P1.4, TX on P1.3.

15.4.7.9 CAN1_IO_P32_P33

```
#define CAN1_IO_P32_P33 6
```

CAN node 1 IO RX on P3.2, TX on P3.3.

15.4.7.10 CAN_ENDIAN_INTEL

```
#define CAN_ENDIAN_INTEL 0
```

Little endian signal encoding.

Deprecated In favour of shorter and cleaner code the hsk_can_data_getSignal() and hsk_can_data_setSignal() functions were switched to using boolean (motorola positive) logic

15.4.7.11 CAN_ENDIAN_MOTOROLA

```
#define CAN_ENDIAN_MOTOROLA 1
```

Big endian signal encoding.

Deprecated In favour of shorter and cleaner code the hsk_can_data_getSignal() and hsk_can_data_setSignal() functions were switched to using boolean (motorola positive) logic

15.4.7.12 CAN_ERROR

```
#define CAN_ERROR 0xff
```

Value returned by functions in case of an error.

15.4.8 Typedef Documentation

15.4.8.1 hsk_can_fifo

```
typedef ubyte hsk_can_fifo
```

CAN message FIFO identifiers.

15.4.8.2 hsk_can_msg

```
typedef ubyte hsk_can_msg
```

CAN message object identifiers.

15.4.8.3 hsk_can_node

```
typedef ubyte hsk_can_node
```

CAN node identifiers.

15.4.9 Function Documentation

15.4.9.1 hsk_can_data_getSignal()

```
ulong hsk_can_data_getSignal (
            const ubyte *const msg,
            const bool motorola,
            const bool sign,
            const ubyte bitPos,
            const char bitCount )
```

Get a signal value from a data field.

Parameters

msg	The message data field to read from
motorola	Indicates big endian (Motorola) encoding
sign	Indicates whether the value has a signed type
bitPos	The bit position of the signal
bitCount	The length of the signal

Returns

The signal from the data field msg

15.4.9.2 hsk_can_data_setSignal()

```
void hsk_can_data_setSignal (
            ubyte *const msg,
            const bool motorola,
            const bool sign,
            const ubyte bitPos,
            const char bitCount,
            const ulong value )
```

Sets a signal value in a data field.

Parameters

msg	The message data field to write into
motorola	Indicates big endian (Motorola) encoding
sign	Indicates whether the value has a signed type
bitPos	The bit position of the signal
bitCount	The length of the signal
value	The signal value to write into the data field

15.4.9.3 hsk_can_disable()

```
void hsk_can_disable (
            const hsk_can_node node )
```

Disable a CAN node.

This completely shuts down a CAN node, cutting it off from the internal clock, to reduce energy consumption.

Parameters

node	The CAN node to disable

15.4.9.4 hsk_can_enable()

```
void hsk_can_enable (
            const hsk_can_node node )
```

Go live on the CAN bus.

To be called when everything is set up.

Parameters

node	The CAN node to enable

15.4.9.5 hsk_can_fifo_connect()

```
ubyte hsk_can_fifo_connect (
            const hsk_can_fifo fifo,
            const hsk_can_node node )
```

Connect a FIFO to a CAN node.

Parameters

fifo	The identifier of the FIFO
node	The CAN node to connect to

Return values

CAN_ERROR	The given FIFO is not valid
0	Success

15.4.9.6 hsk_can_fifo_create()

```
hsk_can_fifo hsk_can_fifo_create (
            ubyte size )
```

Creates a message FIFO.

FIFOs can be used to ensure that multiplexed signals are not lost.

For receiving multiplexed signals it is recommended to use a FIFO as large as the number of multiplexed messages that might occur in a single burst.

If the multiplexor is large, e.g. 8 bits, it's obviously not possible to carve a 256 messages FIFO out of 32 message objects. Make an educated guess and hope that the signal provider is not hostile.

If the number of available message objects is at least one, but less than the requested length this function succeeds, but the FIFO is only created as long as possible.

Parameters

size	The desired FIFO size

Return values

CAN_ERROR	Creating the FIFO failed
[0;32[	The created FIFO id

15.4.9.7 hsk_can_fifo_delete()

```
ubyte hsk_can_fifo_delete (
            const hsk_can_fifo fifo )
```

Delete a FIFO.

Parameters

fifo	The identifier of the FIFO

Return values

CAN_ERROR	The given FIFO is not valid
0	Success

15.4.9.8 hsk_can_fifo_disconnect()

```
ubyte hsk_can_fifo_disconnect (
            const hsk_can_fifo fifo )
```

Disconnect a FIFO from its CAN node.

This takes the FIFO out of active communication, without deleting it.

Parameters

fifo	The identifier of the FIFO

Return values

CAN_ERROR	The given FIFO is not valid
0	Success

15.4.9.9 hsk_can_fifo_getData()

```
void hsk_can_fifo_getData (
```

```
            const hsk_can_fifo fifo,
            ubyte *const msgdata )
```

Gets the data from the currently selected FIFO entry.

This writes DLC bytes from the FIFO entry into msgdata.

Parameters

fifo	The identifier of the FIFO
msgdata	The character array to store the message data in

15.4.9.10 hsk_can_fifo_getId()

```
ulong hsk_can_fifo_getId (
            const hsk_can_fifo fifo )
```

Returns the CAN ID of the selected FIFO entry.

Parameters

fifo	The ID of the FIFO

Returns

The ID of the currently selected message object

15.4.9.11 hsk_can_fifo_next()

```
void hsk_can_fifo_next (
            const hsk_can_fifo fifo )
```

Select the next FIFO entry.

The hsk_can_fifo_updated() and hsk_can_fifo_getData() functions always refer to a certain message within the FIFO. This function selects the next entry.

Parameters

fifo	The ID of the FIFO to select the next entry from

15.4.9.12 hsk_can_fifo_setRxMask()

```
void hsk_can_fifo_setRxMask (
            const hsk_can_fifo fifo,
            ulong msk )
```

Changes the ID matching mask of an RX FIFO.

Every RX FIFO is setup to receive only on complete ID matches. This function allows updating the mask.

To generate a mask from a list of IDs use the following formula:

$$msk = \sim (id_0|id_1|...|id_n)|(id_0 \& id_1 \& ... \& id_n)$$

Precondition

hsk_can_fifo_setupRx()

Parameters

fifo	The FIFO to change the RX mask for
msk	The bit mask to set for the FIFO

15.4.9.13 hsk_can_fifo_setupRx()

```
void hsk_can_fifo_setupRx (
            hsk_can_fifo fifo,
            const ulong id,
            const bool extended,
            const ubyte dlc )
```

Set the FIFO up for receiving messages.

Parameters

fifo	The FIFO to setup
id	The message ID.
extended	Set this to 1 for an extended CAN message
dlc	The data length code, # of bytes in the message, valid values range from 0 to 8

15.4.9.14 hsk_can_fifo_updated()

```
bool hsk_can_fifo_updated (
            const hsk_can_fifo fifo )
```

Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.

It can be used to decide when to call hsk_can_fifo_getData() and hsk_can_fifo_next().

Parameters

fifo	The identifier of the FIFO to check

Return values

1	The FIFO entry was updated since the last call of this function
0	The FIFO entry has not been updated since the last call of this function

15.4.9.15 hsk_can_init()

```
void hsk_can_init (
            const ubyte pins,
            const ulong baud )
```

Setup CAN communication with the desired baud rate.

The CAN node is chosen with the pin configuration.

The bus still needs to be enabled after being setup.

Parameters

pins	Choose one of 7 CANn_IO_∗ configurations
baud	The target baud rate to use

15.4.9.16 hsk_can_msg_connect()

```
ubyte hsk_can_msg_connect (
            const hsk_can_msg msg,
            const hsk_can_node node )
```

Connect a message object to a CAN node.

Parameters

msg	The identifier of the message object
node	The CAN node to connect to

Return values

CAN_ERROR	The given message is not valid
0	Success

15.4.9.17 hsk_can_msg_create()

```
hsk_can_msg hsk_can_msg_create (
            const ulong id,
            const bool extended,
            const ubyte dlc )
```

Creates a new CAN message.

Note that only up to 32 messages can exist at any given time.

Extended messages have 29 bit IDs and non-extended 11 bit IDs.

Parameters

id	The message ID.
extended	Set this to 1 for an extended CAN message.
dlc	The data length code, # of bytes in the message, valid values range from 0 to 8.

Return values

CAN_ERROR	Creating the message failed
[0;32[	A message identifier

15.4.9.18 hsk_can_msg_delete()

```
ubyte hsk_can_msg_delete (
            const hsk_can_msg msg )
```

Delete a CAN message object.

Parameters

msg	The identifier of the message object

Return values

CAN_ERROR	The given message is not valid
0	Success

15.4.9.19 hsk_can_msg_disconnect()

```
ubyte hsk_can_msg_disconnect (
            const hsk_can_msg msg )
```

Disconnect a CAN message object from its CAN node.

This takes a CAN message out of active communication, without deleting it.

Parameters

msg	The identifier of the message object

Return values

CAN_ERROR	The given message is not valid
0	Success

15.4.9.20 hsk_can_msg_getData()

```
void hsk_can_msg_getData (
            const hsk_can_msg msg,
            ubyte *const msgdata )
```

Gets the current data in the CAN message.

This writes DLC bytes from the CAN message object into msgdata.

Parameters

msg	The identifier of the message object
msgdata	The character array to store the message data in

15.4.9.21 hsk_can_msg_receive()

```
void hsk_can_msg_receive (
            const hsk_can_msg msg )
```

Return the message into RX mode after sending a message.

After sending a message the messages with the same ID from other bus participants are ignored. This restores the original setting to receive messages.

Parameters

msg	The identifier of the message to receive

15.4.9.22 hsk_can_msg_send()

```
void hsk_can_msg_send (
            const hsk_can_msg msg )
```

Request transmission of a message.

Parameters

msg	The identifier of the message to send

15.4.9.23 hsk_can_msg_sent()

```
bool hsk_can_msg_sent (
            const hsk_can_msg msg )
```

Return whether the message was successfully sent between this and the previous call of this method.

Parameters

msg	The identifier of the message to check

Return values

1	The message was sent since the last call of this function
0	The message has not been sent since the last call of this function

15.4.9.24 hsk_can_msg_setData()

```
void hsk_can_msg_setData (
            const hsk_can_msg msg,
            const ubyte *const msgdata )
```

Sets the current data in the CAN message.

This writes DLC bytes from msgdata to the CAN message object.

Parameters

msg	The identifier of the message object
msgdata	The character array to get the message data from

15.4.9.25 hsk_can_msg_updated()

```
bool hsk_can_msg_updated (
            const hsk_can_msg msg )
```

Return whether the message was updated via CAN bus between this call and the previous call of this method.

An update does not entail a change of message data. It just means the message was received on the CAN bus.

This is useful for cyclic message occurance checks.

Parameters

msg	The identifier of the message to check

Return values

1	The message was updated since the last call of this function
0	The message has not been updated since the last call of this function

15.4.9.26 hsk_can_status()

```
ubyte hsk_can_status (
            const hsk_can_node node,
            const ubyte field )
```

Returns a status field of a CAN node.

Parameters

node	The CAN node to return the status of
field	The status field to select

Returns

The status field state

See also

CAN Node Status Fields

15.5 hsk_ex/hsk_ex.h File Reference

HSK External Interrupt Routing headers.

Macros

- #define EX_EXINT0 0

  External interrupt channel EXINT0.
- #define EX_EXINT1 1

  External interrupt channel EXINT1.
- #define EX_EXINT2 2

  External interrupt channel EXINT2.
- #define EX_EXINT3 3

  External interrupt channel EXINT3.
- #define EX_EXINT4 4

  External interrupt channel EXINT4.
- #define EX_EXINT5 5

  External interrupt channel EXINT5.
- #define EX_EXINT6 6

  External interrupt channel EXINT6.
- #define EX_EDGE_RISING 0

Trigger interrupt on rising edge.

- #define EX_EDGE_FALLING 1

  Trigger interrupt on falling edge.

- #define EX_EDGE_BOTH 2

  Trigger interrupt on both edges.

- #define EX_EXINT0_P05 0

  External interrupt EXINT0 input port P0.5.

- #define EX_EXINT3_P11 1

  External interrupt EXINT3 input port P1.1.

- #define EX_EXINT0_P14 2

  External interrupt EXINT0 input port P1.4.

- #define EX_EXINT5_P15 3

  External interrupt EXINT5 input port P1.5.

- #define EX_EXINT6_P16 4

  External interrupt EXINT6 input port P1.6.

- #define EX_EXINT3_P30 5

  External interrupt EXINT3 input port P3.0.

- #define EX_EXINT4_P32 6

  External interrupt EXINT4 input port P3.2.

- #define EX_EXINT5_P33 7

  External interrupt EXINT5 input port P3.3.

- #define EX_EXINT6_P34 8

  External interrupt EXINT6 input port P3.4.

- #define EX_EXINT4_P37 9

  External interrupt EXINT4 input port P3.7.

- #define EX_EXINT3_P40 10

  External interrupt EXINT3 input port P4.0.

- #define EX_EXINT4_P41 11

  External interrupt EXINT4 input port P4.1.

- #define EX_EXINT6_P42 12

  External interrupt EXINT6 input port P4.2.

- #define EX_EXINT5_P44 13

  External interrupt EXINT5 input port P4.4.

- #define EX_EXINT6_P45 14

  External interrupt EXINT6 input port P4.5.

- #define EX_EXINT1_P50 15

  External interrupt EXINT1 input port P5.0.

- #define EX_EXINT2_P51 16

  External interrupt EXINT2 input port P5.1.

- #define EX_EXINT5_P52 17

  External interrupt EXINT5 input port P5.2.

- #define EX_EXINT1_P53 18

  External interrupt EXINT1 input port P5.3.

- #define EX_EXINT2_P54 19

  External interrupt EXINT2 input port P5.4.

- #define EX_EXINT3_P55 20

  External interrupt EXINT3 input port P5.5.

- #define EX_EXINT4_P56 21

  External interrupt EXINT4 input port P5.6.

- #define EX_EXINT6_P57 22

  External interrupt EXINT6 input port P5.7.

Typedefs

- typedef ubyte hsk_ex_channel

  Typedef for externel interrupt channels.
- typedef ubyte hsk_ex_port

  Typedef for externel interrupt ports.

Functions

- void hsk_ex_channel_enable (const hsk_ex_channel channel, const ubyte edge, const void(∗const callback)(void))

  Enable an external interrupt channel.
- void hsk_ex_channel_disable (const hsk_ex_channel channel)

  Disables an external interrupt channel.
- void hsk_ex_port_open (const hsk_ex_port port)

  Opens an input port for an external interrupt.
- void hsk_ex_port_close (const hsk_ex_port port)

  Disconnects an input port from an external interrupt.

15.5.1 Detailed Description

HSK External Interrupt Routing headers.

This file offers functions to activate external interrupts and connect them to the available input pins.

Author

kami

15.5.2 Typedef Documentation

15.5.2.1 hsk_ex_channel

```
typedef ubyte hsk_ex_channel
```

Typedef for externel interrupt channels.

15.5.2.2 hsk_ex_port

```
typedef ubyte hsk_ex_port
```

Typedef for externel interrupt ports.

15.5.3 Function Documentation

15.5.3.1 hsk_ex_channel_disable()

```
void hsk_ex_channel_disable (
            const hsk_ex_channel channel )
```

Disables an external interrupt channel.

Parameters

channel	The channel to disable, one of External Interrupt Channels

15.5.3.2 hsk_ex_channel_enable()

```
void hsk_ex_channel_enable (
            const hsk_ex_channel channel,
            const ubyte edge,
            const void(*)(void) callback )
```

Enable an external interrupt channel.

It is good practice to enable a port for the channel first, because port changes on an active interrupt may cause an undesired interrupt.

The callback function can be set to 0 if a change of the function is not desired. For channels EXINT0 and EXINT1 the callback is ignored, implement interrupts 0 and 2 instead.

Parameters

channel	The channel to activate, one of External Interrupt Channels
edge	The triggering edge, one of External Interrupt Triggers
callback	The callback function for an interrupt event

15.5.3.3 hsk_ex_port_close()

```
void hsk_ex_port_close (
            const hsk_ex_port port )
```

Disconnects an input port from an external interrupt.

Parameters

port	The port to close, one of External Interrupt Input Ports

15.5.3.4 hsk_ex_port_open()

```
void hsk_ex_port_open (
            const hsk_ex_port port )
```

Opens an input port for an external interrupt.

Parameters

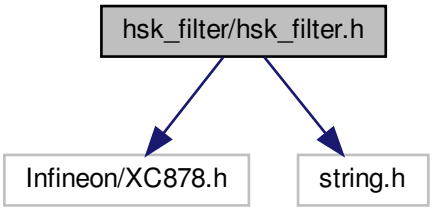
port	The port to open, one of External Interrupt Input Ports

15.6 hsk_filter/hsk_filter.h File Reference

HSK Filter generator.

```
#include <Infineon/XC878.h>
#include <string.h>
```

Include dependency graph for hsk_filter.h:



Macros

- #define FILTER_FACTORY(prefix, valueType, sumType, sizeType, size)
  Generates a filter.
- #define FILTER_GROUP_FACTORY(prefix, filters, valueType, sumType, sizeType, size)
  Generates a group of filters.

15.6.1 Detailed Description

HSK Filter generator.

This file offers preprocessor macros to filter analogue values, by calculating the average of a set of a given length.

The buffer for the filter is stored in xdata memory.

Author

kami

15.6.2 Macro Definition Documentation

15.6.2.1 FILTER_FACTORY

```
#define FILTER_FACTORY(
              prefix,
              valueType,
              sumType,
              sizeType,
              size )
```

Generates a filter.

The filter can be accessed with:

- void <prefix>_init(void)
  - Initializes the filter with 0
- <valueType> <prefix>_update(const <valueType> value)
  - Update the filter and return the current average

Parameters

prefix	A prefix for the generated internals and functions
valueType	The data type of the stored values
sumType	A data type that can contain the sum of all buffered values
sizeType	A data type that can hold the length of the buffer
size	The length of the buffer

15.6.2.2 FILTER_GROUP_FACTORY

```
#define FILTER_GROUP_FACTORY(
              prefix,
              filters,
              valueType,
              sumType,
              sizeType,
              size )
```

Generates a group of filters.

The filters can be accessed with:

- void <prefix>_init(void)
  - Initializes all filters with 0
- <valueType> <prefix>_update(const ubyte filter, const <valueType> value)
  - Update the given filter and return the current average

Parameters

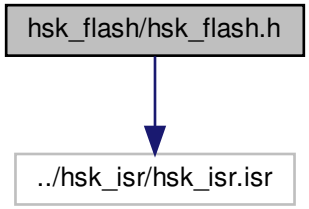
prefix	A prefix for the generated internals and functions
filters	The number of filters
valueType	The data type of the stored values
sumType	A data type that can contain the sum of all buffered values
sizeType	A data type that can hold the length of the buffer
size	The length of the buffer

15.7 hsk_flash/hsk_flash.h File Reference

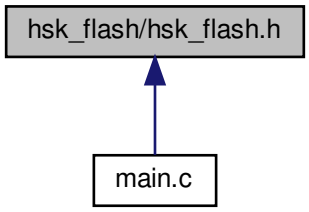
HSK Flash Facility headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_flash.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define XC878_16FF

  Ensure that a flash memory layout is defined.
- #define FLASH_STRUCT_FACTORY(members)

  Used to create a struct that can be used with the hsk_flash_init() function.
- #define FLASH_PWR_FIRST 0

  Returned by hsk_flash_init() when the µC boots for the first time.
- #define FLASH_PWR_RESET 1

  Returned by hsk_flash_init() after booting from a reset without power loss.
- #define FLASH_PWR_ON 2

  Returned by hsk_flash_init() during power on, if valid data was recovered from the D-Flash.

Functions

- ubyte hsk_flash_init (void *const ptr, const uword size, const ubyte version)

  Recovers a struct from a previous session and sets everything up for storage of changes.
- bool hsk_flash_write (void)

  Writes the current data to the D-Flash.

15.7.1 Detailed Description

HSK Flash Facility headers.

This file contains function prototypes to menage information that survives a reset and allow storage within the D-Flash.

It provides the FLASH_STRUCT_FACTORY to create a struct with data that can be stored with hsk_flash_write() and recovered with hsk_flash_init().

The D-Flash is used as a ring buffer, this distributes writes over the entire flash to gain the maximum achievable lifetime. The lifetime expectancy depends on your usage scenario and the size of the struct.

Refer to section 3.3 table 20 and table 21 of the XC87x data sheet for D-Flash life times.

Complete coverage of the D-Flash counts as a single D-Flash cycle. Thus the formula for the expected number of write calls is:

$$writes = \lfloor 4096/sizeof(struct) \rfloor * expectedcycles$$

- `expectedcycles`
  - The expected number of possible write cycles depending on the usage scenario in table 20
- `sizeof(struct)`
  - The number of bytes the struct covers
- `floor()`
  - Round down to the next smaller integer

E.g. to store 20 bytes of configuration data, the struct factory adds 2 bytes overhead to be able to check the consistency of written data, so $sizeof(struct) = 22$. Expecting that most of the µC use is within the first year, table 20 suggests that $expectedcycles = 100000$. In that case the expected number of possible hsk_flash_write() calls is 18.6 million.

Author

kami

15.7.2 Byte Order

C51 stores multiple byte variables in big endian order, whereas the DPTR register, several SFRs and SDCC use little endian.

If the data struct contains multibyte members such as int/uword or long/ulong, this can lead to data corruption, when switching compilers.

Both the checksum and identifier are single byte values and thus will still match after a compiler switch, causing multibyte values to be restored from the flash with the wrong byte order.

A byte order change can be detected with a byte order word in the struct. A BOW initialized with 0x1234 would read 0x3412 after a an order change.

The suggested solution is to only create struct members with byte wise access. E.g. a ulong member may be defined in the following way:

```
ubyte ulongMember[sizeof(ulong)];
```

The value can be set like this:

```
myStruct.ulongMember[0] = ulongValue;
myStruct.ulongMember[1] = ulongValue >> 8;
myStruct.ulongMember[2] = ulongValue >> 16;
myStruct.ulongMember[3] = ulongValue >> 24;
```

Reading works similarly:

```
ulongValue  = (ubyte)myStruct.ulongMember[0];
ulongValue |= (uword)myStruct.ulongMember[1] << 8;
ulongValue |= (ulong)myStruct.ulongMember[2] << 16;
ulongValue |= (ulong)myStruct.ulongMember[3] << 24;
```

Another alternative is to use a single ubyte[] array and store/read all data with the hsk_can_data_setSignal()/hsk_can_data_getSignal() functions. Due to the bit addressing of CAN message data the maximum length of such an array would be 32 bytes (256bits).

An advantage would be that less memory is required, because data no longer needs to be byte aligned.

15.7.3 Macro Definition Documentation

15.7.3.1 FLASH_PWR_FIRST

```
#define FLASH_PWR_FIRST 0
```

Returned by hsk_flash_init() when the µC boots for the first time.

This statements holds true *as far as can be told*. I.e. a first boot is diagnosed when all attempts to recover data have failed.

Two scenarios may cause this:

- No valid data has yet been written to the D-Flash
- The latest flash data is corrupted, may happen in case of power down during write

15.7.3.2 FLASH_PWR_ON

```
#define FLASH_PWR_ON 2
```

Returned by hsk_flash_init() during power on, if valid data was recovered from the D-Flash.

A power on is detected when two criteria are met:

- Data could not be recovered from `xdata` memory
- Valid data was recovered from the D-Flash

15.7.3.3 FLASH_PWR_RESET

```
#define FLASH_PWR_RESET 1
```

Returned by hsk_flash_init() after booting from a reset without power loss.

The typical mark of a reset is that `xdata` memory still holds data from the previous session. If such data is found it will just be picked up.

For performance reasons access to the struct is not guarded, which means that there can be no protection against data corruption, such as might be caused by a software bug like an overflow.

15.7.3.4 FLASH_STRUCT_FACTORY

```
#define FLASH_STRUCT_FACTORY(
               members )
```

Value:

```
/**
This struct is a template for data that can be written to the D-Flash.
It is created by invoking the \ref FLASH_STRUCT_FACTORY macro.
         */\
        volatile struct hsk_flash_struct {\
                 /**
For data integrity/compatibilty detection.

@private
                  */\
                 ubyte hsk_flash_prefix;\
                 \
                 members\
                 \
                 /**
For data integrity detection.

@private
                  */\
                 ubyte hsk_flash_chksum;\
        } xdata
```

Used to create a struct that can be used with the hsk_flash_init() function.

The hsk_flash_init() function expects certain fields to exist, in the struct, which are used to ensure the consistency of data in the flash.

The following example shows how to create a struct named storableData:

```
FLASH_STRUCT_FACTORY(
        ubyte storableByte;
        uword storableWord;
) storableData;
```

Parameters

members	Struct member definitions

15.7.3.5 XC878_16FF

```
#define XC878_16FF
```

Ensure that a flash memory layout is defined.

Either XC878_16FF (64k flash) or XC878_13FF(52k flash) are supported. XC878_16FF is the default.

15.7.4 Function Documentation

15.7.4.1 hsk_flash_init()

```
ubyte hsk_flash_init (
            void *const ptr,
            const uword size,
            const ubyte version )
```

Recovers a struct from a previous session and sets everything up for storage of changes.

There are two modes of recovery. After a fresh boot the data can be recovered from flash, if previously stored there. After a simple reset the data can still be found in XRAM and recovery can be sped up.

If recovery fails entirely all members of the struct will be set to 0.

Parameters

version	Version number of the persisted data structure, used to prevent initilization with incompatible data
ptr	A pointer to the xdata struct/array to persist
size	The size of the data structure to persist

Return values

FLASH_PWR_FIRST	No valid data was recovered
FLASH_PWR_RESET	Continue operation after a reset
FLASH_PWR_ON	Data restore from the D-Flash succeeded

15.7.4.2 hsk_flash_write()

```
bool hsk_flash_write (
            void  )
```

Writes the current data to the D-Flash.

Ongoing writes are interrupted. Ongoing deletes are interrupted unless there is insufficient space left to write the data.

Return values

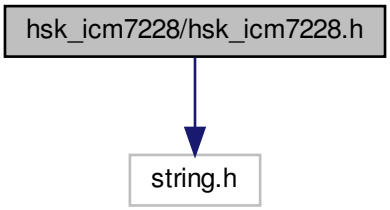
1	The D-Flash write is on the way
0	Not enough free D-Flash space to write, try again later

15.8 hsk_icm7228/hsk_icm7228.h File Reference

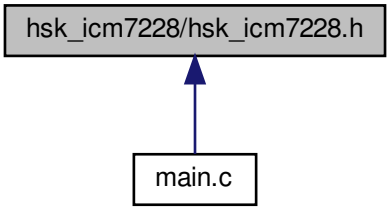
HSK ICM7228 8-Digit LED Display Decoder Driver generator.

```
#include <string.h>
```

Include dependency graph for hsk_icm7228.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ICM7228_FACTORY(prefix, regData, regMode, bitMode, regWrite, bitWrite)

  Generate an ICM7228 driver instance.

Functions

- void hsk_icm7228_writeString (ubyte ∗const buffer, char const ∗str, ubyte pos, ubyte len)

  Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.
- void hsk_icm7228_writeDec (ubyte ∗const buffer, uword value, char power, ubyte const pos, ubyte len)

  Write a 7 segment encoded, right aligned decimal number into an xdata buffer.
- void hsk_icm7228_writeHex (ubyte ∗const buffer, uword value, char power, ubyte const pos, ubyte len)

  Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.
- void hsk_icm7228_illuminate (ubyte ∗const buffer, ubyte segments, ubyte pos, ubyte len)

  Illumante the given number of segments.

15.8.1 Detailed Description

HSK ICM7228 8-Digit LED Display Decoder Driver generator.

This file is a code generating facility, that offers preprocessor macros that produce code for the Intersil ICM7228 display decoder.

Generating code in this fashion avoids the hard coding of I/O registers and bits and even allows the use of multiple ICM7228 ICs.

See also

Intersil ICM7228 Data Sheet: ICM7228.pdf

Author

kami

15.8.2 Macro Definition Documentation

15.8.2.1 ICM7228_FACTORY

```
#define ICM7228_FACTORY(
              prefix,
              regData,
              regMode,
              bitMode,
              regWrite,
              bitWrite )
```

Generate an ICM7228 driver instance.

This creates functions to use a connect ICM7228 IC.

- void <prefix>_init(void)
  - Initialize the buffer and I/O register bits
- void <prefix>_refresh(void)
  - Commit buffered data to the 7 segment displays
- void <prefix>_writeString(char * str, ubyte pos, ubyte len)
  - Wrapper around hsk_icm7228_writeString()
- void <prefix>_writeDec(uword value, char power, ubyte pos, ubyte len)
  - Wrapper around hsk_icm7228_writeDec()
- void <prefix>_writeHex(uword value, char power, ubyte pos, ubyte len)
  - Wrapper around hsk_icm7228_writeHex()

Parameters

prefix	A prefix for the names of generated functions
regData	The register that is connected to the data input
regMode	The register that is connected to the mode pin
bitMode	The bit of the regMode register that is connected to the mode pin
regWrite	The register that is connected to the write pin
bitWrite	The bit of the regWrite register that is connected to the write pin

15.8.3 Function Documentation

15.8.3.1 hsk_icm7228_illuminate()

```
void hsk_icm7228_illuminate (
             ubyte *const buffer,
             ubyte segments,
             ubyte pos,
             ubyte len )
```

Illumante the given number of segments.

Parameters

buffer	The target buffer for the encoded string
segments	The number of segments to illuminate
pos	The target position in the buffer
len	The number of digits available to encode the number

15.8.3.2 hsk_icm7228_writeDec()

```
void hsk_icm7228_writeDec (
             ubyte *const buffer,
             uword value,
             char power,
             ubyte const pos,
             ubyte len )
```

Write a 7 segment encoded, right aligned decimal number into an xdata buffer.

The power parameter controlls the placing of the '.' by 10 to the power. E.g. value = 12, power = -1 and len = 3 would result in the encoding of " 1.2". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "012".

Parameters

buffer	The target buffer for the encoded string
value	The number to encode
power	The 10 base power of the number to encode
pos	The target position in the buffer
len	The number of digits available to encode the number

15.8.3.3 hsk_icm7228_writeHex()

```
void hsk_icm7228_writeHex (
             ubyte *const buffer,
             uword value,
             char power,
             ubyte const pos,
             ubyte len )
```

Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.

The power parameter controlls the placing of the '.' by 16 to the power. E.g. value = 0x1A, power = -1 and len = 3 would result in the encoding of " 1.A". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "01A".

Parameters

buffer	The target buffer for the encoded string
value	The number to encode
power	The 16 base power of the number to encode
pos	The target position in the buffer
len	The number of digits available to encode the number

15.8.3.4 hsk_icm7228_writeString()

```
void hsk_icm7228_writeString (
             ubyte *const buffer,
             char const * str,
             ubyte pos,
             ubyte len )
```

Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.

This function is usually invoked through the <prefix>_writeString() function created by ICM7228_FACTORY.

The function will write into the buffer until it has been filled with len characters or it encounters a 0 character reading from str. If the character '.' is encountered it is merged with the previous character, unless that character is a '.' itself. Thus a single dot does not use additional buffer space. The 7 character string "foo ..." would result in 6 encoded bytes. Thus the proper len value for that string would be 6.

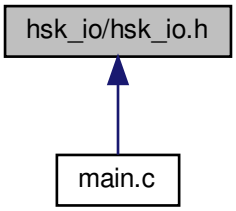
Parameters

buffer	The target buffer for the encoded string
str	The buffer to read the ASCII string from
pos	The position in the buffer to write the encoded string to
len	The target length of the encoded string

15.9 hsk_io/hsk_io.h File Reference

HSK I/O headers.

This graph shows which files directly or indirectly include this file:



Macros

- #define IO_PORT_IN_INIT(port, pins)
  Initializes a set of port pins as inputs.
- #define IO_PORT_ON_GND 0
  Bit mask to set the logical 1 to GND level for all selected pins.
- #define IO_PORT_ON_HIGH 0xff
  Bit mask to set the logical 1 to high level for all selected pins.
- #define IO_PORT_GET(port, pins, on)
  Evaluates to a bit mask of logical pin states of a port.
- #define IO_PORT_STRENGTH_WEAK 0
  Bit mask to set weak drive strength for all selected pins.
- #define IO_PORT_STRENGTH_STRONG 0xff
  Bit mask to set strong drive strength for all selected pins.
- #define IO_PORT_DRAIN_DISABLE 0
  Bit mask to disable drain mode for all selected pins.
- #define IO_PORT_DRAIN_ENABLE 0xff
  Bit mask to enable drain mode for all selected pins.
- #define IO_PORT_OUT_INIT(port, pins, strength, drain, on, set)
  Initializes a set of port pins as outputs.
- #define IO_PORT_OUT_SET(port, pins, on, set)
  Set a set of output port pins.
- #define IO_PORT_PULL_DISABLE 0
  Bit mask to disable pull up/down for all selected pins.
- #define IO_PORT_PULL_ENABLE 0xff
  Bit mask to enable pull up/down for all selected pins.
- #define IO_PORT_PULL_DOWN 0
  Bit mask to select pull down for all selected pins.
- #define IO_PORT_PULL_UP 0xff
  Bit mask to select pull up for all selected pins.
- #define IO_PORT_PULL_INIT(port, pins, pull, dir)

Sets the pull-up/-down properties of port pins.

- #define IO_VAR_SET(var, bits, on, set)

  Set a set of variable bits.

- #define IO_VAR_GET(var, bits, on)

  Evaluates to a bit mask of logical states of a variable.

15.9.1 Detailed Description

HSK I/O headers.

This file contains macro definitions to use and initialize I/O ports and variables bitwise.

All the macros take a port and a mask to select the affected pins. All operations are masked with the selected pins so it is safe to define 0xff (every bit 1) to activate a certain property.

Set and get macros take a bit field to define the value that represents the `on` or `true` state, so the logic code can always use a `1` for `true`/`on`.

The macros are grouped as:

- Input Port Access
- Output Port Access
- Variable Access

Author

kami

15.9.2 I/O Port Pull-Up/-Down Table

The device boots with all parallel ports configured as inputs. The following table lists the pins that come up with activated internal pull up:

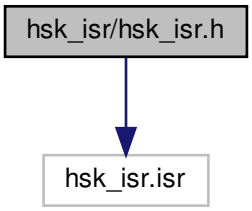
Port\Bit	7	6	5	4	3	2	1	0
P0	1	1	x	x	x	1	x	x
P1	1	1	1	1	1	1	1	1
P3	x	1	x	x	x	x	x	x
P4	x	x	x	x	x	1	x	x
P5	1	1	1	1	1	1	1	1

15.10 hsk_isr/hsk_isr.h File Reference

HSK Shared Interrupt Service Routine headers.

```
#include "hsk_isr.isr"
```

Include dependency graph for hsk_isr.h:



Data Structures

- struct hsk_isr5_callback
  Shared interrupt 5 routine.
- struct hsk_isr6_callback
  Shared interrupt 6 routine.
- struct hsk_isr8_callback
  Shared interrupt 8 routine.
- struct hsk_isr9_callback
  Shared interrupt 9 routine.
- struct hsk_isr14_callback
  Shared non-maskable interrupt routine.

Variables

- volatile struct hsk_isr5_callback hsk_isr5
  Introduce callback function pointers for ISR 5.
- volatile struct hsk_isr6_callback hsk_isr6
  Introduce callback function pointers for ISR 6.
- volatile struct hsk_isr8_callback hsk_isr8
  Introduce callback function pointers for ISR 8.
- volatile struct hsk_isr9_callback hsk_isr9
  Introduce callback function pointers for ISR 9.
- volatile struct hsk_isr14_callback hsk_isr14
  Introduce callback function pointers for NMI ISR.

15.10.1 Detailed Description

HSK Shared Interrupt Service Routine headers.

This header is used by other libraries to use interrupts with multiple sources. A callback function can be provided for each available interrupt source.

Author

kami

15.10.2 SFR Pages

An ISR callback function cannot make assumptions about current SFR pages like the regular functions that can expect all pages to be set to 0.

Instead a callback function needs to set all pages and restore whatever page was in use previously.

The following table lists the store and restore selectors by context and must be obeyed to avoid memory corruption:

Save	Restore	Context
SST0	RST0	ISRs
SST1	RST1	ISR callback functions
SST2	RST2	NMI ISR
SST3	RST3	NMI callback functions

Every callback function is called with RMAP = 0. If the callback function changes RMAP it does not have to take care of restoring it. RMAP is always restored to its original state by the shared ISRs.

15.10.3 Register Banks

Interrupts are each a root node of their own call tree. This is why they must preserve all the working registers.

the pushing and popping of the 8 `Rn` registers for each interrupt call costs 64 CCLK cycles.

To avoid this overhead different register banks are used. Call trees, i.e. interrupts, can use the same register bank if they cannot interrupt each other. Each used register bank costs 8 bytes of regular `data` memory. To minimize this cost all interrupts must have the same priority.

The following table is used:

Priority	Context	Bank
-	Regular code	0
0	ISR, callback	1
1	ISR, callback	-
2	ISR, callback	-
3	ISR, callback	-
NMI	NMI ISR, callback	2

Assigning higher priority to an ISR will affect (as in break) the operation of all lower priority ISRs.

15.10.4 Variable Documentation

15.10.4.1 hsk_isr14

```
volatile struct hsk_isr14_callback hsk_isr14
```

Introduce callback function pointers for NMI ISR.

Functions called back from the NMI ISR should use SST3/RST3 instead of SST1/RST1, because they might interrupt other ISRs.

15.10.4.2 hsk_isr5

```
volatile struct hsk_isr5_callback hsk_isr5
```

Introduce callback function pointers for ISR 5.

15.10.4.3 hsk_isr6

```
volatile struct hsk_isr6_callback hsk_isr6
```

Introduce callback function pointers for ISR 6.

15.10.4.4 hsk_isr8

```
volatile struct hsk_isr8_callback hsk_isr8
```

Introduce callback function pointers for ISR 8.

15.10.4.5 hsk_isr9

```
volatile struct hsk_isr9_callback hsk_isr9
```

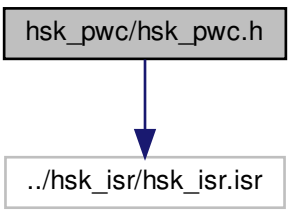
Introduce callback function pointers for ISR 9.

15.11 hsk_pwc/hsk_pwc.h File Reference

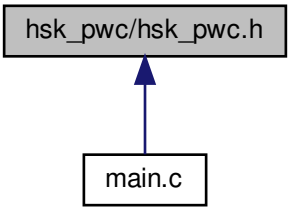
HSK Pulse Width Counter headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_pwc.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define PWC_CC0 0

  Capture/Compare channel 0 on EXINT3.
- #define PWC_CC1 1

  Capture/Compare channel 1 on EXINT4.
- #define PWC_CC2 2

  Capture/Compare channel 2 on EXINT5.
- #define PWC_CC3 3

  Capture/Compare channel 3 on EXINT6.
- #define PWC_CC0_P30 0

  Capture/Compare channel 0 input port P3.0 configuration.
- #define PWC_CC0_P40 1

  Capture/Compare channel 0 input port P4.0 configuration.
- #define PWC_CC0_P55 2

  Capture/Compare channel 0 input port P5.5 configuration.
- #define PWC_CC1_P32 3

  Capture/Compare channel 1 input port P3.2 configuration.
- #define PWC_CC1_P41 4

  Capture/Compare channel 1 input port P4.1 configuration.
- #define PWC_CC1_P56 5

  Capture/Compare channel 1 input port P5.6 configuration.
- #define PWC_CC2_P33 6

  Capture/Compare channel 2 input port P3.3 configuration.
- #define PWC_CC2_P44 7

  Capture/Compare channel 4 input port P4.4 configuration.
- #define PWC_CC2_P52 8

  Capture/Compare channel 2 input port P5.2 configuration.
- #define PWC_CC3_P34 9

  Capture/Compare channel 3 input port P3.4 configuration.
- #define PWC_CC3_P45 10

  Capture/Compare channel 3 input port P4.5 configuration.
- #define PWC_CC3_P57 11

  Capture/Compare channel 3 input port P5.7 configuration.
- #define PWC_EDGE_FALLING 0

Configuration selection to trigger pulse detection on falling edge.

- #define PWC_EDGE_RISING 1

  Configuration selection to trigger pulse detection on rising edge.

- #define PWC_EDGE_BOTH 2

  Configuration selection to trigger pulse detection on both edges.

- #define PWC_MODE_EXT 1

  Available capture modes, capture on external interrupt.

- #define PWC_MODE_SOFT 3

  Available capture modes, capture on sofware event.

- #define PWC_UNIT_SUM_RAW 0

  Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

- #define PWC_UNIT_WIDTH_RAW 1

  Average of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

- #define PWC_UNIT_WIDTH_NS 2

  Average of buffered pulse widths in multiples of $10^{-9} s$.

- #define PWC_UNIT_WIDTH_US 3

  Average of buffered pulse widths in multiples of $10^{-6} s$.

- #define PWC_UNIT_WIDTH_MS 4

  Average of buffered pulse widths in multiples of $10^{-3} s$.

- #define PWC_UNIT_FREQ_S 5

  Average frequency of buffered pulses in multiples of $1/s$.

- #define PWC_UNIT_FREQ_M 6

  Average frequency of buffered pulses in multiples of $1/m$.

- #define PWC_UNIT_FREQ_H 7

  Average frequency of buffered pulses in multiples of $1/h$.

- #define PWC_UNIT_DUTYH_RAW 8

  Latest high pulse in multiples of $1/48 * 10^{-6} s$.

- #define PWC_UNIT_DUTYH_NS 9

  Latest high pulse in multiples of $1 * 10^{-9} s$.

- #define PWC_UNIT_DUTYH_US 10

  Latest high pulse in multiples of $1 * 10^{-6} s$.

- #define PWC_UNIT_DUTYH_MS 11

  Latest high pulse in multiples of $1 * 10^{-3} s$.

- #define PWC_UNIT_DUTYL_RAW 12

  Latest low pulse in multiples of $1/48 * 10^{-6} s$.

- #define PWC_UNIT_DUTYL_NS 13

  Latest low pulse in multiples of $1 * 10^{-9} s$.

- #define PWC_UNIT_DUTYL_US 14

  Latest low pulse in multiples of $1 * 10^{-6} s$.

- #define PWC_UNIT_DUTYL_MS 15

  Latest low pulse in multiples of $1 * 10^{-3} s$.

Typedefs

- typedef ubyte hsk_pwc_channel

  Typedef for PWC channel IDs.

- typedef ubyte hsk_pwc_port

  Typedef for PWC input port.

Functions

- void hsk_pwc_init (ulong window)

  This function initializes the T2CCU Capture/Compare Unit for capture mode.
- void hsk_pwc_channel_open (const hsk_pwc_channel channel, ubyte averageOver)

  Configures a PWC channel without an input port.
- void hsk_pwc_port_open (const hsk_pwc_port port, ubyte averageOver)

  Opens an input port and the connected channel.
- void hsk_pwc_channel_close (const hsk_pwc_channel channel)

  Close a PWC channel.
- void hsk_pwc_channel_edgeMode (const hsk_pwc_channel channel, const ubyte edgeMode)

  Select the edge that is used to detect a pulse.
- void hsk_pwc_channel_captureMode (const hsk_pwc_channel channel, const ubyte captureMode)

  Allows switching between external and soft trigger.
- void hsk_pwc_channel_trigger (const hsk_pwc_channel channel)

  Triggers a channel in soft trigger mode.
- void hsk_pwc_enable (void)

  Enables T2CCU module if disabled.
- void hsk_pwc_disable (void)

  Turns off the T2CCU clock to preserve power.
- ulong hsk_pwc_channel_getValue (const hsk_pwc_channel channel, const ubyte unit)

  Returns a measure of the values in a channel buffer.

15.11.1 Detailed Description

HSK Pulse Width Counter headers.

This library uses the T2CCU to measure pulse width on the external interrupt pins.

Every caputre channel blocks an external interrupt. Opening a channel will block this interrupt and change its configuration.

Pulse with measurement has a window time that is configured with hsk_pwc_init() and defines the time frame within which pulses can be detected.

If no pulse occurs during the window, the channel buffer is invalidated and the hsk_pwc_channel_getValue() function will returns invalid (0) until the buffer is repopulated with valid measurements.

In order to guarantee the detection of invalid channels, the hsk_pwc_channel_getValue() function has to be called at least once every 256 window times.

Author

kami

15.11.2 Macro Definition Documentation

15.11.2.1 PWC_CC0

```
#define PWC_CC0 0
```

Capture/Compare channel 0 on EXINT3.

15.11.2.2 PWC_CC0_P30

```
#define PWC_CC0_P30 0
```

Capture/Compare channel 0 input port P3.0 configuration.

15.11.2.3 PWC_CC0_P40

```
#define PWC_CC0_P40 1
```

Capture/Compare channel 0 input port P4.0 configuration.

15.11.2.4 PWC_CC0_P55

```
#define PWC_CC0_P55 2
```

Capture/Compare channel 0 input port P5.5 configuration.

15.11.2.5 PWC_CC1

```
#define PWC_CC1 1
```

Capture/Compare channel 1 on EXINT4.

15.11.2.6 PWC_CC1_P32

```
#define PWC_CC1_P32 3
```

Capture/Compare channel 1 input port P3.2 configuration.

15.11.2.7 PWC_CC1_P41

```
#define PWC_CC1_P41 4
```

Capture/Compare channel 1 input port P4.1 configuration.

15.11.2.8 PWC_CC1_P56

```
#define PWC_CC1_P56 5
```

Capture/Compare channel 1 input port P5.6 configuration.

15.11.2.9 PWC_CC2

```
#define PWC_CC2 2
```

Capture/Compare channel 2 on EXINT5.

15.11.2.10 PWC_CC2_P33

```
#define PWC_CC2_P33 6
```

Capture/Compare channel 2 input port P3.3 configuration.

15.11.2.11 PWC_CC2_P44

```
#define PWC_CC2_P44 7
```

Capture/Compare channel 4 input port P4.4 configuration.

15.11.2.12 PWC_CC2_P52

```
#define PWC_CC2_P52 8
```

Capture/Compare channel 2 input port P5.2 configuration.

15.11.2.13 PWC_CC3

```
#define PWC_CC3 3
```

Capture/Compare channel 3 on EXINT6.

15.11.2.14 PWC_CC3_P34

```
#define PWC_CC3_P34 9
```

Capture/Compare channel 3 input port P3.4 configuration.

15.11.2.15 PWC_CC3_P45

```
#define PWC_CC3_P45 10
```

Capture/Compare channel 3 input port P4.5 configuration.

15.11.2.16 PWC_CC3_P57

```
#define PWC_CC3_P57 11
```

Capture/Compare channel 3 input port P5.7 configuration.

15.11.2.17 PWC_EDGE_BOTH

```
#define PWC_EDGE_BOTH 2
```

Configuration selection to trigger pulse detection on both edges.

15.11.2.18 PWC_EDGE_FALLING

```
#define PWC_EDGE_FALLING 0
```

Configuration selection to trigger pulse detection on falling edge.

15.11.2.19 PWC_EDGE_RISING

```
#define PWC_EDGE_RISING 1
```

Configuration selection to trigger pulse detection on rising edge.

15.11.2.20 PWC_MODE_EXT

```
#define PWC_MODE_EXT 1
```

Available capture modes, capture on external interrupt.

15.11.2.21 PWC_MODE_SOFT

```
#define PWC_MODE_SOFT 3
```

Available capture modes, capture on sofware event.

15.11.3 Typedef Documentation

15.11.3.1 hsk_pwc_channel

```
typedef ubyte hsk_pwc_channel
```

Typedef for PWC channel IDs.

15.11.3.2 hsk_pwc_port

```
typedef ubyte hsk_pwc_port
```

Typedef for PWC input port.

15.11.4 Function Documentation

15.11.4.1 hsk_pwc_channel_captureMode()

```
void hsk_pwc_channel_captureMode (
            const hsk_pwc_channel channel,
            const ubyte captureMode )
```

Allows switching between external and soft trigger.

This does not reconfigure the input ports. Available modes are specified in the PWC_MODE_∗ defines. PWC_M↩ ODE_EXT is the default.

Parameters

channel	The channel to configure.
captureMode	The mode to set the channel to.

15.11.4.2 hsk_pwc_channel_close()

```
void hsk_pwc_channel_close (
            const hsk_pwc_channel channel )
```

Close a PWC channel.

Parameters

channel	The channel to close.

15.11.4.3 hsk_pwc_channel_edgeMode()

```
void hsk_pwc_channel_edgeMode (
            const hsk_pwc_channel channel,
            const ubyte edgeMode )
```

Select the edge that is used to detect a pulse.

Available edges are specified in the PWC_EDGE_∗ defines.

Parameters

channel	The channel to configure the edge for.
edgeMode	The selected edge detection mode.

15.11.4.4 hsk_pwc_channel_getValue()

```
ulong hsk_pwc_channel_getValue (
            const hsk_pwc_channel channel,
            const ubyte unit )
```

Returns a measure of the values in a channel buffer.

It also takes care of invalidating channels that haven't been captured for too long.

The value is returned in a requested unit, the units defined as PWC_UNIT_∗ are available.

Parameters

channel	The channel to return the buffer sum of
unit	' The unit to return the channel value in

Return values

>0	The channel value in the requested unit
0	Invalid channel, measurement timed out

15.11.4.5 hsk_pwc_channel_open()

```
void hsk_pwc_channel_open (
            const hsk_pwc_channel channel,
            ubyte averageOver )
```

Configures a PWC channel without an input port.

The channel is set up for software triggering (PWC_MODE_SOFT), and triggering on both edges (PWC_EDGE_↩ BOTH).

Parameters

channel	The PWC channel to open
averageOver	The number of pulse values to average over when returning a value or speed. The value must be between 1 and 8.

15.11.4.6 hsk_pwc_channel_trigger()

```
void hsk_pwc_channel_trigger (
            const hsk_pwc_channel channel )
```

Triggers a channel in soft trigger mode.

Parameters

channel	The channel to trigger.

15.11.4.7 hsk_pwc_disable()

```
void hsk_pwc_disable (
            void  )
```

Turns off the T2CCU clock to preserve power.

15.11.4.8 hsk_pwc_enable()

```
void hsk_pwc_enable (
            void  )
```

Enables T2CCU module if disabled.

15.11.4.9 hsk_pwc_init()

```
void hsk_pwc_init (
            ulong window )
```

This function initializes the T2CCU Capture/Compare Unit for capture mode.

The capturing is based on the CCT timer. Timer T2 is not used and thus can be useed without interference.

The window time is the time frame within which pulses should be detected. A smaller time frame results in higher precission, but detection of longer pulses will fail.

Window times vary between $\sim$1ms ($(2^{16}-1)/(48*10^6)$) and $\sim$5592ms ($(2^{16}-1)*2^{12}/(48*10^6)$). The shortest window time delivers $\sim$20ns and the longest time $\sim$85µs precision.

The real window time is on a logarithmic scale (base 2), the init function will select the lowest scale that guarantees the required window time. I.e. the highest precision possible with the desired window time, which is at least 2^{15} for all windows below or equal 5592ms.

Parameters

window	The time in ms to detect a pulse.

15.11.4.10 hsk_pwc_port_open()

```
void hsk_pwc_port_open (
            const hsk_pwc_port port,
            ubyte averageOver )
```

Opens an input port and the connected channel.

The available configurations are available from the PWC_CCn_$*$ defines.

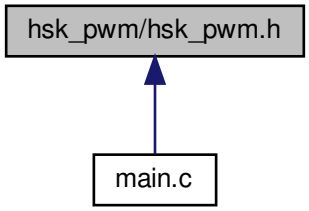
Parameters

port	The input port to open
averageOver	The number of pulse values to average over when returning a value or speed. The value must be between 1 and CHAN_BUF_SIZE

15.12 hsk_pwm/hsk_pwm.h File Reference

HSK Pulse Width Modulation headers.

This graph shows which files directly or indirectly include this file:



Macros

- #define PWM_60 0

  PWM channel 60, Timer T12 driven.
- #define PWM_61 1

  PWM channel 61, Timer T12 driven.
- #define PWM_62 2

  PWM channel 62, Timer T12 driven.
- #define PWM_63 3

  PWM channel 63, Timer T13 driven.
- #define PWM_CC60 0

  IO channel configuration for PWM_60.
- #define PWM_COUT60 1

  Output channel configuration for PWM_60.
- #define PWM_CC61 2

  IO channel configuration for PWM_61.
- #define PWM_COUT61 3

  Output channel configuration for PWM_61.
- #define PWM_CC62 4

  IO channel configuration for PWM_62.
- #define PWM_COUT62 5

  Output channel configuration for PWM_62,.
- #define PWM_COUT63 6

  Output channel configuration for PWM_63.
- #define PWM_OUT_60_P30 0

  PWM_60 output configuration for P3.0 through PWM_CC60.
- #define PWM_OUT_60_P31 1

  PWM_60 output configuration for P3.1 through PWM_COUT60.
- #define PWM_OUT_60_P40 2

  PWM_60 output configuration for P4.0 through PWM_CC60.
- #define PWM_OUT_60_P41 3

  PWM_60 output configuration for P4.1 through PWM_COUT60.
- #define PWM_OUT_61_P00 4

  PWM_61 output configuration for P0.0 through PWM_CC61.
- #define PWM_OUT_61_P01 5

PWM_61 output configuration for P0.1 through PWM_COUT61.

- #define PWM_OUT_61_P31 6

  PWM_61 output configuration for P3.1 through PWM_CC61.
- #define PWM_OUT_61_P32 7

  PWM_61 output configuration for P3.2 through PWM_CC61.
- #define PWM_OUT_61_P33 8

  PWM_61 output configuration for P3.3 through PWM_COUT61.
- #define PWM_OUT_61_P44 9

  PWM_61 output configuration for P4.4 through PWM_CC61.
- #define PWM_OUT_61_P45 10

  PWM_61 output configuration for P4.5 through PWM_COUT61.
- #define PWM_OUT_62_P04 11

  PWM_62 output configuration for P0.4 through PWM_CC62.
- #define PWM_OUT_62_P05 12

  PWM_62 output configuration for P0.5 through PWM_COUT62.
- #define PWM_OUT_62_P34 13

  PWM_62 output configuration for P3.4 through PWM_CC62.
- #define PWM_OUT_62_P35 14

  PWM_62 output configuration for P3.5 through PWM_COUT62.
- #define PWM_OUT_62_P46 15

  PWM_62 output configuration for P4.6 through PWM_CC62.
- #define PWM_OUT_62_P47 16

  PWM_62 output configuration for P4.7 through PWM_COUT62.
- #define PWM_OUT_63_P03 17

  PWM_63 output configuration for P0.3 through PWM_COUT63.
- #define PWM_OUT_63_P37 18

  PWM_63 output configuration for P3.7 through PWM_COUT63.
- #define PWM_OUT_63_P43 19

  PWM_63 output configuration for P4.3 through PWM_COUT63.

Typedefs

- typedef ubyte hsk_pwm_channel

  Type definition for PWM channels.
- typedef ubyte hsk_pwm_outChannel

  Type definition for output channels.
- typedef ubyte hsk_pwm_port

  Type definition for ports.

Functions

- void hsk_pwm_init (const hsk_pwm_channel channel, const ulong freq)

  Sets up the the CCU6 timer frequencies that control the PWM cycle.
- void hsk_pwm_port_open (const hsk_pwm_port port)

  Set up a PWM output port.
- void hsk_pwm_port_close (const hsk_pwm_port port)

  Close a PWM output port.
- void hsk_pwm_channel_set (const hsk_pwm_channel channel, const uword max, const uword value)

  Set the duty cycle for the given channel.

- void hsk_pwm_outChannel_dir (hsk_pwm_outChannel channel, const bool up)

  Set the direction of an output channel.
- void hsk_pwm_enable (void)

  Turns on the CCU6.
- void hsk_pwm_disable (void)

  Deactivates the CCU6 to reduce power consumption.

15.12.1 Detailed Description

HSK Pulse Width Modulation headers.

This file provides function prototypes to perform Timer T12 and T13 based PWM with CCU6.

The CCU6 offers the following PWM channels:

- PWM_60
- PWM_61
- PWM_62
- PWM_63

Each PWM channel is connected to two IO channels for output:

- PWM_CCx
- PWM_COUTx

The distinction between PWM and IO channels is important to understand the side effects of some operations.

Refer to the PWM_OUT_x_* defines to know which channel can be connected to which output pins.

The functions are implemented under the assumption, that the use of the timers T12 and T13 as well of the CCU6 is exclusive to this library.

The safe boot order for pwm output is the following:

- hsk_pwm_init()
- hsk_pwm_enable()
- hsk_pwm_port_open()

Author

kami

15.12.2 Macro Definition Documentation

15.12.2.1 PWM_60

```
#define PWM_60 0
```

PWM channel 60, Timer T12 driven.

15.12.2.2 PWM_61

```
#define PWM_61 1
```

PWM channel 61, Timer T12 driven.

15.12.2.3 PWM_62

```
#define PWM_62 2
```

PWM channel 62, Timer T12 driven.

15.12.2.4 PWM_63

```
#define PWM_63 3
```

PWM channel 63, Timer T13 driven.

15.12.2.5 PWM_CC60

```
#define PWM_CC60 0
```

IO channel configuration for PWM_60.

15.12.2.6 PWM_CC61

```
#define PWM_CC61 2
```

IO channel configuration for PWM_61.

15.12.2.7 PWM_CC62

```
#define PWM_CC62 4
```

IO channel configuration for PWM_62.

15.12.2.8 PWM_COUT60

```
#define PWM_COUT60 1
```

Output channel configuration for PWM_60.

15.12.2.9 PWM_COUT61

```
#define PWM_COUT61 3
```

Output channel configuration for PWM_61.

15.12.2.10 PWM_COUT62

```
#define PWM_COUT62 5
```

Output channel configuration for PWM_62,.

15.12.2.11 PWM_COUT63

```
#define PWM_COUT63 6
```

Output channel configuration for PWM_63.

15.12.2.12 PWM_OUT_60_P30

```
#define PWM_OUT_60_P30 0
```

PWM_60 output configuration for P3.0 through PWM_CC60.

15.12.2.13 PWM_OUT_60_P31

```
#define PWM_OUT_60_P31 1
```

PWM_60 output configuration for P3.1 through PWM_COUT60.

15.12.2.14 PWM_OUT_60_P40

```
#define PWM_OUT_60_P40 2
```

PWM_60 output configuration for P4.0 through PWM_CC60.

15.12.2.15 PWM_OUT_60_P41

```
#define PWM_OUT_60_P41 3
```

PWM_60 output configuration for P4.1 through PWM_COUT60.

15.12.2.16 PWM_OUT_61_P00

```
#define PWM_OUT_61_P00 4
```

PWM_61 output configuration for P0.0 through PWM_CC61.

15.12.2.17 PWM_OUT_61_P01

```
#define PWM_OUT_61_P01 5
```

PWM_61 output configuration for P0.1 through PWM_COUT61.

15.12.2.18 PWM_OUT_61_P31

```
#define PWM_OUT_61_P31 6
```

PWM_61 output configuration for P3.1 through PWM_CC61.

15.12.2.19 PWM_OUT_61_P32

```
#define PWM_OUT_61_P32 7
```

PWM_61 output configuration for P3.2 through PWM_CC61.

15.12.2.20 PWM_OUT_61_P33

```
#define PWM_OUT_61_P33 8
```

PWM_61 output configuration for P3.3 through PWM_COUT61.

15.12.2.21 PWM_OUT_61_P44

```
#define PWM_OUT_61_P44 9
```

PWM_61 output configuration for P4.4 through PWM_CC61.

15.12.2.22 PWM_OUT_61_P45

```
#define PWM_OUT_61_P45 10
```

PWM_61 output configuration for P4.5 through PWM_COUT61.

15.12.2.23 PWM_OUT_62_P04

```
#define PWM_OUT_62_P04 11
```

PWM_62 output configuration for P0.4 through PWM_CC62.

15.12.2.24 PWM_OUT_62_P05

```
#define PWM_OUT_62_P05 12
```

PWM_62 output configuration for P0.5 through PWM_COUT62.

15.12.2.25 PWM_OUT_62_P34

```
#define PWM_OUT_62_P34 13
```

PWM_62 output configuration for P3.4 through PWM_CC62.

15.12.2.26 PWM_OUT_62_P35

```
#define PWM_OUT_62_P35 14
```

PWM_62 output configuration for P3.5 through PWM_COUT62.

15.12.2.27 PWM_OUT_62_P46

```
#define PWM_OUT_62_P46 15
```

PWM_62 output configuration for P4.6 through PWM_CC62.

15.12.2.28 PWM_OUT_62_P47

```
#define PWM_OUT_62_P47 16
```

PWM_62 output configuration for P4.7 through PWM_COUT62.

15.12.2.29 PWM_OUT_63_P03

```
#define PWM_OUT_63_P03 17
```

PWM_63 output configuration for P0.3 through PWM_COUT63.

15.12.2.30 PWM_OUT_63_P37

```
#define PWM_OUT_63_P37 18
```

PWM_63 output configuration for P3.7 through PWM_COUT63.

15.12.2.31 PWM_OUT_63_P43

```
#define PWM_OUT_63_P43 19
```

PWM_63 output configuration for P4.3 through PWM_COUT63.

15.12.3 Typedef Documentation

15.12.3.1 hsk_pwm_channel

```
typedef ubyte hsk_pwm_channel
```

Type definition for PWM channels.

15.12.3.2 hsk_pwm_outChannel

```
typedef ubyte hsk_pwm_outChannel
```

Type definition for output channels.

15.12.3.3 hsk_pwm_port

```
typedef ubyte hsk_pwm_port
```

Type definition for ports.

15.12.4 Function Documentation

15.12.4.1 hsk_pwm_channel_set()

```
void hsk_pwm_channel_set (
             const hsk_pwm_channel channel,
             const uword max,
             const uword value )
```

Set the duty cycle for the given channel.

I.e. the active time frame slice of period can be set with max and value.

To set the duty cycle in percent specify a max of 100 and values from 0 to 100.

Parameters

channel	The PWM channel to set the duty cycle for, check the PWM_6x defines
max	Defines the scope value can move in
value	The current duty cycle value

15.12.4.2 hsk_pwm_disable()

```
void hsk_pwm_disable (
             void  )
```

Deactivates the CCU6 to reduce power consumption.

15.12.4.3 hsk_pwm_enable()

```
void hsk_pwm_enable (
             void  )
```

Turns on the CCU6.

Deactivates the power disable mode and sets the T12 and T13 Timer Run bits.

Precondition

All hsk_pwm_init() calls have to be completed to call this

15.12.4.4 hsk_pwm_init()

```
void hsk_pwm_init (
            const hsk_pwm_channel channel,
            const ulong freq )
```

Sets up the the CCU6 timer frequencies that control the PWM cycle.

The channels PWM_60, PWM_61 and PWM_62 share the timer T12, thus initializing one of them, initializes them all. The channel PWM_63 has exclusive use of the timer T13 and can thus be used with its own operating frequency.

Frequencies up to ~732.4Hz are always between 15 and 16 bits precision.

Frequencies above 48kHz offer less than 1/1000 precision. From there it is a linear function, i.e. 480kHz still offer 1/100 precision.

The freq value 0 will result in ~0.02Hz ($48000000/2^{31}$).

The following formula results in the freq value that yields exactly the desired precision, this is useful to avoid precision loss by rounding:

$$freq(precision) = 480000000 * precision$$

E.g. 10 bit precision: $freq(1/2^{10}) = 468750$

Parameters

channel	The channel to change the frequency for
freq	The desired PWM cycle frequency in units of 0.1Hz

15.12.4.5 hsk_pwm_outChannel_dir()

```
void hsk_pwm_outChannel_dir (
            hsk_pwm_outChannel channel,
            const bool up )
```

Set the direction of an output channel.

The channel value can be taken from any of the PWM_CCx/PWM_COUTx defines.

Parameters

channel	The IO channel to set the direction bit for
up	Set 1 to output a 1 during the cycle set with hsk_pwm_channel_set(), set 0 to output a 0 during the cycle set with hsk_pwm_channel_set()

15.12.4.6 hsk_pwm_port_close()

```
void hsk_pwm_port_close (
            const hsk_pwm_port port )
```

Close a PWM output port.

This configures the necessary port direction bits.

The port can be any one of the PWM_OUT_x_∗ defines.

Parameters

port	The output port to deactivate

15.12.4.7 hsk_pwm_port_open()

```
void hsk_pwm_port_open (
             const hsk_pwm_port port )
```

Set up a PWM output port.

This configures the necessary port direction bits and activates the corresponding output channels.

The port can be any one of the PWM_OUT_x_∗ defines.

Precondition

This function should only be called after hsk_pwm_enable(), otherwise the output port will be driven (1) until PWM is enabled

Parameters

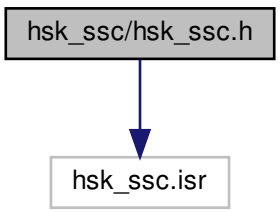
port	The output port to activate

15.13 hsk_ssc/hsk_ssc.h File Reference

HSK Synchronous Serial Interface headers.

```
#include "hsk_ssc.isr"
```

Include dependency graph for hsk_ssc.h:



Macros

- #define SSC_MRST_P05 1

  Master mode RX, slave mode TX port P0.5.
- #define SSC_MRST_P14 0

  Master mode RX, slave mode TX port P1.4.
- #define SSC_MRST_P15 2

  Master mode RX, slave mode TX port P1.5.
- #define SSC_MTSR_P04 (1 << 2)

  Master mode TX, slave mode RX port P0.4.
- #define SSC_MTSR_P13 (0 << 2)

  Master mode TX, slave mode RX port P1.3.
- #define SSC_MTSR_P14 (2 << 2)

  Master mode TX, slave mode RX port P1.4.
- #define SSC_SCLK_P03 (1 << 4)

  Synchronous clock port P0.3.
- #define SSC_SCLK_P12 (0 << 4)

  Synchronous clock port P1.2.
- #define SSC_SCLK_P13 (2 << 4)

  Synchronous clock port P1.3.
- #define SSC_MASTER 1

  Master mode, output shift clock on SCLK.
- #define SSC_SLAVE 0

  Slave mode, receive shift clock on SCLK.
- #define SSC_BAUD(bps) (uword)(12000000ul / (bps) - 1)

  Converts a baud rate value in bits/s into a baud rate value for the hsk_ssc_init() function.
- #define SSC_CONF(width, heading, phase, polarity, duplex) (((width) - 1) | ((heading) << 4) | ((phase) << 5) | ((polarity) << 6) | ((duplex) << 7))

  Generates an SSC configuration byte.
- #define hsk_ssc_busy() ESSC

  Returns whether the SSC is currently busy with data transmission.

Functions

- void hsk_ssc_init (const uword baud, const ubyte config, const bool mode)

  The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.
- void hsk_ssc_ports (const ubyte ports)

  Configure the I/O ports of the SSC unit.
- void hsk_ssc_talk (char *buffer, ubyte len)

  Send and receive data.
- void hsk_ssc_enable ()

  Turn the SSC module on.
- void hsk_ssc_disable ()

  Turn the SSC module off.

15.13.1 Detailed Description

HSK Synchronous Serial Interface headers.

General purpose serial communication, setup in the following order:

- hsk_ssc_init()
- hsk_ssc_ports()
- hsk_ssc_enable()

Communication is established by the hsk_ssc_talk() function. Use hsk_ssc_busy() to detect whether a buffer was completely read and written.

Author

kami

15.13.2 Half Duplex Operation

For half duplex operation TX and RX pins need to be short circuited.

The TX pin is set up in open drain mode, i.e. an external pull-up resistor is required.

The TX pin needs to be manually configured before calling hsk_ssc_talk() in order to speak or listen on the bus. To listen the TX pin needs to be configured as an input pin, to speak on the bus as an output pin. For efficiency reasons this is not handled by this library (it would result in lots of runtime logic for what should be a single instruction).

Instead it is recommended to define macros in a central header. E.g. for the port configuration SSC_MRST_P05 in slave mode the following code would work:

```
#define SSC_TX()    (P0_DIR |= 1 << 5)
#define SSC_RX()    (P0_DIR &= ~(1 << 5))
```

Syntactically it can be used like a regular function:

```
SSC_TX();
hsk_ssc_talk(buffer, sizeof(buffer) - 1);
```

15.13.3 Macro Definition Documentation

15.13.3.1 hsk_ssc_busy

```
#define hsk_ssc_busy( ) ESSC
```

Returns whether the SSC is currently busy with data transmission.

15.13.3.2 SSC_BAUD

```
#define SSC_BAUD(
              bps ) (uword)(12000000ul / (bps) - 1)
```

Converts a baud rate value in bits/s into a baud rate value for the hsk_ssc_init() function.

The distance between adjustable baud rates grows exponentially. Available baud rates in kBit progress like this:

$$\{12000, 6000, 4000, 3000, 2400, 2000, ...\}$$

Use the following formula to determine the baud rate that results from a desired value:

$$realBps(bps) = \frac{12000000}{\lfloor \frac{12000000}{bps} \rfloor}$$

Note

The maximum speed is 12 Mbit/s in master mode and 6 Mbit/s in slave mode.

Parameters

bps	The desired number in bits/s

Returns

A timer reload value

15.13.3.3 SSC_CONF

```
#define SSC_CONF(
              width,
              heading,
              phase,
              polarity,
              duplex ) (((width) - 1) | ((heading) << 4) | ((phase) << 5) | ((polarity) << 6)
| ((duplex) << 7))
```

Generates an SSC configuration byte.

For details check the XC878 user manual section 12.3.5.1.

Parameters

width	The data with in bits, the available range is $[2; 8]$
heading	Use 0 for transmitting/receiving LSB first, 1 for MSB first
phase	Use 0 to shift on leading and latch on trailing edge, use 1 to shift on trailing and latch on leading edge
polarity	Use 0 for low idle clock, and 1 for high idle clock
duplex	Use 0 for full duplex mode and 1 for half duplex

15.13.3.4 SSC_MASTER

```
#define SSC_MASTER 1
```

Master mode, output shift clock on SCLK.

15.13.3.5 SSC_SLAVE

```
#define SSC_SLAVE 0
```

Slave mode, receive shift clock on SCLK.

15.13.4 Function Documentation

15.13.4.1 hsk_ssc_disable()

```
void hsk_ssc_disable ( )
```

Turn the SSC module off.

15.13.4.2 hsk_ssc_enable()

```
void hsk_ssc_enable ( )
```

Turn the SSC module on.

15.13.4.3 hsk_ssc_init()

```
void hsk_ssc_init (
            const uword baud,
            const ubyte config,
            const bool mode )
```

The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.

Calling this function turns the SSC off until hsk_ssc_enable() is called.

Parameters

baud	The timer reload value for the baud rate generator, use SSC_BAUD to generate this value
config	The SSC configuration byte, use SSC_CONF to generate it
mode	Select master or slave operation

15.13.4.4 hsk_ssc_ports()

```
void hsk_ssc_ports (
```

```
            const ubyte ports )
```

Configure the I/O ports of the SSC unit.

Warning

Do not use when the SSC is enabled.

Parameters

ports	Selects an SSC I/O Ports I/O port configuration

15.13.4.5 hsk_ssc_talk()

```
void hsk_ssc_talk (
            char * buffer,
            ubyte len )
```

Send and receive data.

The buffer with the given length should contain the data to transceive and will be filled with the received data upon completion.

The provided buffer needs to reside in xdata memory, e.g. to create and use a string buffer the following should work:

```
char xdata buffer[] = "20 character buffer.";
...
hsk_ssc_talk(buffer, sizeof(buffer) - 1);
```

Note that char must not be const and that $sizeof(buffer) - 1$ is used to prevent sending and overwriting the terminal 0 character. There may be cases where a terminal 0 character is desired.

Parameters

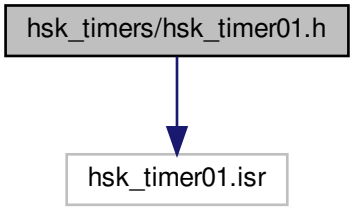
buffer	The rx/tx transmission buffer
len	The length of the buffer

15.14 hsk_timers/hsk_timer01.h File Reference

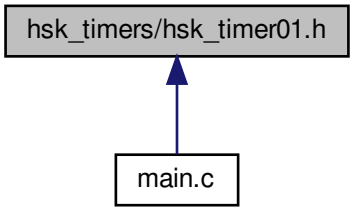
HSK Timer 0/1 headers.

```
#include "hsk_timer01.isr"
```

Include dependency graph for hsk_timer01.h:



This graph shows which files directly or indirectly include this file:



Functions

- void hsk_timer0_setup (const uword interval, const void(∗const callback)(void))

  Setup timer 0 to tick at a given interval.
- void hsk_timer0_enable (void)

  Enables the timer 0 and its interrupt.
- void hsk_timer0_disable (void)

  Disables timer 0 and its interrupt.
- void hsk_timer1_setup (const uword interval, const void(∗const callback)(void))

  Setup timer 1 to tick at a given interval.
- void hsk_timer1_enable (void)

  Enables the timer 1 and its interrupt.
- void hsk_timer1_disable (void)

  Disables timer 1 and its interrupt.

15.14.1 Detailed Description

HSK Timer 0/1 headers.

Provides access to the timers 0 and 1. Each timer can be provided with a callback function that will be called by the timers ISR.

Author

kami

15.14.2 Function Documentation

15.14.2.1 hsk_timer0_disable()

```
void hsk_timer0_disable (
            void  )
```

Disables timer 0 and its interrupt.

15.14.2.2 hsk_timer0_enable()

```
void hsk_timer0_enable (
            void  )
```

Enables the timer 0 and its interrupt.

15.14.2.3 hsk_timer0_setup()

```
void hsk_timer0_setup (
            const uword interval,
            const void(*)(void) callback )
```

Setup timer 0 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

interval	The ticking interval in µs, don't go beyond 5461.
callback	A function pointer to a callback function.

15.14.2.4 hsk_timer1_disable()

```
void hsk_timer1_disable (
```

```
            void  )
```

Disables timer 1 and its interrupt.

15.14.2.5 hsk_timer1_enable()

```
void hsk_timer1_enable (
            void  )
```

Enables the timer 1 and its interrupt.

15.14.2.6 hsk_timer1_setup()

```
void hsk_timer1_setup (
            const uword interval,
            const void(*)(void) callback )
```

Setup timer 1 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

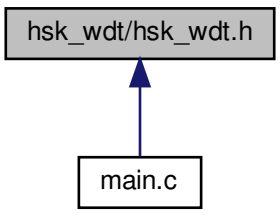
Parameters

interval	The ticking interval in µs, don't go beyond 5461.
callback	A function pointer to a callback function.

15.15 hsk_wdt/hsk_wdt.h File Reference

HSK Watchdog Timer headers.

This graph shows which files directly or indirectly include this file:



Functions

- void hsk_wdt_init (const uword window)

  Sets up the watchdog timer.
- void hsk_wdt_enable (void)

  Activates the Watchdog Timer.
- void hsk_wdt_disable (void)

  Disables the Watchdog Timer.
- void hsk_wdt_service (void)

  Resets the watchdog timer.

15.15.1 Detailed Description

HSK Watchdog Timer headers.

Provides access to the Watchdog Timer (WDT) of the XC878.

Depending on the configured window time the µC reset is delayed for 1.024ms (window $<$ 5460µs) or 65.536ms (window $>=$ 5460µs).

This time can be used by assigning a callback function to hsk_isr14 member hsk_isr14_callback::NMIWDT and setting the NMICON.NMIWDT bit.

Warning

The WDT should be set up at the end of the boot procedure. Setting the WDT up at the beginning of the boot process can trigger all kinds of erratic behaviour like reset races or a complete lockup.

Author

kami

15.15.2 Hazards

The WDT has proven a useful tool in hazardous EMI conditions. Severe EMI may freeze the µC without causing a proper reboot. In most cases the WDT can mitigate this issue by reactivating the system.

However the WDT is trigger happy. A series of refresh time interval measurements shows that the WDT resets the µC long before the end of its interval shortly after boot. The best mitigation is to refresh the WDT with the hsk_↩ wdt_service() function unconditionally. Instead of using fixed timings (e.g. for 20ms watchdog time a 5ms refresh interval should have been quite safe).

That however does not solve the problem with NMIs. Any non-maskable interrupt may cause the WDT to reset the µC. This means it is incopatible to the hsk_flash library, which requires precise timings with only a couple of µs tolerance. To meet this requirement the library uses the Flash Timer of the XC878, which triggers NMIs.

15.15.3 Function Documentation

15.15.3.1 hsk_wdt_disable()

```
void hsk_wdt_disable (
             void  )
```

Disables the Watchdog Timer.

15.15.3.2 hsk_wdt_enable()

```
void hsk_wdt_enable (
            void  )
```

Activates the Watchdog Timer.

15.15.3.3 hsk_wdt_init()

```
void hsk_wdt_init (
            const uword window )
```

Sets up the watchdog timer.

The window time specifies the time available to call hsk_wdt_service() before a reset is triggered. Possible times range from 21.3µs to 350ms.

The window time is rounded up to the next higher possible value. Exceeding the value range causes an overflow that results in shorter window times.

Parameters

window	The time window in multiples of 10µs

15.15.3.4 hsk_wdt_service()

```
void hsk_wdt_service (
            void  )
```

Resets the watchdog timer.

This function needs to be called to prevent the WDT from resetting the device.

15.16 main.c File Reference

Simple test file that is not linked into the library.

```
#include <Infineon/XC878.h>
#include "config.h"
#include "hsk_boot/hsk_boot.h"
#include "hsk_timers/hsk_timer01.h"
#include "hsk_can/hsk_can.h"
#include "hsk_icm7228/hsk_icm7228.h"
#include "hsk_adc/hsk_adc.h"
#include "hsk_pwm/hsk_pwm.h"
#include "hsk_pwc/hsk_pwc.h"
#include "hsk_flash/hsk_flash.h"
#include "hsk_wdt/hsk_wdt.h"
```

```
#include "hsk_io/hsk_io.h"
```

Include dependency graph for main.c:

Macros

- #define PERSIST_VERSION 1

  The version of the persist struct.

Functions

- void p1_init (void)

  Set up buffer and ports for display driver at I/O port P1 .
- void p1_refresh (void)

  Reflesh displays at I/O port P1 with the buffered data.
- void p1_writeString (char const ∗const str, ubyte const pos, ubyte const len)

  Write an ASCII encoded string into p1_buffer.
- void p1_writeDec (uword const value, char const power, ubyte const pos, ubyte const len)

  Write a decimal number into p1_buffer.
- void p1_writeHex (uword const value, char const power, ubyte const pos, ubyte const len)

  Write a hexadecimal number into p1_buffer.
- void p1_illuminate (ubyte const segments, ubyte const pos, ubyte const len)

  Illuminate a number of segments in p1_buffer.
- void main (void)

  Call init functions and invoke the run routine.
- void init (void)

  Initialize ports, timers and ISRs.
- void run (void)

  The main test code body.
- void tick0 (void)

  A ticking function called back by the timer T0 ISR.

Variables

- ubyte p1_buffer [8]

  Buffer for display driver at I/O port P1 .
- volatile struct hsk_flash_struct persist

  This structure is used to persist data between resets.
- volatile uword tick0_count_250 = 0

  A counter used to detecting that 250ms have passed.
- volatile ubyte tick0_count_20 = 10

  A counter used to detecting that 20ms have passed.
- volatile uword adc7

  The storage variable for the potentiometer on the eval board.

15.16.1 Detailed Description

Simple test file that is not linked into the library.

This file is normally rigged to run on the XC800 Starter Kit eval board and used for testing whatever code is currently under development.

Author

kami

15.16.2 Macro Definition Documentation

15.16.2.1 PERSIST_VERSION

```
#define PERSIST_VERSION 1
```

The version of the persist struct.

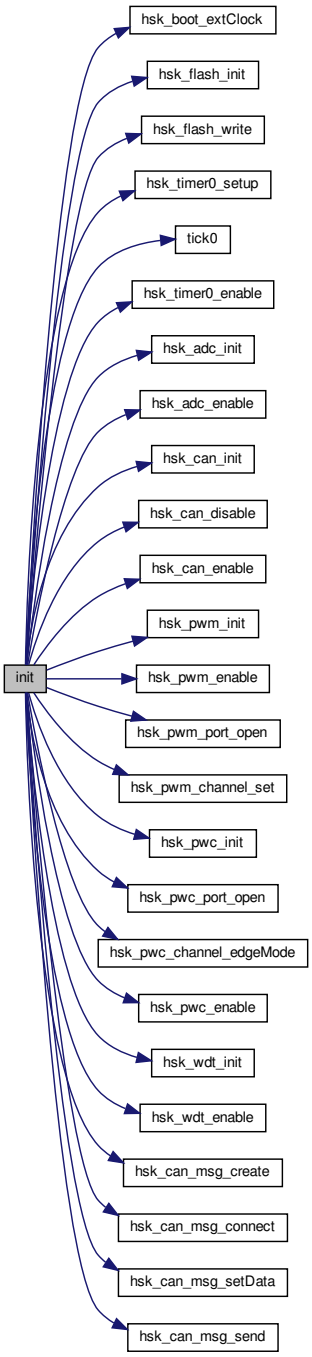
15.16.3 Function Documentation

15.16.3.1 init()

```
void init (
            void  )
```

Initialize ports, timers and ISRs.

Here is the call graph for this function:

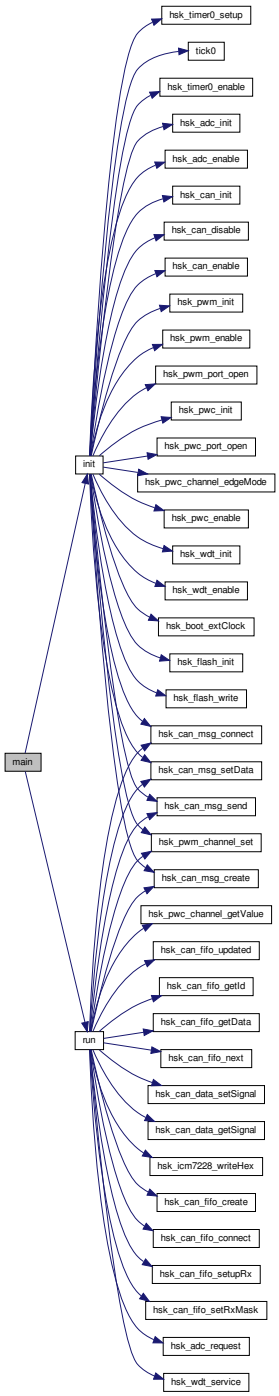


15.16.3.2 main()

```
void main (
            void  )
```

Call init functions and invoke the run routine.

Here is the call graph for this function:



15.16.3.3 p1_illuminate()

```
void p1_illuminate (
            ubyte const segments,
            ubyte const pos,
            ubyte const len )  [inline]
```

Illuminate a number of segments in p1_buffer.

Parameters

segments	The number of segments to illuminate
pos	The target position in the buffer
len	The number of digits available to encode the number

See also

ICM7228_FACTORY
hsk_icm7228_illuminate

15.16.3.4 p1_init()

```
void p1_init (
              void  )
```

Set up buffer and ports for display driver at I/O port P1 .

See also

ICM7228_FACTORY

15.16.3.5 p1_refresh()

```
void p1_refresh (
              void  )
```

Reflesh displays at I/O port P1 with the buffered data.

See also

ICM7228_FACTORY

15.16.3.6 p1_writeDec()

```
void p1_writeDec (
              uword const value,
              char const power,
              ubyte const pos,
              ubyte const len )  [inline]
```

Write a decimal number into p1_buffer.

Parameters

value	The number to encode
power	The 10 base power of the number to encode
pos	The target position in the buffer
len	The number of digits available to encode the number

See also

ICM7228_FACTORY
hsk_icm7228_writeDec

15.16.3.7 p1_writeHex()

```
void p1_writeHex (
            uword const value,
            char const power,
            ubyte const pos,
            ubyte const len )  [inline]
```

Write a hexadecimal number into p1_buffer.

Parameters

value	The number to encode
power	The 16 base power of the number to encode
pos	The target position in the buffer
len	The number of digits available to encode the number

See also

ICM7228_FACTORY
hsk_icm7228_writeHex

15.16.3.8 p1_writeString()

```
void p1_writeString (
            char const *const str,
            ubyte const pos,
            ubyte const len )  [inline]
```

Write an ASCII encoded string into p1_buffer.

Parameters

str	The buffer to read the ASCII string from
pos	The position in the buffer to write the encoded string to
len	The target length of the encoded string

See also

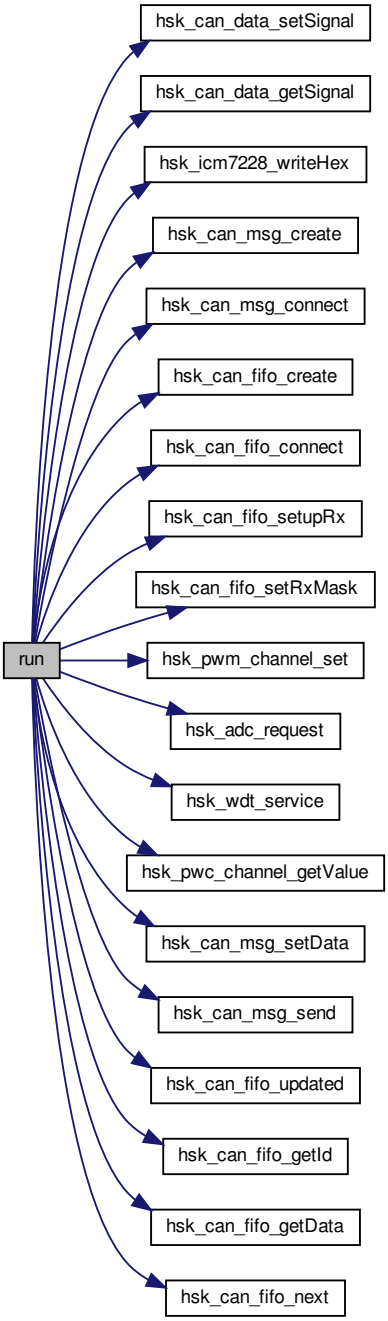
ICM7228_FACTORY
hsk_icm7228_writeString

15.16.3.9 run()

```
void run (
            void  )
```

The main test code body.

Here is the call graph for this function:



15.16.3.10 tick0()

```
void tick0 (
            void  )
```

A ticking function called back by the timer T0 ISR.

15.16.4 Variable Documentation

15.16.4.1 adc7

```
volatile uword adc7
```

The storage variable for the potentiometer on the eval board.

15.16.4.2 p1_buffer

```
ubyte p1_buffer[8]
```

Buffer for display driver at I/O port P1 .

See also

ICM7228_FACTORY

15.16.4.3 persist

```
persist
```

This structure is used to persist data between resets.

15.16.4.4 tick0_count_20

```
volatile ubyte tick0_count_20 = 10
```

A counter used to detecting that 20ms have passed.

15.16.4.5 tick0_count_250

```
volatile uword tick0_count_250 = 0
```

A counter used to detecting that 250ms have passed.

Index