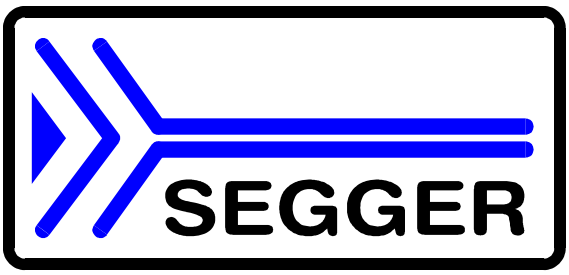
# emLoad

Software Version 3.14m

Manual revision 0




A product of SEGGER Microcontroller GmbH & Co. KG

# Disclaimer

Specifications written in this manual are believed to be accurate, but are not guaranteed to be entirely free of error. Specifications in this manual may be changed for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER MICROCONTROLLER GmbH & Co. KG (the manufacturer) assumes no responsibility for any errors or omissions and makes and you receive no warranties. The manufacturer specifically disclaims any implied warranty of fitness for a particular purpose.

# Copyright notice



# Trademarks

Names mentioned in this manual may be trademarks of their respective companies.
Brand and product names are trademarks or registered trademarks of their respective holders.

# Contact / registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available.
For registration please provide the following information:

- Company name and address
- Your name
- Your job title
- Your Email address and telephone number
- Name and version of the product you purchased

Please send this information to: register@segger.com

Contact address

SEGGER Microcontroller GmbH & Co. KG
In den Weiden 11
D-40721 Hilden
Germany
Email: support@segger.com
Internet: www.segger.com

# Version of software, manual

This manual describes the software ***emLoad*** version 3.14l.

Print date: 11.09.2014

| Manual | Date | By | Explanation |
|---|---|---|---|
| 3.14mR0 | 140911 | OO | - Port added: RX, IAR compiler |
| 3.14lR0 | 140512 | OO | - RX210 devices added to RX, HEW4 compiler |
| 3.14kR0 | 140417 | OO | - Port added: CM3, ST STM32F40x, IAR compiler |
| 3.14jR0 | 131014 | OO | - Port added: M32C, Renesas NC308 compiler |
| 3.14iR0 | 130926 | OO | - Port added: CM3, Freescale Kinetis K10, KEIL MDK ARM<br>- Port added: CM3, ST STM32F20x, KEIL MDK ARM |
| 3.14hR0 | 130801 | OO | - RX63x devices added to RX, HEW4 compiler |
| 3.14R4 | 130222 | OO | - Port added: CM3, STM32F10x, IAR Compiler |
| 3.14R3 | 121218 | OO | - Port added: CM3, STM32L15x, IAR Compiler |
| 3.14R2 | 120604 | OO | - Port added: RX, HEW4 Compiler |
| 3.14R1 | 111027 | OO | - Port added: CM3, Luminary, IAR Compiler |
| 3.14R0 | 110530 | OO | - Fixed prototypes for FLASH0_*/FLASH1_* routines.<br>- Port added: R8C IAR Compiler |
| 3.12R12 | 101102 | OO | - Port added: R32C, HEW4 Compiler<br>- 1M Devices added to R32C port |
| 3.12R11 | 101015 | OO | - Port added : ATMEL SAM7L, IAR Compiler |
| 3.12R10 | 100920 | OO | - Port added : CM3, LPC17xx, IAR Compiler |
| 3.12R9 | 100503 | OO | - Port added : R32C, IAR Compiler |
| 3.12R8 | 100421 | OO | - AT91SAM7SE devices added to AT91SAM7 port |
| 3.12R7 | 100415 | OO | - Port added : M16C/65, IAR Compiler |
| 3.12R6 | 070928 | JE | - Support for block A added to M32C ports |
| 3.12R5 | 070928 | JE | - M3087 CPUs added to Renesas M32C ports |
| 3.12R4 | 070301 | JE | - 'Mitsubishi' replaced with 'Renesas'<br>- Renesas M16C port adapted to new IAR workbench |
| 3.12R3 | 061019 | JE | - Port added : ATMEL Atmega644, IAR-compiler |
| 3.12R2 | 060823 | JE | - New port for ARM AT91SAM7 added. |
| 3.12R1 | 060623 | JE | - New port for ARM LH754XX added. |
| 3.12R0 | 060515 | JE | - Support for external flash area added. |
| 3.10R11 | 060113 | JE | - New port for M32C and Renesas NC308 compiler added. |

| | | | |
|---|---|---|---|
| 3.10R10 | 051024 | JE | - New port for ARM LPC2XXX added. |
| 3.10R9 | 050906 | JE | - New devices added to M32C port. |
| 3.10R8 | 050705 | JE | - Command line options reworked. |
| 3.10R7 | 050511 | JE | - New port for Mitsubishi M32C added. |
| 3.10R6 | 050420 | JE | - Detailed explanation added for how to use external flash devices. |
| 3.10R5 | 050411 | JE | - M16C port: Mentioned, that FixVect.xxx may need to be modified |
| 3.10R4 | 050224 | JE | - New devices added to M16C port |
| 3.10R3 | 050211 | JE | - New devices added to M16C port |
| 3.10R2 | 040729 | JE | - Explanation of FLASH_RELOCATECODE improved |
| 3.10R1 | 040527 | JE | - Macros added to configuration chapters<br>- Available ports added<br>- Updater revised |
| 3.10R0 | 040525 | JE | - Port added : ATMEL ATmega128, IAR-compiler |
| 3.00R8 | 040415 | JE | - Port added: Mitsubishi M16C, TASKING-compiler |
| 3.00R7 | 040406 | JE | - Interrupt processing explained<br>- Complete revise |
| 3.00R6 | 040330 | JE | - Adapted to new version of PC program |
| 3.00R5 | 040312 | JE | - FLASHConf.h added |
| 3.00R4 | 040128 | JE | - Routines of USER.c added to chapter « How to port »<br>- ENABLE_TRANSMITTER and DISABLE_TRANSMITTER added<br>- Flash routines changed |
| 3.00R3 | 030805 | JE | - Port added: Mitsubishi M16C, NC30-compiler |
| 3.00R2 | 030521 | JE | - Port added: ARM AT91, IAR-compiler |
| 3.00R1 | 030411 | JE | - Port added: Mitsubishi M16C, IAR-compiler |
| 3.00R0 | 030306 | JE | - Initial release |

# Contents

# 1. About this document

This guide describes how to install and use the ***emLoad*** software for embedded applications.
***emLoad*** consists of two parts: The bootstrap loader (BTL) and the ***HEXLoad*** software for the PC.
Parts of the source code for the target hardware are listed and explained, especially those which may be adapted to the target processor and the actually used FLASH memory.

## 1.1. Assumptions

This guide assumes that you already have a solid knowledge of the following:

- The software-tools used to build your application (assembler, linker, "C"-compiler)
- The C-language
- The target processor

If you feel your knowledge of C is not good enough, we recommend *The C Programming Language* by Kernighan and Richie, which describes the standard in C-programming and in newer editions also covers ANSI C.

## 1.2. Typographic Conventions for Syntax

This manual uses the following typographic conventions for syntax:

Regular size Arial for normal text

`Regular size courier for text that you enter at the command-prompt and for what you see on your display`

`Regular size courier for system-functions mentioned in the text`

```
Reduced size courier in a frame for
program examples
```

**Boldface Arial for very important sections**

*Italic text for keywords*

## 1.3. Glossary

The following table shows the abbreviations used in this manual:

| Abbreviation | Meaning |
|---|---|
| BTL | Boots trap loader |
| UART | Universal asynchronous receiver transmitter |
| SFR | Special function register |

# 2. Introduction to *emLoad*

## 2.1. What is *emLoad*

***emLoad*** is a software which allows program updates and verification in embedded applications via serial interface. The software consists of a Windows program and a program for the target application (BTL).
The only things required are an embedded application with a FLASH-type memory for program storage, a communications interface (type RS-232) and the software for application and PC: ***emLoad***.

## 2.2. Function of the software

After RESET the BTL is started instead of the application program. The BTL waits for a configurable time (default .5 sec.) for a data frame from the PC. If the communication with the PC times out, the BTL checks if a valid application is in the flash memory. If this is so, it is started.
The only difference for the application program when running with bootloader is that it is located in different areas of the flash memory and that it is not started right after RESET; but with a certain delay. The application program is otherwise not affected by the BTL and has all resources available. It can use the entire RAM of the target system and can use interrupts without limitation.

## 2.3. Availability and FLASH devices

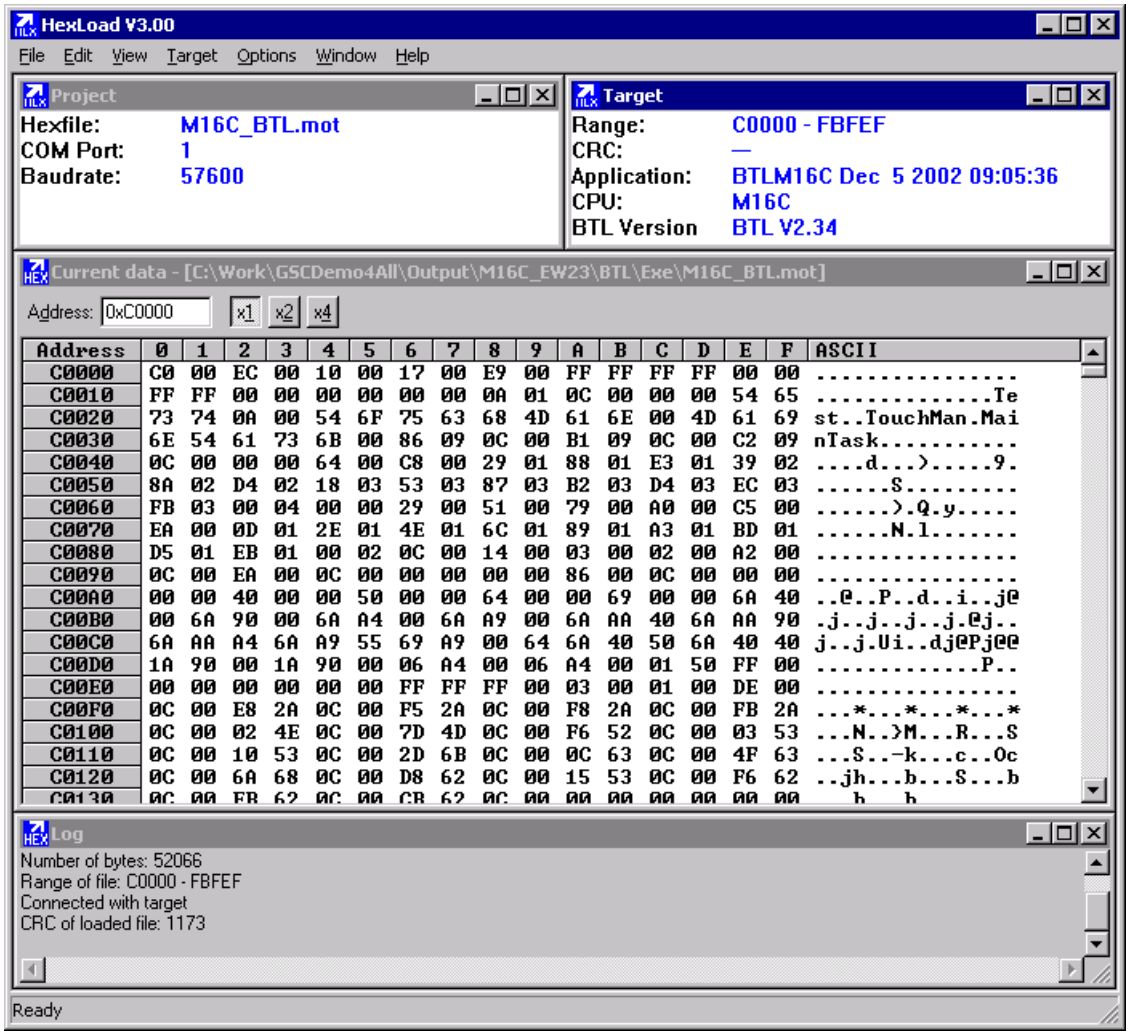
The software is written completely in ANSI-"C" and can therefore be used on most microcontrollers. The only things needed to port the BTL for a particular application are: a "C"-module for access to the peripherals of the microcontroller and a "C"-module containing the programming algorithm for the FLASH-memory chip(s). For latest information about supported devices, please visit our web site. Ports for other microcontrollers can be made in short time.

## 2.4. Configuration

The target program can be configured to meet the requirements of the application by modifying a configuration file `BTLCONF.H`. Adjustable are baudrate, application name, system frequency of the CPU, interface selection, reset delay and the optional password.
Further customizations - i.e. special initialization of the hardware - can be achieved by modifying the source code. Using ***emLoad*** with an external FLASH chip is no problem, even automatic recognition of the used FLASH chip is possible through reading of device- and manufacturer ID.

# 3. PC-program: *HEXLoad*

The PC-Program is very easy to use. Any hex file can be loaded and transferred to the BTL for a program update. ***HEXLoad*** is a 32 bit Windows application and can be started from the Explorer or from the command line. The following is a "screen shot" of the PC-program with loaded hex file programming a target chip:




## 3.1. Installation

There is no special installation necessary. Just copy the `HEXLoad.EXE` file into any sub-directory you like. No special DLLs or runtime libraries are needed.

| **Attention:** | ***HEXLoad*** is a 32 bit application and works under Windows NT, 2000 and XP! |
|---|---|

## 3.2. Starting *HEXLoad*

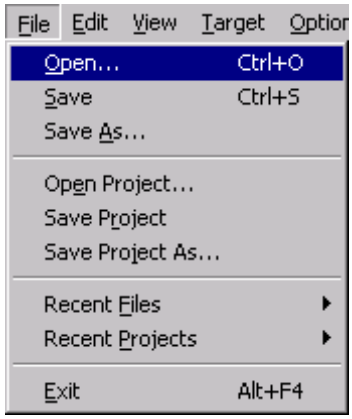
Start ***HEXLoad*** like you are used to start any Windows application.

| **Attention:** | Make sure you only start one instance of ***HEXLoad***. |
|---|---|

# 3.3. Menu items

***HEXLoad*** is easy to use because design of the menu items is similar to many Windows applications. Following the different items are explained in detail.

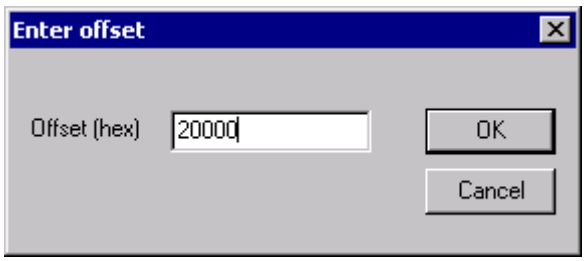
## 3.3.1. File Menu




| | |
|---|---|
| Open: | Open any hex file saved on disk or network |
| Save: | Save changed file |
| Save As: | Save changed file under different name |
| Open Project: | Open a project file saved on disk or network |
| Save Project: | Save project file |
| Save Project As: | Save project file under different name |
| Recent files: | List of most recently used hex files |
| Recent Projects: | List of most recently used projects |
| Exit: | Exits HEXLoad software |

## 3.3.2. Edit Menu

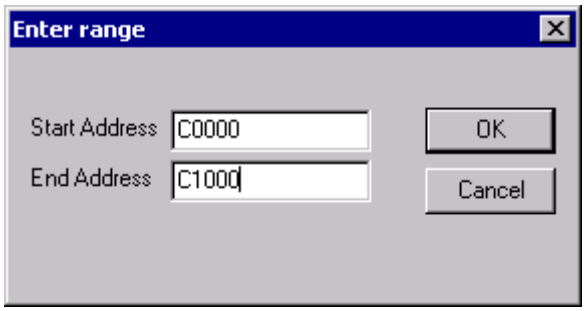
Relocate: Relocate target program


Enter the new desired offset to relocate the program.

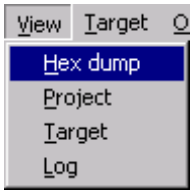
| | |
|---|---|
| **Attention:** | Use this option with care! A relocated program may not work at its new offset! |

Delete range: Deletes the given range from the loaded data



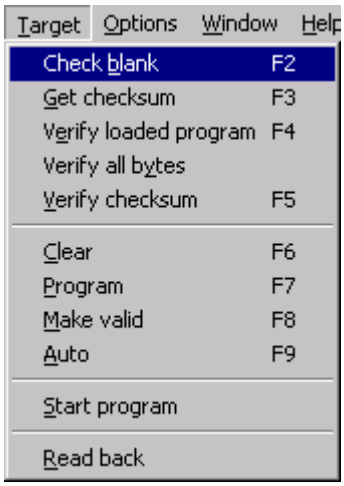

Enter the range of data to delete.

## 3.3.3. View Menu

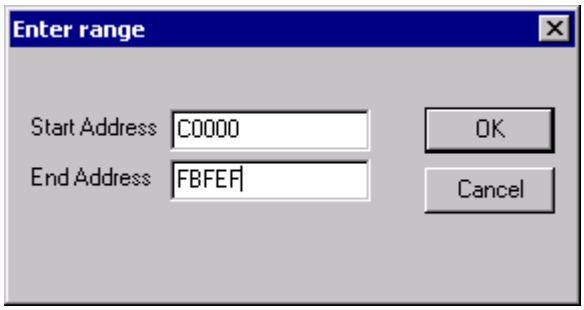



| | |
|---|---|
| Hex dump: | Opens a dump window with the possibility of editing the data. |
| Project: | Opens the project window containing the connection parameters and the name of the loaded hex file. |
| Target: | Opens the target window. It shows whether ***HEXLoad*** is connected to a target or not. |
| Log: | Opens the log window. ***HEXLoad*** logs all operations to this window. |

## 3.3.4. Target Menu



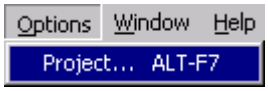

| | |
|---|---|
| Check Blank: | Checks if the user area of FLASH chip is blank |
| Get Checksum: | Calculate Checksum of target user area |
| Verify loaded PG: | Verifies if every byte of the loaded program is identical in the target user area |
| Verify all bytes: | Verifies all bytes of the target user area if is identical with the loaded program |
| VerifyChecksum: | Verifying checksum of loaded hex file and target user area |
| Clear: | Erase user area of FLASH |
| Program: | Program the loaded file into the user area of the FLASH. This works only if the FLASH is blank |
| Make Valid: | Validate application program. The application program will be executed automatically only if it has been declared valid |
| Auto: | clear, program, verify CRC, make valid if CRC is O.K. |
| Start Program: | Start application program, leave BTL |
| Read back: | Read data from FLASH memory after the following dialog: |



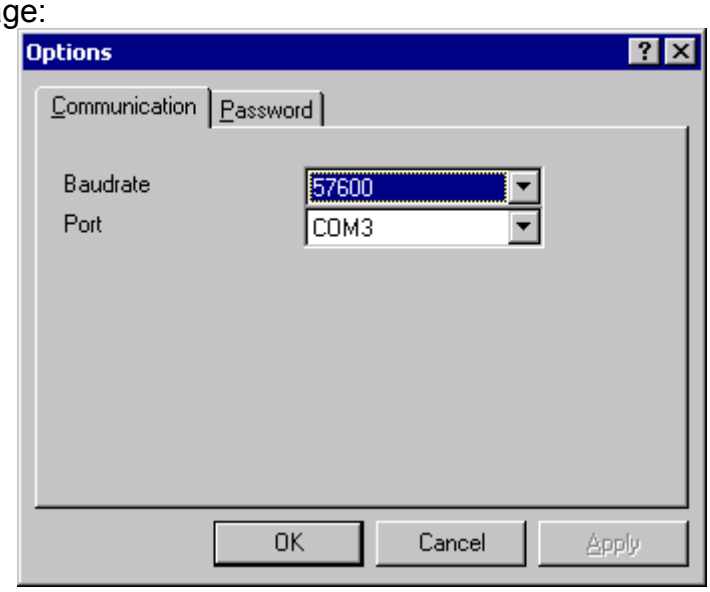

Enter start and end address and press ok to read any area of the target

## 3.3.5. Options Menu

Project: Opens a tab control with two pages to change the project properties.

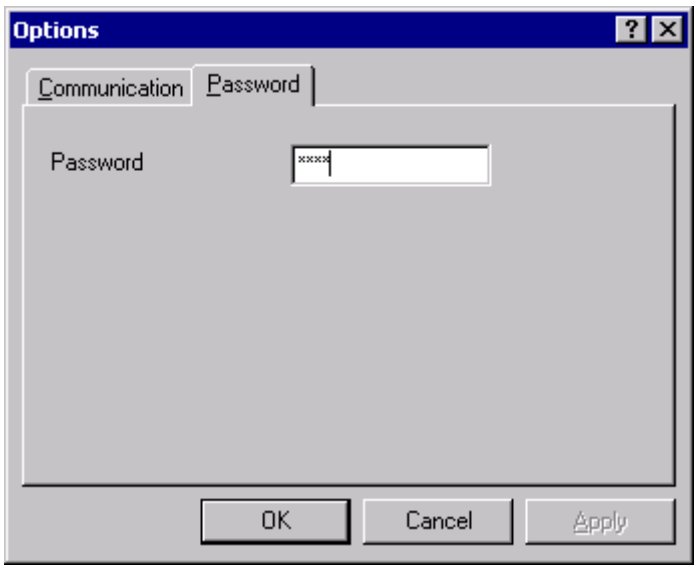



Communication page:




Change baudrate and ComPort to the required values.

Password page:




If a password is required by setting in the `Config.H` file you have to enter it here to get access to FLASH application memory area.

# 3.4. Command line options

## 3.4.1. Table of commands

| | |
|---|---|
| `auto` | Clear, program, verify and make valid. |
| `baudrate<BAUDRATE>` | Set baudrate. |
| `checkblank` | Checks if target memory is blank. |
| `clear` | Clear target. |
| `com<PORT>` | Set COM-port. |
| `exit[,<TIMEOUT>]` | Finish application after job. Waits TIMEOUT ms for a connection, 0 for endless. |
| `makevalid` | Makes the target valid. |
| `password<PASSWORD>` | Set password. |
| `program` | Write current data into Target. |
| `readback<STARTADR-ENDADR>` | Reads a range of bytes from target. |
| `relocate<VALUE>` | Relocates loaded data. |
| `saveas<FILENAME>.<EXT>` | Save data file (Use *.mot or *.hex as EXT). |
| `verify` | Evaluates if target checksum is the same as from the current data. |
| `?` | Shows all available commands. |

- All commands are identical with the commands in the menu bar.
- All commands are processed from left to right.
- If using `-exit` Hexload will stop execution if any error occurs. The return code in this case is != 0.

## 3.4.2. Examples

```
Hexload.exe flasher2_v160.mot -passwordAW -auto –exit
```

In this example Hexload first reads the file flasher2_v160.mot, sets the password to "AW", execute the commands clear, program, verify and makevalid and finish execution.

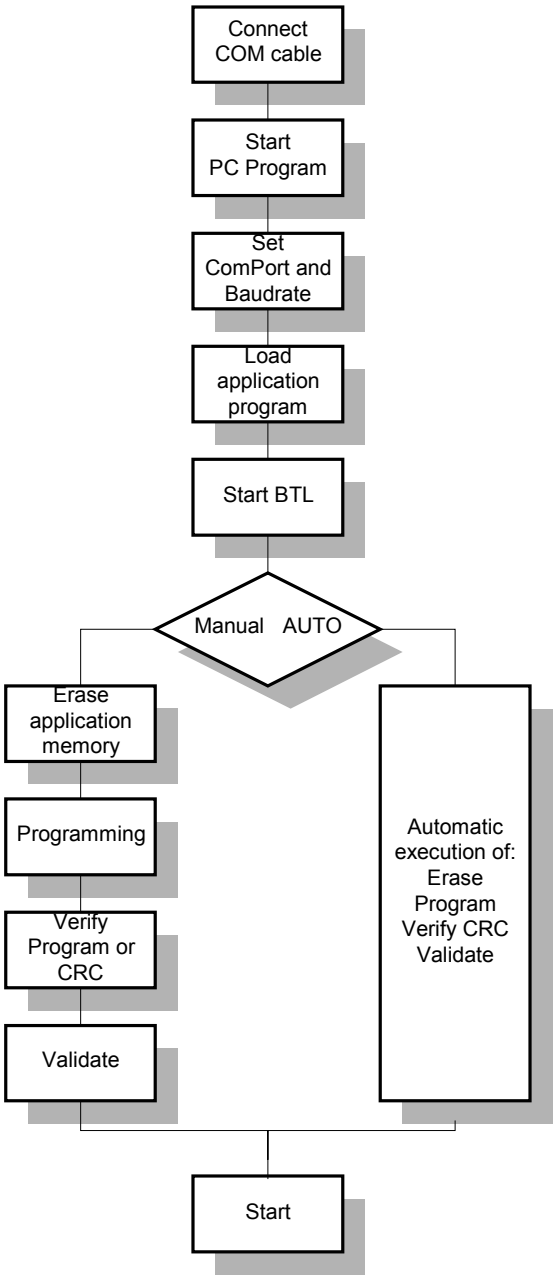
```
Hexload.exe -readbackFC0000-FC1000 -saveasC:\TEMP\RANGE.MOT
–exit
```

Reads the area 0xFC0000-0xFC1000, saves it as c:\temp\range.mot and finish execution after job is done.

# 3.5. Using the ***emLoad*** software

Program updates via serial communication port is possible by using the Windows program HexLoad.exe.

To update an application program the following steps have to be executed



Serial communication

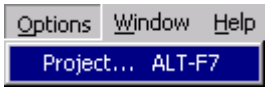
Connect the communication cable to your hardware and a COM port of your PC.

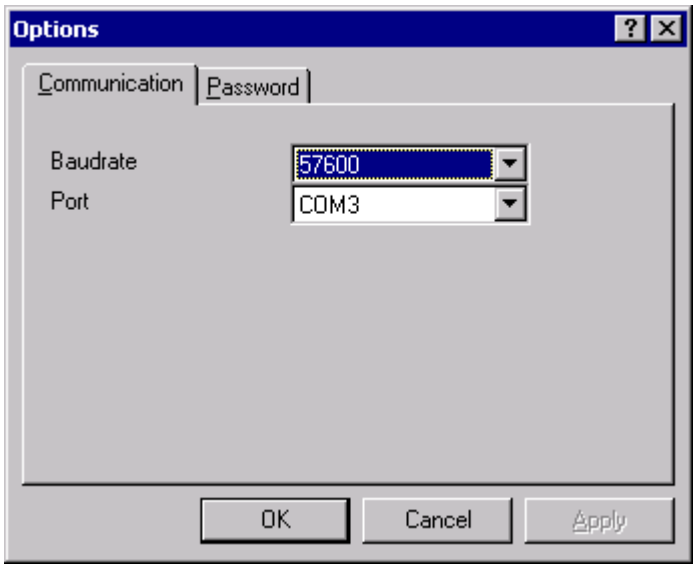
Starting ***HEXLoad***

Start ***HEXLoad*** like you are used to start any Windows application.

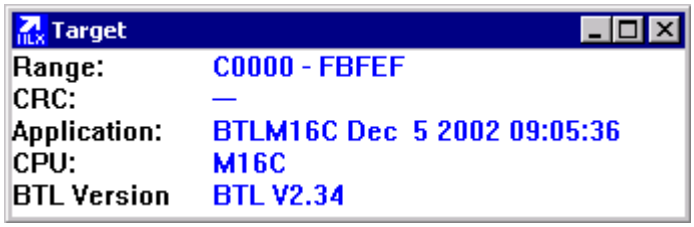
Com port and baudrate

Check the settings for communication port and Baudrate by Menu `Options - Communication`.




Select the necessary Baudtate and ComPort, click OK to accept.
If Communication is OK, in the target window of HEXLoad you should see detailed information about Range, Application, CPU and the BTL software version like shown in the following screen shot:



Load application program

Any Intel-hex or Motorola-hex file can be opened. Just use the `File - Open` command or shortcut `Ctrl+O` and select the necessary file like you are used to.

Starting the BTL

The BTL will be started and activated after each RESET. It will remain active until the application program is started. During this time it is able to communicate and to execute received commands. As soon as the application software is started it is no longer possible to communicate with the BTL. The BTL can be restarted anytime by the application software.

The application software will be started by the following conditions:

- Memory contains a valid program and no communication with ***HEXLoad***. (Communication time-out period usually 0.5 sec.)
- immediately after a START command

Erasing the memory

After receiving the ERASE MEMORY command the BTL sends an acknowledge and erases the complete FLASH memory excepting the boot-block containing the BTL. Erasing the memory can take up to a few seconds depending an the size of the memory. A message about success or not is send back via the serial interface.
By erasing the memory the application program is marked as invalid and the BTL remains active.

Programming

Programming of the FLASH memory is done by transmitting Hex file in data packages via serial communication. Each line is started with the PROGRAM command. If the hex line was received without errors the allocation of all data inside the application memory is checked and immediately programmed. A message about success or not is send back via the serial interface.
Programming a complete Hex file is simply done by the Menu: `Target - Program` or by pushing <F7>.

Verify program update

To verify the program update you have two options: Verifying the complete program or verifying the checksum.

Verification of the application program byte by byte:

It is possible to verify the content of the program memory. The result will be transmitted via serial interface.
Sending of hex files and evaluation of the BTL messages is managed by the ***HEXLoad*** program. Use the Menu funktion `Target - Verify` <F4>.

Verifying checksum:

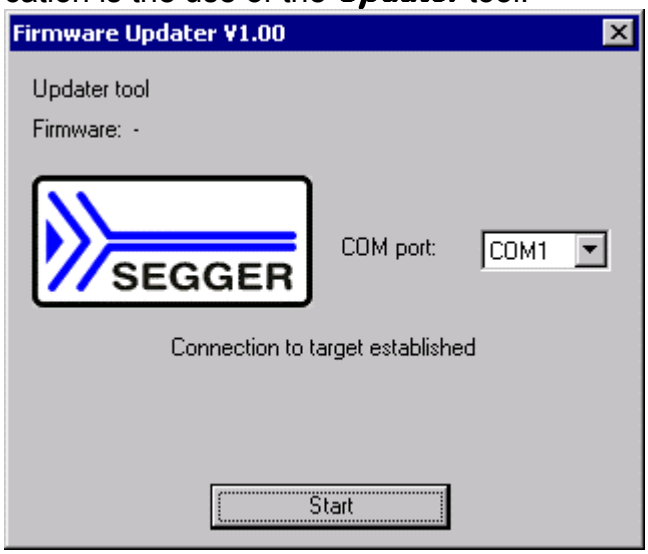
***HEXLoad*** calculates the checksum of the available program memory of the FLASH Memory block by addition byte wise. The result is send as a 16bit Word. Just use the Menu function `Target - Get Checksum` (<F3>).

Validating the program

To start a new program automatically after RESET it has to be validated. Either by automatic mode or manually by the ***HEXLoad*** Menu command `Target - MakeValid` (<F8>). Otherwise the BTL will wait for next commands via the serial interface and not automatically start the application program. The topmost 16 bytes of the user memory area are reserved (4 bytes used) for this purpose.

# 4. PC-program: *Updater*

An easy way to update a target via serial communication port with a new application is the use of the ***Updater*** tool:




The ***Updater*** is an add-on and not part of the ***emLoad*** software. It has been designed to give the end user the possibility of an easy firmware update without using the ***HEXLoad*** program. It is shipped as source code, which is required to modify and to recompile the tool.

## 4.1. How to exchange the firmware

The firmware to be used by the ***Updater*** is embedded in the EXE file of the tool. To exchange the firmware, the ***Updater*** has to be recompiled. The following steps show how to add new firmware:

- Compile and link the application program to be used and generate a Motorola 'S' record file.
- Use the tool `Bin2C.exe` shipped with the ***Updater*** to convert the Motorola 'S' record file to a 'C' file.
- Rename the 'C'-file to `Firmware.c` and replace the file `Firmware\Firmware.c` of the ***Updater*** with the new one.
- Open the ***Updater*** workspace.
- Open the include file `Main.h` and adapt the configuration settings to your needs. The `BAUDRATE` macro defines the baudrate used to communicate with the target. The `FIRMWARE` macro defines the text shown in the application window right of 'Firmware'.

```
#define BAUDRATE 57600
#define FIRMWARE "-"
```

- Rebuild the project. The 'ready to use' ***Updater*** with the new firmware can be found under `Output\Updater\Release\Updater.exe`.

## 4.2. How the *Updater* works

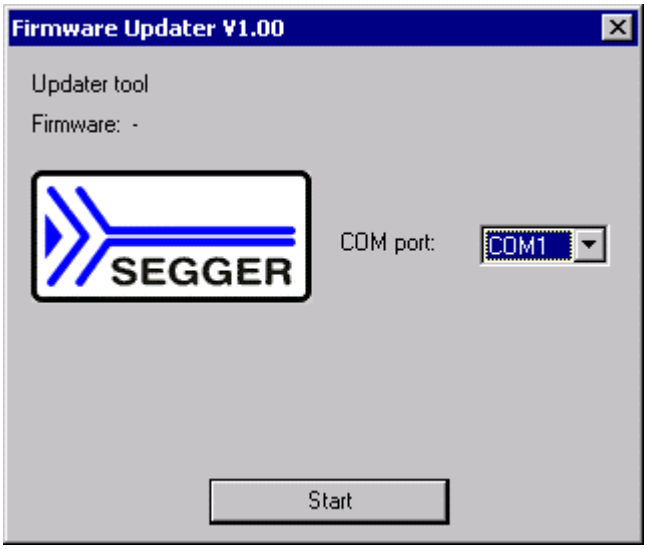
After the ***Updater*** is started it shows the dialog shown above which gives the user the possibility to set up the COM port of the PC. After the ***Updater*** tool gets contact with the target it shows a notification message in the application window. Now the user should only press the 'Start' button or the <ENTER> key

to update the target. The tool now clears the flash, programs the new file into the flash, makes a CRC check and makes the target application valid.

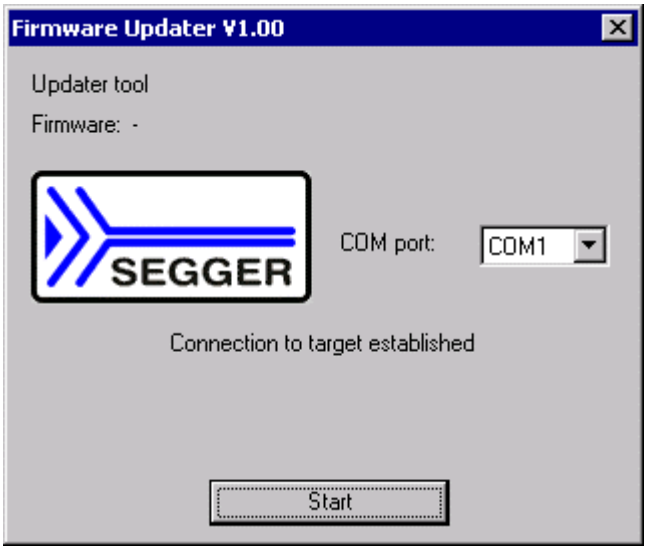
## 4.3. Using the ***Updater***

The following steps show how to use the ***Updater***.

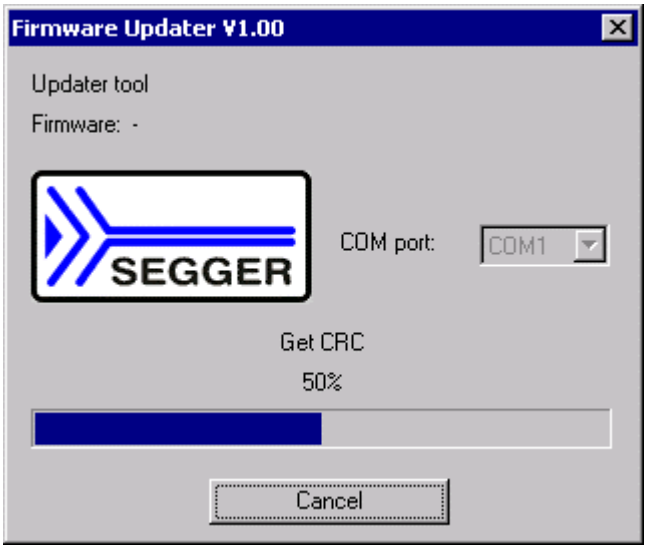
- Connect the target UART with the desired COM port of the PC.
- Start the ***Updater***. and select the desired COM port.

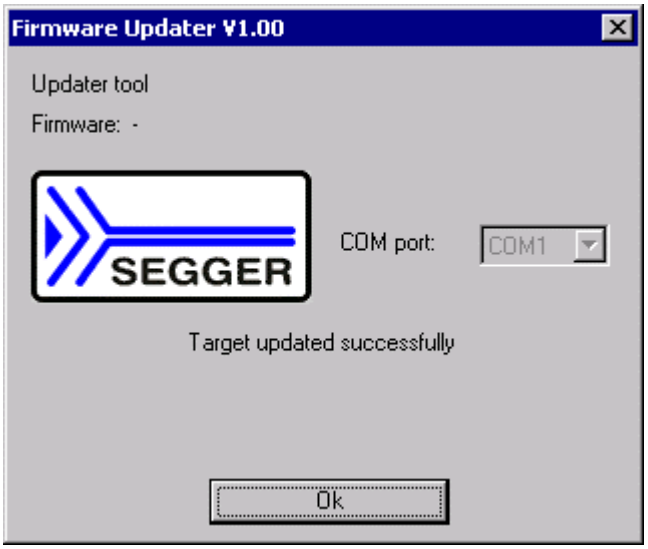



- Connect the target to the power supply.



- Press the ‘Start’ button or the <ENTER> key.



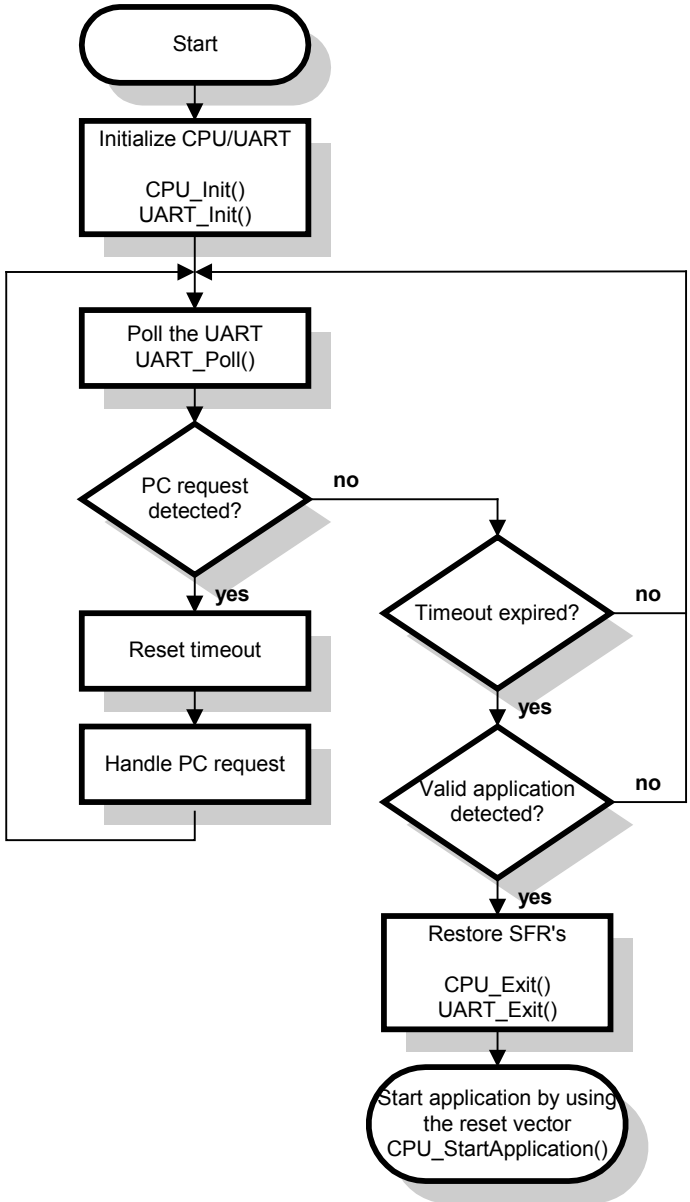

- Now further targets can be updated.

# 5. Understanding the BTL

After RESET, the BTL is started. It then tries to detect a communication request from the PC via UART. If the PC has been detected, the BTL keeps running and the user can use ***HEXLoad*** to program, read back and erase the flash. If there is no communication request or the user closes ***HEXLoad,*** the BTL checks if there is a valid application program in the flash. If a valid application program is present, the BTL starts it using the reset vector of the application program.

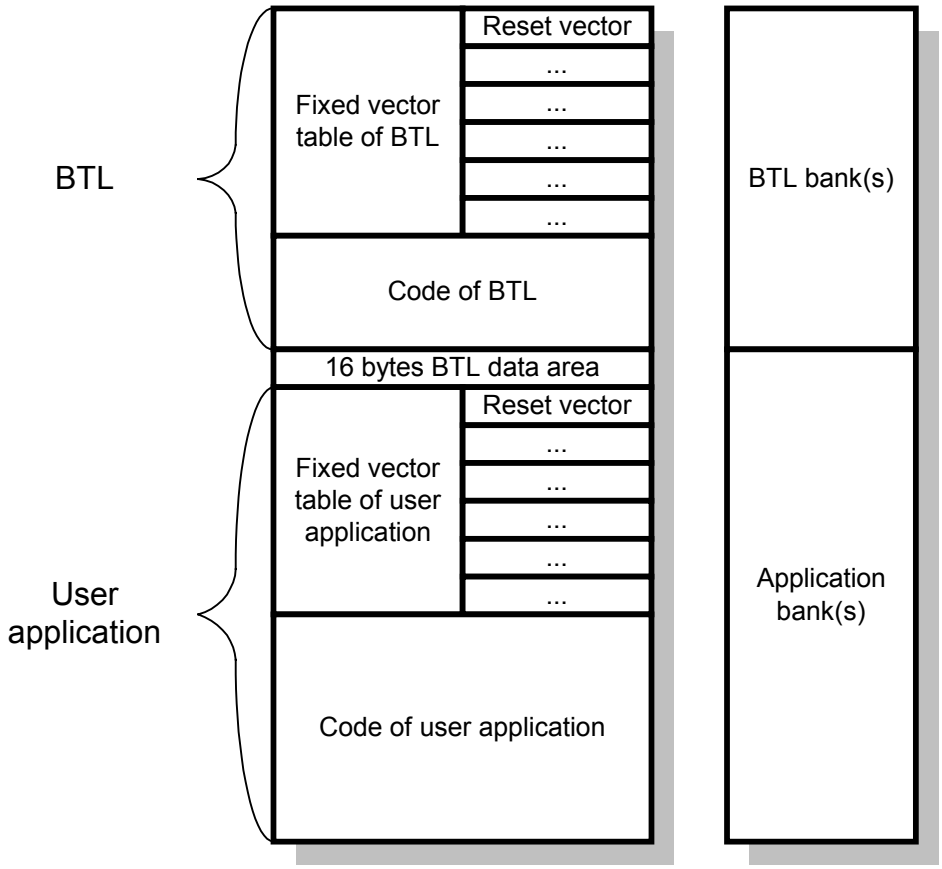
## 5.1. Flowchart

The diagram below shows the flowchart of the BTL software:



## 5.2. Memory map

The diagram below shows a typical memory map used with the BTL:




The upper 16 bytes of the user area are reserved by the BTL data area. This area typically contains the validation code which is used to tell the BTL that a valid application program is in the flash memory.

## 5.3. Interrupts

The BTL itself does not use interrupts.
Interrupts can be used in the application program without limitation.
In order to achieve this, different strategies have to be used for different CPUs.

### 5.3.1. Different types of interrupt processing

There are basically 3 different types of interrupt processing:

1. Interrupt vectors with fixed base address
2. Interrupt vectors with variable base address
3. Vector less interrupt processing

The following describes how to manage these systems in ***emLoad***.

#### 5.3.1.1. Fixed vectors

The memory area of the vector table typically contains the reset vector. The vector table should be part of the BTL which should include code to forward the interrupts to the addresses defined in the vector table of the application program. This design of interrupt handling is a older CPU design used for example by NEC K0 , NEC K4 or 6502 CPUs. The PC is loaded with the contents of a fixed address:
PC = (Addr)

#### 5.3.1.2. Variable vectors

Systems using this modern type of interrupt handling works well with the a BTL. Typically the startup code of the application program sets the base address of the vector table. The BTL needs no code to forward interrupts. This type of interrupt handling is used for example by Renesas M16C and M32C.
The PC is loaded with the contents of the base address + an offset:
PC = (Base Addr + Off)

#### 5.3.1.3. Fixed address

The third concept of interrupt handling is setting the PC register to a specific value. If an interrupt occurs the code at a fixed address depending on the interrupt number is executed. Sample systems using this type of interrupt handling are ARM and NEC V850.
The PC is loaded with a fixed address:
PC = Addr

Code in RAM

If using a system like an ARM CPU the application program is often executed from RAM. In this case the BTL needs no code for interrupt forwarding. The startup code of the application program first copies the application into the desired RAM area and then executes it from RAM.

Code in ROM

If using a system like a NEC V850 the application program is typically executed from ROM. In this case the BTL should forward the interrupts by including a jump to the interrupt code of the application program.

## 5.4. Reset

There are basically 2 ways of behavior after RESET: using a reset vector containing the address to be loaded into the PC (Fixed vector) or starting the execution at a fixed address.
To make sure the BTL is started after RESET, the BTL has to reside in the same bank as the reset vector or the start address of the CPU. The application program has to be linked so that the reset vector / start address has been moved down by the size of the BTL bank + 16 bytes.

### 5.4.1. Fixed vector

This method loads after RESET the contents of the PC register from a fixed address, the reset vector and starts execution from the address pointed by it. In this case the application program should include its start address located at the reset vector address of the application program. The BTL uses this address to start the application program. Samples for starting a CPU by this way are Renesas M16C or M32C.

### 5.4.2. Fixed address

This kind of CPUs starts execution after RESET at a fixed address. In this case the application program should include code at the start address plus the size of the BTL bank to jump to the entry point of the application program. Examples for this type of RESET processing are NEC V850 or ARM.

# 6. Configuration

This chapter explains the configuration options of ***emLoad***. The folder `Config` contains 2 configuration files:

- The file `BTLConf.h` contains general configuration options.
- `FLASHConf.h` configures the flash driver.

## 6.1. Configuring BTLConf.h

The following table gives an overview of the configuration macros of emLoad:

| Macro | Description |
|---|---|
| APPNAME | Application name displayed by HexLoad. |
| BTL_HUGE | Defines the keyword for using huge pointers. |
| BTL_RW_U32NO | Activates the use of functions for reading and writing 32 bit values from/to BTL data.packets. Default value is 1 (use functions). |
| BTL_WAIT0_MS | Time to wait for a communication request. |
| BTL_WRITE_BLOCK_SIZE | Number of bytes for accessing the flash. |
| DISABLE_TRANSMITTER | Disables the transmitter after sending data. |
| ENABLE_TRANSMITTER | Enables the transmitter before sending data. |
| FEEDWATCHDOG | Triggers a watchdog. |
| FLASH_USER_LEN | Defines the length of the user area including the reserved bytes. |
| FLASH_USER_START | Defines the beginning of the flash user area. |
| FLASH_USER_RESBYTES | Defines the number of bytes used by the BTL in the application area. The default value is 16 bytes. |
| PASSWORD | Password used to communicate with target. |

The configuration file BTLConf.h is included by BTL.h, the main include file of ***emLoad***. The following items have to be defined in this file:

### 6.1.1. Application name

Description

The application name is displayed by ***HEXLoad*** in the target section.

Example

```
#define APPNAME "BTLM16C " __DATE__ " " __TIME__
```

### 6.1.2. Huge pointer

Description

The macro `BTL_HUGE` is used by the BTL to define flash memory pointers. If the used memory model of the compiler has short pointers (16 bit) per default and the flash memory is located above 0xFFFF the definition of this macro is needed. The used keyword should also make sure, that the element pointed at is not limited to one 64 Kbyte page.

Example

```
#define BTL_HUGE __far
```

### 6.1.3. Use of functions for reading and writing 32 bit values

Description

The BTL needs to read/write 32 bit values from/to communication data packets. If the target works in little endian mode and non aligned addresses for 32 bit values are allowed, the use of the functions is not required. In this case the use of the functions can be disabled. This reduces the size of the BTL.

Example

```
#define BTL_RW_U32NO 0 /* Disable the use of the functions */
```

### 6.1.4. Wait time after reset

Description

The BTL waits for the defined time for a data frame from the PC. If it does not receive a data frame from the PC within this time, the BTL starts the application by accessing the RESET vector of the application.

Example

```
#define BTL_WAIT0_MS 500
```

### 6.1.5. Write block size

Description

Some flash devices can not be programmed byte by byte but block by block. This macro specifies the number of bytes for accessing the flash.

Example

```
#define BTL_WRITE_BLOCK_SIZE 16
```

### 6.1.6. Transmitter enable / disable

Description

If the receive and the send lines (Rx/Tx) use the same data line, it may be necessary to enable the transmitter before sending data and to disable it when transmission is completed. For this purpose the macros ENABLE_TRANSMITTER() and DISABLE_TRANSMITTER() can be defined. The macro ENABLE_TRANSMITTER() is executed before sending a data packet to the PC. After sending the entire data packet the macro DISABLE_TRANSMITTER() is executed. If these macros are used, ensure that the function UART_Send1() does not return until the complete byte has been sent.

Example

```
#define ENABLE_TRANSMITTER()  P10D |=  (1<<6); P10  &= ~(1<<6)
#define DISABLE_TRANSMITTER() P10  |=  (1<<6)
```

### 6.1.7. Feed watchdog

Description

This macro can be used to trigger a watchdog. It should not contain a function call, because the macro also will be executed from some parts of the BTL relocated to RAM.

Example

```
#define FEEDWATCHDOG P6 |= 0x80
```

## 6.1.8. User flash area

Description

The user flash area defines the area to be managed by emLoad. If the used CPU has internal flash, emLoad also supports an additional external user flash area. You have to define 2 macros to specify a user flash area:

- FLASH_USER_START
  Defines the beginning of the user area.
- FLASH_USER_LEN
  Defines the length of the user area including the reserved bytes.

A second area can be defined as follows:

- FLASH_USER1_START
  Defines the beginning of the external user area.
- FLASH_USER1_LEN
  Defines the length of the external user area.

Example

Your flash area reaches from 0x40000 to 0x7FFFF and contains the following sectors:

0x40000 – 0x4FFFF
0x50000 – 0x5FFFF
0x60000 – 0x6FFFF
0x70000 – 0x77FFF
0x78000 – 0x7FFFF

The reset vector of your CPU is located in 0x7FFFC – 0x7FFFF. So the BTL has to reside in the upper sector and the user flash area reaches from 0x40000 – 0x77FFF. Your BTLConf.h should include the following entries:

```
#define FLASH_USER_START 0x40000
#define FLASH_USER_LEN   0x38000
```

Add. information

Some ports of ***emLoad*** contain default values of the user flash area depending on the used CPU. In this cases the definition of the flash user area is not required, if the whole flash should be available for emLoad.

## 6.1.9. Number of data bytes

Description

To mark the application program as ‘valid’ the BTL writes a validation sequence at the end of the application area. This macro defines the number of data bytes reserved for the BTL.

Example

```
#define FLASH_USER_RESBYTES 256
```

### 6.1.10. Password

Description

If you need to protect your target with a password you can define it in BTLConf.h as an text replacement macro. When using ***HEXLoad*** the user has to specify this password under “Options/Password”. The password would be evaluated by emLoad. If the password does not match ***emLoad*** would not communicate with ***HEXLoad*** .

Example

```
#define PASSWORD "abc"
```

## 6.2. Configuring FLASH_Config.h

The following table gives an overview of the configuration macros of emLoad:

| Macro | Description |
|---|---|
| FLASH_U8 | Definition of a 8 bit unsigned value. |
| FLASH_U16 | Definition of a 16 bit unsigned value. |
| FLASH_U32 | Definition of a 32 bit unsigned value. |
| FLASH_HUGE | Used to define pointers to the flash memory. |
| FLASH_RELOCATECODE | Defines if the flash module should copy the routines for writing and erasing into RAM. Default is 1, which means the routines will be copied into RAM. |

The configuration file FLASH_Config.h is included by the flash memory modules of ***emLoad***. The following items have to be defined in this file:

### 6.2.1. Basic data types

Description

The following data types needs to be defined: FLASH_U32, FLASH_U16 and FLASH_U8. The macro FLASH_HUGE is used within the flash modules to define pointers to the flash memory. For details please refer to the chapter “Configuring BTLConf.h”.

Example

```
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
```

### 6.2.2. Huge pointer

Description

The macro FLASH_HUGE is used within the flash modules to define pointers to the flash memory. It has the same function as the macro BTL_HUGE. For details please refer to the chapter “Configuring BTLConf.h”.

Example

```
#define FLASH_HUGE BTL_HUGE
```

### 6.2.3. Relocate flash routines

Description

This macro defines whether the code for writing and erasing the flash is copied into RAM or not. If the BTL executes code in the same flash module as the sectors to be modified, the code needs to be relocated. This is required, because executing code in a flash module is not possible while it is in rewriting mode.
If relocation is enabled, the BTL uses a buffer for code relocation.
Before writing to the flash or erasing a sector, the flash module relocates (copies) the right routine to the buffer and starts execution at the buffer, typically by a subroutine call.
This can be problematic for different reasons. Some of these reasons are:

- CPU has separate data and instruction caches (e.g. ARM9, MIPS).
- Compiler generates code with absolute jumps.
- Thumb/ARM mode switches on ARM CPUs (When code executes in thumb mode and "interwork" is used).
- Compiler generates code which does not fit in buffer area.
- Compiler generates code which references PC-relative data located far away from the routine.
- Interrupts occur during execution of the relocated code, while flash module is not ready (code required for ISR may not be accessible).

Each port of the BTL has been tested with its configuration as it has been shipped. If the relocation does not work in a different environment, please check the following: Look at the assembly output and make sure that

- the relocated code does not contain absolute jumps.
- the buffer is big enough.
- there is no other problem which prevents the relocated routine from executing properly.
- if the CPU has separate data and instruction caches, the instruction cache is disabled.

Example

```
#define FLASH_RELOCATECODE 1
```

# 7. Generic program modules of the BTL

The BTL has been designed to be easily portable to other CPU cores and Flash devices. It has therefore been divided into different modules:

BTL core: `mainBTL.C`

The module actually containing the BTL:

Application specific portion: `User.C`

This module is responsible to supply application specific behavior like a special init. Per default, the routines contained herein have no functionality.

Handling the communication: `CRCCCITT.C`

The module contains code to calculate the 16-bit CRC checksum.

Managing multiple flash areas: `FlashMap.C`

This module is required if more than one flash area needs to be handled (internal and external flash).

# 8. How to port

The only thing you have to do is to adapt the CPU and the UART-module located in the PORT-folder. When starting the BTL the right processor mode has to be configured, a timer has to be started and the UART communication has to be enabled. The following chapter explains the routines called by the BTL.

## 8.1. CPU related routines, CPU.c

### 8.1.1. CPU_Exit()

Description

This routine has to set all special function registers modified in the CPU module (in CPU_Init(), CPU_Poll() or CPU_StartApplication()) back to their initial state after reset. This is necessary, because the application program expects all sfrs to be in the state documented as “after RESET”.

Prototype

```
void CPU_Exit(void);
```

### 8.1.2. CPU_GetName()

Description

This routine has to return a pointer to the CPU-name. It would be shown by ***HEXLoad*** in the target description.

Prototype

```
const char * CPU_GetName(void);
```

Example

```
char* CPU_GetName(void) {
  return "M16C";
}
```

### 8.1.3. CPU_Init()

Description

This routine has to make sure the right processor mode has been selected, the clock mode has to be configured properly and a timer has been started. The timer would be used by CPU_Poll to notice if a millisecond has been passed.

Prototype

```
void CPU_Init(void);
```

### 8.1.4. CPU_Poll()

Description

This routine is called regularly from the main loop of the BTL. It is used as time base for the BTL and has to notice if a millisecond has been elapsed. The time

is used to determine when the communication to the PC is timed out and the application has to be started. There are two options:

- Precise timing using a hardware timer.
- Simple timing using a counter.

Prototype

```
int CPU_Poll(void);
```

Return value

1 if a millisecond has been elapsed, otherwise 0.

Example

The following sample uses a hardware timer:

```
int CPU_Poll(void) {
  if (TB0IC & (1<<3)) { /* Check if interupt request flag has been set */
    TB0IC &= ~(1<<3);   /* Clear interupt request flag                  */
    return 1;           /* A ms has elapsed */
  }
  return 0;             /* No new ms has elapsed */
}
```

The following sample uses a counter:

```
static int _Cnt;

int CPU_Poll(void) {
  if (++_Cnt == 50) {
    _Cnt = 0;
    return 1;           /* A ms has elapsed */
  }
  return 0;             /* No new ms has elapsed */
}
```

## 8.1.5. CPU_StartApplication()

Description

This routine would be called to start the application program program. It is called under 2 circumstances:

- If the timeout time (BTL_WAIT0_MS, configured in BTLConf.h), has expired, no communication request from the PC has been detected and a valid application program has been programmed into the target.
- If the user chooses “Start Program” from the target menu from HEX-Load.

In dependence of the way of starting the CPU (using a reset vector or starting at a fixed address) the function should

- either jump to the address pointed by the reset vector of the application program (if using a reset vector)
- or jump to a fixed address, typically the start address plus the size of the BTL bank (if using a fixed address)

Prototype

```
void CPU_StartApplication(void);
```

Example

The following sample implementation uses a reset vector. It jumps to the address pointed by the reset vector of the application program:

```
typedef void (voidRoutine)(void);

void CPU_StartApplication(void) {
  (**(voidRoutine**)(FLASH_USER_START +
                     FLASH_USER_LEN – 16 - 4))();  /* use RESET vector */
}
```

The following sample implementation uses a fixed address. It jumps to the start address plus the size of the BTL bank:

```
typedef void (voidRoutine)(void);

void CPU_StartApplication(void) {
  (*(voidRoutine*)(FLASH_USER_START))();
}
```

# 8.2. UART related routines, UART.c

## 8.2.1. UART_Exit()

Description

This routine has to set all special function registers modified in the UART module (in UART_Init(), UART_Poll() or UART_Send1()) back to their initial state after reset. This is necessary, because the application program expects all sfrs to be in the state documented as “after RESET”. Before doing so, this routine needs to make sure that all bytes which should have been transmitted (by calling UART_Send1) have already been sent. Do not deinitialize the UART before the last byte has been transmitted!

Prototype

```
void UART_Exit(void);
```

Example

```
void UART_Exit(void) {
  while (!(UC1 & (1<<1))); /* Wait until TB empty    */
  while (!(UC0 & (1<<3))); /* Wait until Tx finished */
  UMR   = 0x00;            /* lock Sio, error reset  */
  UC1   = 0x00;            /* Lock Rx and Tx         */
  SRIC  = 0x00;            /* Disable interrupt      */
}
```

## 8.2.2. UART_Init()

Description

This routine has to enable the UART communication. It should be useful if the configuration macros of BTLConf.h would be used to configure the communication:

- BAUDRATE - Baud rate to communicate
- UARTSEL - Used to select the UART
- UPCLOCK - CPU frequency

Communication parameters:

- 8 data bits
- Odd parity
- 1 stop bit

Prototype

```
void UART_Init(void);
```

Example

```
void UART_Init(void) {
  UMR   = 0x00;        /* Lock Sio, error reset                  */
  UC0   = 0x10;        /* RTS/CTS disabled, clock divisor 1      */
  UBRG  = BAUDDIVIDE;  /* Calculated Baudrate                    */
  UC1   = 0x00;        /* Lock Rx and Tx                         */
  UMR   = 0x05         /* 8 Data                                 */
         +(0<<5)       /* 0: 0dd parity                          */
         +(1<<6)       /* 1: parity enable                       */
         +(0<<7);      /* 0: no sleep                            */
  UCON  = 0x00;        /* transmit-interrupt on buffer empty     */
  UC1   = 0x05;        /* enable reception and transmition       */
}
```

## 8.2.3. UART_Poll()

Description

This routine has to return if a character has been received.

Prototype

```
int UART_Poll(unsigned char * p);
```

| Parameter | Description |
|---|---|
| P | Pointer to an unsigned character to store the received character. |

Return value

1 if a character has been received, otherwise 0. If 1 the received character has to be stored to *p.

Example

```
int UART_Poll(uchar * p) {
  uint SioInput;
  if (!(SRIC & (1<<3))) {  /* Return if nothing to do */
    return 0;
  }
  SRIC &= ~(1<<3);         /* Clear interupt request flag */
  /*  Get new character */
  SioInput = URB;
  if (SioInput & 0xf000) {
    /* Error handling */
    char umr = UMR;
    UMR = 0x0;             /* Reset       */
    UC1 &= ~(1<<2);        /* Disable Rx  */
    UMR = umr;
    UC1 |=  (1<<2);        /* Enable Rx   */
  } else {
    /* Store received character to *p */
    *p = SioInput;
    return 1;
  }
  return 0;
}
```

## 8.2.4. UART_Send1()

Description

This routine has to send 1 character. Before sending the character is has to make sure the output buffer has been transmitted.
If the macros `ENABLE_TRANSMITTER()` and `DISABLE_TRANSMITTER()` are used, ensure that this function does not return until the complete byte has been sent.

Prototype

```
void UART_Send1(unsigned char c);
```

| Parameter | Description |
|---|---|
| C | Character to be send. |

Example

```
void UART_Send1(uchar c) {
  while (!(UC1 & (1<<1))); /* Wait until Transmitter Buffer empty    */
  UTB = c;                 /* Transmit data byte */
}
```

# 8.3. FLASH related routines, FLASH.c

If you need to manage CPU internal flash memory you have to include the empty flash driver FLASH.c to your project and to adapt the routines to your flash.

## 8.3.1. FLASH_EraseSector()

Description

This routine has to erase all sectors of the CPU internal flash.

Prototype

```
int FLASH_EraseSector(unsigned int SectorIndex);
```

| Parameter | Description |
|---|---|
| SectorIndex | Zero based index of sector to be erased. |

Return value

0 if sector has been erased successfully, otherwise 1.

## 8.3.2. FLASH_GetNumSectors()

Description

Returns the number of physical flash sectors.

Prototype

```
int FLASH_GetNumSectors(void);
```

Return value

Number of physical flash sectors.

## 8.3.3. FLASH_WriteAdr()

Description

This routine has to write the given array into the flash.

Prototype

```
int FLASH_WriteAdr(void * pDest,
                   const void * pSrc,
                   FLASH_U32 Len);
```

| Parameter | Description |
|---|---|
| pDest | Pointer to the destination address |
| pSrc | Source pointer. |
| Len | Number of bytes to be written. |

Return value

0 if all bytes have been written successfully, otherwise a pointer to the address on which the problem has been occurred.

## 8.4. User routines, USER.c

The routines located in this module have no functionality by default. They can be used for additional initialization or other purposes.

### 8.4.1. USER_Init()

Description

This routine is called after CPU_Init(). A typical use of this function could be additional hardware initialization.

Prototype

```
void USER_Init(void);
```

### 8.4.2. USER_Exit()

Description

This routine is called before the application program will be started. This routine can be used to restore the reset values to the registers used in USER_Init().

Prototype

```
void USER_Exit(void);
```

### 8.4.3. USER_Poll()

Description

This routine is called after CPU_Poll().

Prototype

```
void USER_Poll(void);
```

# 8.5. Using external flash routines

***emLoad*** includes a NOR flash chip driver for any erase sector oriented flash chip. It can handle most of the standard 29x or 28x flash chips.

## 8.5.1. Supported hardware

The NOR flash driver can be used with the popular NOR-flash devices. Currently the following devices are supported:

| Manufacturer | Device |
|---|---|
| AMD | Am29F002B/T |
| | Am29F004B/T |
| | Am29F008B/T |
| | |
| | Am29LV002B/T |
| | Am29LV004B/T |
| | Am29LV008B/T |
| | |
| | Am29LV040B |
| | |
| | Am29F200B/T |
| | Am29F400B/T |
| | Am29F800B/T |
| | Am29F800B/T |
| | |
| | Am29LV200B/T |
| | Am29LV400B/T |
| | Am29LV800B/T |
| | |
| | Am29DL16xB/T |
| | Am29DL32xB/T |
| Fujitsu | MBM29F800TA/TB |
| Hyundai | HY29F800B/T |
| Intel | 28F128J3xxx |
| | 28F320C3xxx |
| Macronix | MX29F004B/T |
| Sharp | LH28F320xxx |
| STMicroelectronics | M29F800AT/AB |
| Any | Any AMD compatible flash device |
| Any | Any INTEL compatible flash device |

Most other NOR flash devices are compatible with one of the supported devices. Thus the driver can be used with these devices or need a little modification, which can be easily done. Please get in touch with us, when you experience having problem modifying the flash access routines.

## 8.5.2. Configuration

To configure the NOR flash driver, please set the following macros according your hardware in the “FLASH_Config.h” (found in the ‘Config’ directory)
To use NOR flash driver, please define one of the following macros:

| Manufacturer | Device | Macro |
|---|---|---|
| AMD | Am29DL16xB | FLASH_29DL16xB |
| | Am29DL16xT | FLASH_29DL16xT |
| | Am29DL32xB | FLASH_29DL32xB |
| | Am29DL32xT | FLASH_29DL32xT |
| | Am29F002B | FLASH_29F002B |
| | Am29F002T | FLASH_29F002T |
| | Am29F004B | FLASH_29F004B |
| | Am29F004T | FLASH_29F004T |
| | Am29F008B | FLASH_29F008B |
| | Am29F008T | FLASH_29F008T |
| | Am29F200B | FLASH_29LV200B |
| | Am29F200T | FLASH_29LV200T |
| | Am29F400B | FLASH_29LV400B |
| | Am29F400T | FLASH_29LV400T |
| | Am29F800B | FLASH_29LV800B |
| | Am29F800B | FLASH_29F800B |
| | Am29F800T | FLASH_29LV800T |
| | Am29F800T | FLASH_29F800T |
| | Am29LV002B | FLASH_29LV002B |
| | Am29LV002T | FLASH_29LV002T |
| | Am29LV004B | FLASH_29LV004B |
| | Am29LV004T | FLASH_29LV004T |
| | Am29LV008B | FLASH_29LV008B |
| | Am29LV008T | FLASH_29LV008T |
| | Am29LV040B | FLASH_29LV040B |
| | Am29LV200B | FLASH_29LV200B |
| | Am29LV200T | FLASH_29LV200T |
| | Am29LV400B | FLASH_29LV400B |
| | Am29LV400T | FLASH_29LV400T |
| | Am29LV800B | FLASH_29LV800B |
| | Am29LV800T | FLASH_29LV800T |
| | Am29DL16xB | FLASH_29DL16xB |
| | Am29DL16xT | FLASH_29DL16xT |
| | Am29DL32xB | FLASH_29DL32xB |
| Fujitsu | MBM29F800TB | FLASH_29F800B |
| | MBM29F800TA | FLASH_29F800T |
| Hyundai | HY29F800B | FLASH_29F800B |
| | HY29F800T | FLASH_29F800T |
| Intel | 28F320J3xxx | FLASH_28F320J3 |
| | 28F640J3xxx | FLASH_28F640J3 |
| | 28F128J3xxx | FLASH_28F128J3 |
| | 28F256J3xxx | FLASH_28F256J3 |
| | 28F320C3xxx/B | FLASH_28F320B |
| | 28F320C3xxx/B | FLASH_28F320T |

| Manufacturer | Device | Macro |
|---|---|---|
| Macronix | MX29F004B | FLASH_29F004B |
| | MX29F004T | FLASH_29F004T |
| Sharp | LH28F320BJE | FLASH_28F320B |
| | LH28F320TJE | FLASH_28F320T |
| STMicroelectronics | M29F800AB | FLASH_29F800B |
| | M29F800AT | FLASH_29F800T |
| Any | Any AMD compatible | FLASH_29XX(Note 1) |
| Any | Any Intel compatible | FLASH_28XX (Note1) |

**(Note 1)**
For these “generic” defines, the flash sectoring needs to be defined. For details please refer to the chapter Flash sectoring

If you intend to use 2 flash chips (both 16bit wide), that combined define a 32bit flash module, please use the following macro to enable both the correct sectoring of the flash module and correct programming algorithm.

| Macro | Explanation |
|---|---|
| FLASH_32BIT | Set to 1 enables the “32bit mode” algorithm |

## 8.5.3. Flash sectoring

If a flash chip is selected, the flash driver knows the sectoring of the chip. Only if a “generic” define for the chip selection is used, the sectoring needs to be defined. This is done by defining the Sector addresses for all relevant sectors of the chip e.g. as follows (in case of a 256 Kbyte device with 4 sectors):

```
#define FLASH_SA0     (0x000000) /* 16K Boot block */
#define FLASH_SA1     (0x004000) /*  8K Parameter block */
#define FLASH_SA2     (0x006000) /*  8K Parameter block */
#define FLASH_SA3     (0x008000) /* 240 Main memory block */
#define FLASH_SA4     (0x040000) /*     End */
```

## 8.5.4. Additional options

The following table shows the additional configuration options available for the NOR flash driver:

| Macro | Explanation |
|---|---|
| FLASH_8BIT | Selects the 8 bit mode. Set to 1 if the driver should work in 8 bit mode. The default value depends on the selected flash. Note that not each flash supports both modes. |
| FLASH_16BIT | Selects the 16 bit mode. Set to 1 if the driver should work in 16 bit mode. The default value depends on the selected flash. Note that not each flash supports both modes. |
| FLASH_32BIT | Set to 1 enables the “32bit mode” algorithm. |
| FLASH_BASEADR | This defines the base address of the flash chip. It is important for setting up erase and write commands to the device. |
| FLASH_RELOCATECODE | (Note 1) |

**(Note 1)**
FLASH_RELOCATECODE defines whether the code for writing and erasing the flash is copied into RAM or not. If ***emLoad*** executes code in the same flash module as the sectors to be modified, the code needs to be relocated. This is required, because executing code in a flash module is not possible while it is in rewriting mode.

If relocation is enabled, ***emLoad*** uses a buffer for code relocation.
Before writing to the flash or erasing a sector, the flash module relocates (copies) the right routine to the buffer and starts execution at the buffer, typically by a subroutine call. This can be problematic for different reasons. Some of these reasons are:

- CPU has separate data and instruction caches (e.g. ARM9, MIPS).
- Compiler generates code with absolute jumps.
- Thumb/ARM mode switches on ARM CPUs (When code executes in thumb mode and "interwork" is used).
- Compiler generates code which does not fit in buffer area.
- Compiler generates code which references PC-relative data located far away from the routine.
- Interrupts occur during execution of the relocated code, while flash module is not ready (code required for ISR may not be accessible).
- If the relocation does not work in a different environment, please check the following:
  - Look at the assembly output and make sure that the relocated code does not contain absolute jumps. The buffer is big enough.
  - Check if there is no other problem which prevents the relocated routine from executing properly.
  - Check if the CPU has separate data and instruction caches, the instruction cache is disabled.

# 8.6. Interrupts

As described in a prior chapter the BTL sometimes should forward interrupts to the application program. This part of the BTL should be written in assembler and could not be included in the generic part of the software.

## 8.6.1. Different types of interrupt processing

There are basically 3 different types of interrupt processing:

1. Interrupt vectors with fixed base address
2. Interrupt vectors with variable base address
3. Vector less interrupt processing

The following describes how to manage these systems in ***emLoad***.

### 8.6.1.1. Fixed vectors

The interrupt handler is located in the BTL. This interrupt handler has to jump to the interrupt handler of the application program. This is basically an indirect jump, using the vector of the application program. If the CPU has such an indirect jump, things are easy; the interrupt processing in the BTL consists of a single jump indirect instruction for each interrupt vector.
If the CPU does not have such an instruction, things are more complicated. In this case, a series of instructions is required, basically doing the following:

- Reserve space on the stack for return address
- Save registers
- Read vector and write into reserved space on stack
- Restore registers
- Return

Example

The following sample shows how to forward the interrupts of the fixed vector table of a Renesas M16C CPU:

```
.
; *********************************************************************
; *
; *                     Function macro
; *
; *********************************************************************

ISR_HANDLER MACRO Isr, Adr
  PUBLIC Isr
  Isr:
  PUSH.W #0                       ; push 2 dummy bytes
  PUSH.B #0                       ; push 1 dummy byte
  PUSHM  A0, R0                   ; push used regs
  STC    SP, A0                   ; get SP
  LDE.W  (Adr - 04010H) + 0, R0   ; load new PCl and PCm to R0
  MOV.W  R0, 4[A0]                ; modify PCl and PCm on stack
  LDE.B  (Adr - 04010H) + 2, R0L  ; load new PCh to R0L
  MOV.B  R0L, 6[A0]               ; modify PCl and PCm on stack
  POPM   A0, R0                   ; pop used regs
  RTS                             ; use rts for jump
  ENDM

; *********************************************************************
; *
; *                     CODE
; *
; *********************************************************************

  RSEG CODE
  ISR_HANDLER __undefined_instruction_handler, 0fffdch
  ISR_HANDLER __overflow_handler             , 0fffe0h
```

```
ISR_HANDLER __break_instruction_handler    , 0fffe4h
ISR_HANDLER __address_match_handler        , 0fffe8h
ISR_HANDLER __single_step_handler          , 0fffech
ISR_HANDLER __watchdog_timer_handler       , 0ffff0h
ISR_HANDLER __DBC_handler                  , 0ffff4h
ISR_HANDLER __NMI_handler                  , 0ffff8h

END
```

#### 8.6.1.2. Variable vectors

Systems using this modern type of interrupt handling works well with the a BTL. Typically the startup code of the application program sets the base address of the vector table. The BTL needs no code to forward interrupts. This type of interrupt handling is used for example by Renesas M16C and M32C.

#### 8.6.1.3. Fixed address

CPUs of this type load the PC with a fixed value.

Code in RAM

No interrupt handling is required, since the application writes its own interrupt handler into the RAM. (Typical for ARMs with remapping).

Code in ROM

The interrupt handler is located in the BTL. This interrupt handler has to jump to the interrupt handler of the application program. This is basically a direct jump, to the interrupt handler of the application program. Typically the CPU has such a direct jump and things are easy:; the interrupt processing in the BTL consists of a single jump instruction for each interrupt vector.

Example

The following sample shows how to forward the interrupts of a NEC V850 system:

```
          org 10h
          jr  2010h

          org 20h
          jr  2020h

          org 30h
          jr  2030h

          org 40h
          jr  2040h
```

# 9. Available ports

The table below lists all currently available ports.

| Port | Supported CPU's |
|---|---|
| Generic | Should be adapted to a desired target |
| | |
| 78K4_IAR | NEC 78K4 |
| ARM_AT91SAM7 | (please take a look to the subchapter ARM AT91SAM7) |
| ARM_AT91SAM7L | (please take a look to the subchapter ARM AT91SAM7L) |
| ARM_AT91M40800_IAR | Arm AT91M40800 |
| ARM_AT91M55800_IAR | Arm AT91M55800 |
| CM3_LM3 | (please take a look to the subchapter Cortex-M3 Luminary) |
| CM3_LPC17xx | (please take a look to the subchapter Cortex-M3 LPC17xx) |
| M16C20_IAR | Renesas M30201F6 |
| M16C60_IAR | (please take a look to the subchapter M16C) |
| M16C60_NC30 | (please take a look to the subchapter M16C) |
| M16C60_TASKING | (please take a look to the subchapter M16C) |
| M16C65_IAR | (please take a look to the subchapter M16C65) |
| M16C80_IAR | Renesas M30800FCFP, M30800FCGP, M30803FGFP, M30803FGGP |
| M16C80_NC308 | Renesas M30800FCFP, M30800FCGP, M30803FGFP, M30803FGGP |
| M32C_IAR | (please take a look to the subchapter M32C) |
| M32C_NC308 | (please take a look to the subchapter M32C) |
| R32C_HEW4 | (please take a look to the subchapter R32C) |
| RX_HEW4 | (please take a look to the subchapter RX) |
| R8C_IAR | (please take a look to the subchapter R8C) |
| MIPS_VR4181A_GHSM2K | VR4181A |
| V850SA1_GHSM2K | NEC µPD70F3017A |
| V850SA2_IAR | NEC µPD70F3201, µPD70F3201Y |
| V850SF1_GHSM2K | NEC µPD70F3079Y |

The following subchapters describe some of the ***emLoad*** ports currently available in detail. If the CPU of the target system you are interested in is not listed here, please contact us. May we can provide you with some sample code even if the CPU is not listed here.

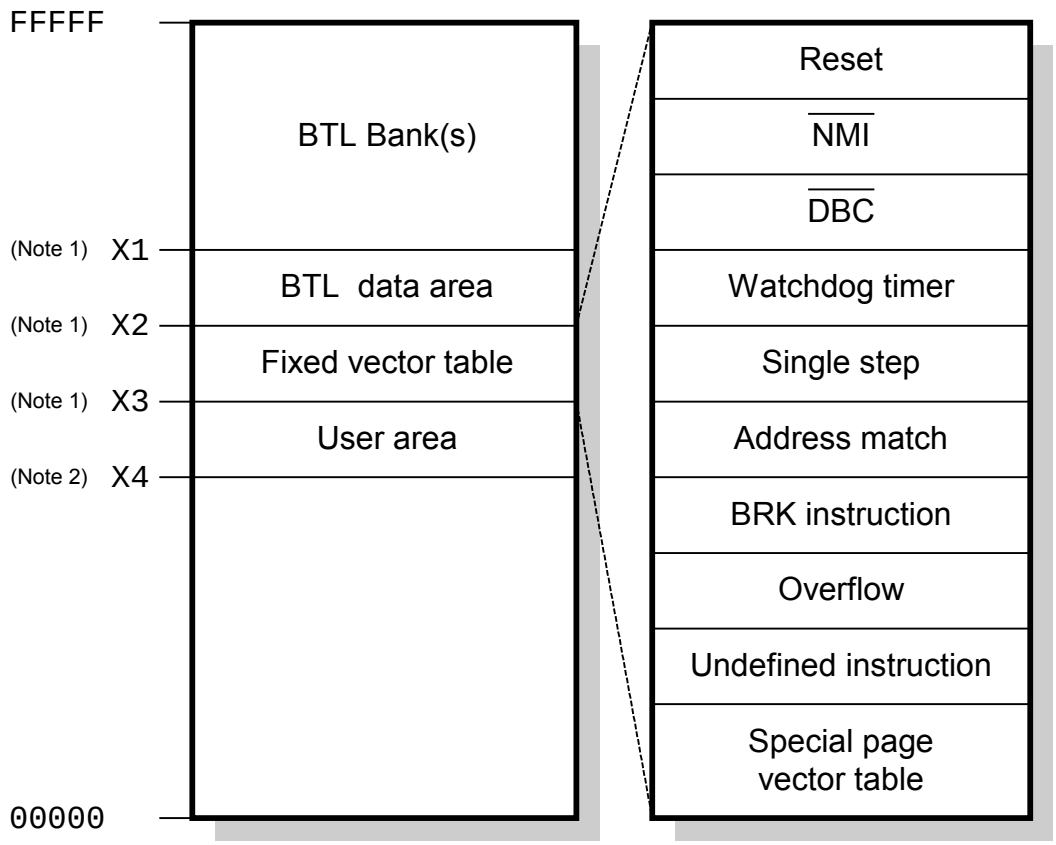
# 9.1. Renesas M16C

## 9.1.1. Supported CPU's:

| | | | |
|---|---|---|---|
| M30280F6HP, | M30280F6THP, | M30280F6VHP, | M30280F8HP, |
| M30280F8THP, | M30280F8VHP, | M30280FAHP, | M30280FATHP, |
| M30280FAVHP, | M30281F6HP, | M30281F6THP, | M30281F6VHP, |
| M30281F8HP, | M30281F8THP, | M30281F8VHP, | M30281FAHP, |
| M30281FATHP, | M30281FAVHP, | M30290F8HP | M30290F8THP |
| M30290F8VHP | M30290FAHP | M30290FATHP | M30290FAVHP |
| M30290FCHP | M30290FCTHP | M30290FCVHP | M30291F8HP |
| M30291F8THP | M30291F8VHP | M30291FAHP | M30291FATHP |
| M30291FAVHP | M30291FCHP | M30291FCTHP | M30291FCVHP |
| M30622F8PFP, | M30622F8PGP, | M30620FCPFP, | M30620FCPGP, |
| M30624FGPFP, | M30624FGPGP, | M30625FGPGP, | M30626FHPFP, |
| M30626FHPGP, | M30627FHPGP, | M30626FJPFP, | M30626FJPGP, |
| M30627FJPGP, | M30620FCAFP, | M30620FCAGP, | M30620FCMFP, |
| M30620FCMGP, | M30620FGFP, | M30620FGGP, | M30620FGLFP, |
| M30620FGLGP, | M30624FGAFP, | M30624FGAGP, | M30624FGFP, |
| M30624FGGP, | M30624FGMFP, | M30624FGMGP, | M30624FGLFP, |
| M30624FGLGP, | M30625FGGP, | M30625FGLGP, | M306NBFCTFP, |
| M306N0FGTFP, | M306NAFGTFP, | M3062GF8NFP, | M3062GF8NGP, |
| M30620FCNFP, | M30620FCNGP, | M30624FCNFP, | M30624FCNGP, |
| M30624FGNFP, | M30624FGNGP, | M30624FGNHP, | M306N4FCTFP, |
| M306N4FCVFP, | M306N5FCTFP, | M306N5FCVFP, | M306N4FGTFP, |
| M306N4FGVFP, | M30262F6GP, | M30262F8GP | |

## 9.1.2. Memory map

The diagram below shows the memory map of the M16C/62 memory.



Note 1: The addresses X1-X3 depend on the target CPU. The BTL needs 8K of ROM and is located at 0xFE000-0xFFFFF. If using a target with a 16K sector at the end of the flash, the address X1 is 0xFC000. If using a target with a 8K sector or 2 4K sectors at the end, the address X1 is 0xFE000.
The address X2 can be calculated as follows: X2 = X1 – 0x10.
The address X3 can be calculated as follows: X3 = X1 – 0x34.

Note 2: The beginning of the user area depends on the target. The address X4 is typically the first address of the flash area, for example 0xFC000 for a target with 256K of flash memory.

The BTL resides in the top bank of CPUs internal FLASH. Unfortunately this bank is 8 or 16kb in size (the BTL uses only approx. 5kb), but you loose the entire bank(s) for your application program. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x4010 or 0x2010 bytes depending on your target CPU. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations in the XCL-file as shown below.

### 9.1.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            20000000    /* [Hz]                              */
#define UARTSEL            1           /* select uart                       */
#define BAUDRATE           57600L      /* baudrate                          */

/* Common defines */
#define APPNAME            "BTLM16C " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* wait time after reset           */
                                       /* before app. is started [ms]     */
```

### 9.1.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. Sample:<br>10000000 for 10MHz<br>16000000 for 16 MHz |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication<br>(1200 ... 115200) |

### 9.1.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
#ifndef FLASH_CONFIG_H
#define FLASH_CONFIG_H

/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE huge

/* Define CPU type */
#define M30262F8GP

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"

#endif /* Avoid multiple inclusion */
```

### 9.1.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| M30262F8GP | Definition of the used CPU type. One of the CPU's listed under "Supported CPU's" has to be defined. Depending on the used CPU the BTL uses target depending default values for the user flash area. Furthermore it tells the BTL what kind of flash memory (HND or DINOR) is used. |

## 9.1.7. IAR-compiler

### 9.1.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.10A |
| Linker | 4.59I |
| Assembler | 3.10A |
| Workbench | 4.3° |

### 9.1.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

Link file: User.xcl

The M16C port for IAR contains a linker command file for the application program, USER.xcl. This linker command file should be used as a starting point. The file below shows a link file according to the memory map above for a target with 256K of flash memory. It may be changed if using an other target.

### 9.1.7.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M16C_IAR_V310A.dep` | Project file |
| `M16C_IAR_V310A.ewd` | Project file |
| `M16C_IAR_V310A.ewp` | Project file |
| `M16C_IAR_V310A.eww` | Workspace file |
| `PORT\BTL.xcl` | Linker command file for the BTL |
| `PORT\CPUM16C.h` | Special function register definitions for M16C |
| `PORT\FIXVECT.s34` | Pass on fixed vectors, may need to be modified |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |
| `PORT\USER.xcl` | Linker command file for the application program |

## 9.1.8. Renesas NC30-compiler

### 9.1.8.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 4.00r2 |
| Linker | 3.20.00 |
| Assembler | 4.00r2 |

### 9.1.8.2. Compiling and linking

The project should be rebuild using the batch file M.bat in the main folder. Before the project could be compiled the file PREP.bat should be adapted to the customers tool path by modifying the following line:

```
SET TOOLPATH=C:\TOOL\C\RENESAS\NC30WA400
```

The tool path should not include the \BIN path. After executing M.bat the EXE-subfolder should contain the executable file for the target hardware.

User startup files

The NC30 port for Renesas contains custom startup files for the application program, NCRT0_USER.A30 and SECT30_USER.INC. This files should be used as a starting point. The ROM start address and the fixed vector table address must may be modified.

### 9.1.8.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M.bat` | Batch file to build the target executable |
| `Prep.bat` | Batch file to set the tool path (should be modified) |
| `PORT\CPUM16C.h` | Special function register definitions for M16C |
| `PORT\FIXVECT.a30` | Pass on fixed vectors, may need to be modified |
| `PORT\NCRT0.a30` | Startup code for the BTL |
| `PORT\NCRT0_USER.a30` | Startup code for the application program |
| `PORT\SECT30.inc` | Section definitions for the BTL |
| `PORT\SECT30_USER.inc` | Section definitions for the application program |

## 9.1.9. TASKING-compiler

### 9.1.9.1. Used tools

| Tool | Version |
|---|---|
| Compiler/Assembler | 2.3r1 |
| Linker | 2.3r1 |
| Workbench | EDE |

### 9.1.9.2. Compiling and linking

The project should be opened by double clicking the workspace file `emLoad_M16C_TASKING.psp` or by opening it with the TASKING workbench. After modifying the files `BTLConf.h` and `FLASH_Config.h` the project can be rebuild. After rebuilding the project the executable code of the BTL is located in the project folder under `emload_m16c_tasking.hex`.

User project file

The NC30 port for TASKING contains a sample project/workspace `User_M16C_TASKING.psp` which can be used as a starting point for an application program. The project and the memory definition file of the user project must may be changed.

### 9.1.9.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `emLoad_M16C_TASKING_V23r1.pjt` | BTL project file for TASKING EDE |
| `emLoad_M16C_TASKING_V23r1.psp` | BTL workspace file for TASKING EDE |
| `User_M16C_TASKING.pjt` | Application project file for TASKING EDE |
| `User_M16C_TASKING.psp` | Application workspace file for TASKING EDE |
| `PORT\btlm16c.i` | Memory definition file for BTL |
| `PORT\CPUM16C.h` | Special function register definitions for M16C |
| `PORT\FIXVECT.asm` | Pass on fixed vectors, may need to be modified |
| `PORT\FLASH_Select.h` | Defines flash type & area by the defined CPU |
| `PORT\user.i` | Memory definition file for application |
| `PORT\user_cstart.src` | Startup code for application |
| `PORT\user_main.c` | Main routine for application |

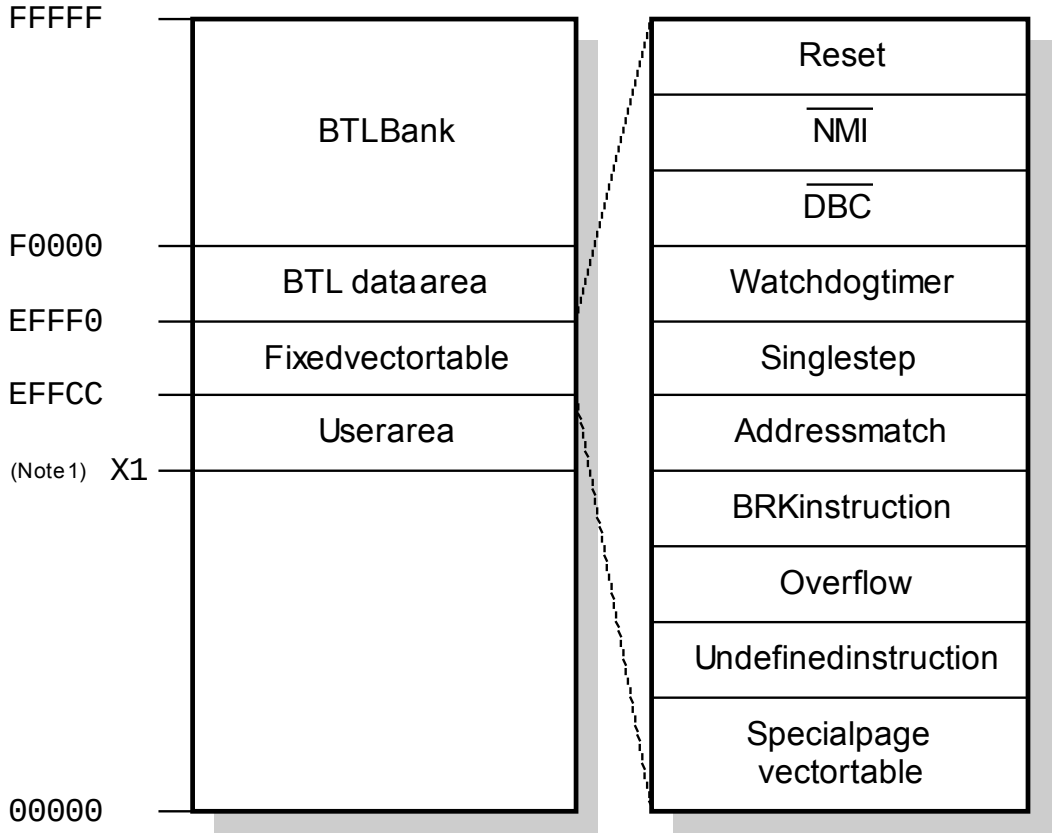
# 9.2. Renesas M16C65

## 9.2.1. Supported CPU's:

R5F36506NFA, R5F36506NFB, R5F36506DFA, R5F36506DFB,
R5F3651ENFC, R5F3650ENFA, R5F3650ENFB, R5F3651EDFC,
R5F3650EDFA, R5F3650EDFB, R5F3651KNFC, R5F3650KNFA,
R5F3650KNFB, R5F3651KDFC, R5F3650KDFA, R5F3650KDFB,
R5F3651MNFC, R5F3650MNFA, R5F3650MNFB, R5F3651MDFC,
R5F3650MDFA, R5F3650MDFB, R5F3651NNFC, R5F3650NNFA,
R5F3650NNFB, R5F3651NDFC, R5F3650NDFA, R5F3650NDFB,
R5F3651RNFC, R5F3650RNFA, R5F3650RNFB, R5F3651RDFC,
R5F3650RDFA, R5F3650RDFB

## 9.2.2. Memory map

The diagram below shows the memory map of the M16C/65 memory.




Note 1: The beginning of the user area depends on the target. The address X1 is typically the first address of the flash area, for example 0xFC000 for a target with 256K of flash memory.

The BTL resides in the top bank of CPUs internal FLASH. Unfortunately this bank is 64kb in size (the BTL uses only approx. 5kb), but you loose the entire bank for your application program. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x10010 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations in the XCL-file as shown above.

### 9.2.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            8000000     /* [Hz]                              */
#define UARTSEL            0           /* select uart                       */
#define BAUDRATE           38400L      /* baudrate                          */

/* Common defines */
#define APPNAME            "BTLM16C65 " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* wait time after reset           */
                                       /* before app. is started [ms]     */
```

### 9.2.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. Sample:<br>10000000 for 10MHz<br>16000000 for 16 MHz |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication<br>(1200 ... 115200) |

### 9.2.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
#ifndef FLASH_CONFIG_H
#define FLASH_CONFIG_H

/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE huge

/* Define CPU type */
#define R5F3650TD

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"

#endif /* Avoid multiple inclusion */
```

### 9.2.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| R5F3650TD | Definition of the used CPU type. A list of CPU defines can be found in the upper section of the file “FLASH_Select.h”. Depending on the used CPU the BTL uses target depending default values for the user flash area. Furthermore it tells the BTL what kind of flash memory is used. |

## 9.2.7. IAR-compiler

### 9.2.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.30D |
| Linker | 4.61G |
| Assembler | 3.30A |
| Workbench | 5.2.9.580.8668 |

### 9.2.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

Link file: USER.xcl

The M16C/65 port for IAR contains a linker command file for the application program, USER.xcl. This linker command file should be used as a starting point. The file shipped with the M16C/65 port has been configured to match a memory map for a target with 786K of flash memory. It may be changed if using an other target.

### 9.2.7.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M16C65_IAR_V330D.ewd` | Project file |
| `M16C65_IAR_V330D.ewp` | Project file |
| `M16C65_IAR_V330D.eww` | Workspace file |
| `PORT\BTL_ROM1.xcl` | Linker command file for the BTL |
| `PORT\CPUM16C65.h` | Special function register definitions for M16C65 |
| `PORT\FIXVECT.s34` | Pass on fixed vectors, may need to be modified |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |
| `PORT\USER.xcl` | Linker command file for the application program |

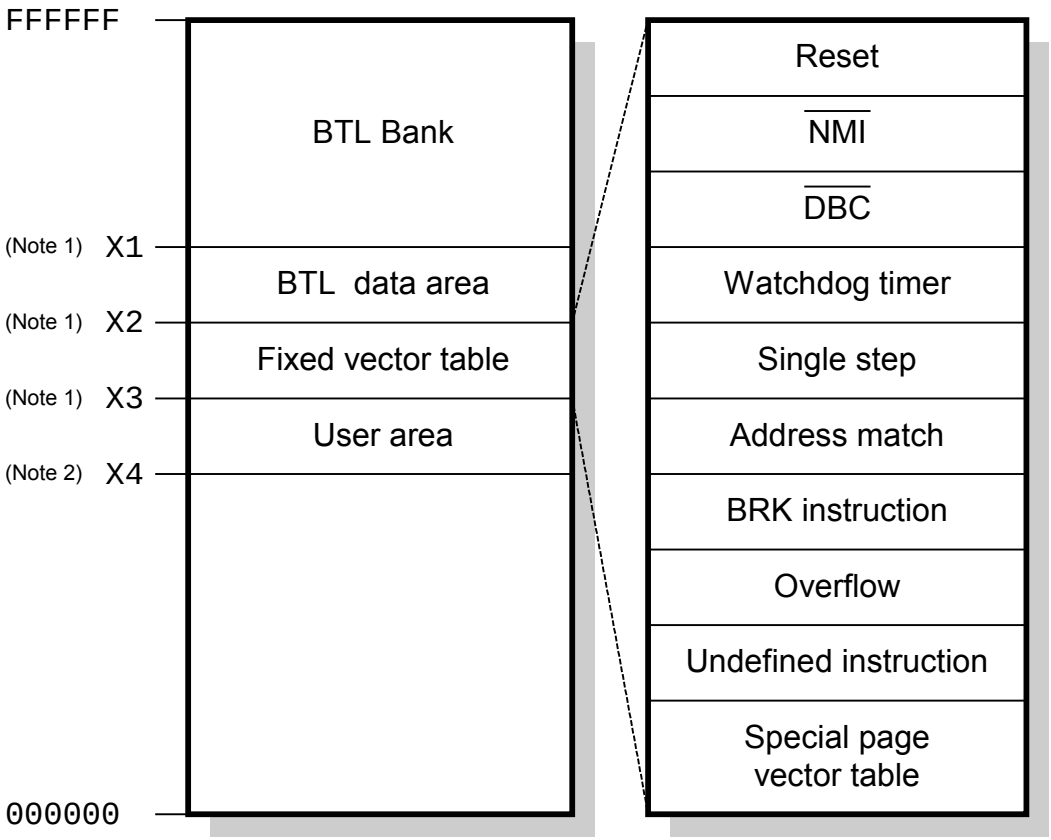
# 9.3. Renesas M32C

## 9.3.1. Supported CPU's:

| | | | |
|---|---|---|---|
| M30835FJGP, | M30833FJGP, | M30833FJFP, | M30853FHFP, |
| M30853FHGP, | M30853FHTGP, | M30853FJFP, | M30853FJGP, |
| M30853FJTGP, | M30853FWFP, | M30853FWGP, | M30853FWTGP, |
| M30855FHGP, | M30855FHTGP, | M30855FJGP, | M30855FJTGP, |
| M30855FWGP, | M30855FWTGP, | M3087BFLGP, | M30879FLGP, |
| M30879FLFP, | M3087BFLAGP, | M30879FLAGP, | M30879FLAFP |
| M3087BFKGP, | M30879FKGP, | M3087BFKAGP, | M30879FKAGP |
| M30878FJGP, | M30876FJGP, | M30878FJAGP, | M30876FJAGP |
| M30875FHGP, | M30873FHGP, | M30875FHAGP, | M30873FHAGP |

## 9.3.2. Memory map

The diagram below shows the memory map of the M32C memory.




Note 1: The addresses X1-X3 depend on the target CPU. The BTL needs 4K of ROM and is located at 0xFFF000-0xFFFFFF. If using a target with a 16K sector at the end of the flash, the address X1 is 0xFFC000. If using a target with a 4K sector at the end, the address X1 is 0xFFF000. For out of the box compatibility a model for a target of 16K is shipped.
The address X2 can be calculated as follows: X2 = X1 – 0x10.
The address X3 can be calculated as follows: X3 = X1 – 0x34.
Some devices have a separate block A located at 0xF000-0xFFFF. This block can also be used with ***emLoad*** .

Note 2: The beginning of the user area depends on the target. The address X4 is typically the first address of the flash area, for example 0xF80000 for a target with 512K of flash memory or 0xF00000 for a target with 1MB of flash memory.

The BTL resides in the top bank of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x4010 or 0x1010 bytes depending on your target CPU. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations for the linker.

### 9.3.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK           24000000        // [Hz]
#define UARTSEL           0               // select uart
#define BAUDRATE          115200          // baudrate

/* Common defines */
#define APPNAME           "BTLM32C " __DATE__ " " __TIME__
#define PASSWORD          ""
#define BTL_WAIT0_MS      500             // dwell time after reset
#define BTL_HUGE          __far
```

### 9.3.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1<br>…<br>4: UART 4 |
| BAUDRATE | Baudrate used for serial communication |

### 9.3.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE BTL_HUGE

/* Define CPU type */
#define M30879FLGP

/* Include FLASH_Select.h after defining CPU type */
#include "FLASH_Select.h"
```

### 9.3.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_M32C_USE_BLOCK_A | Enables support for block A. The default value is 0. Set to 1 to be able to program block A. |
| M30879FLGP | Definition of the used CPU type. One of the CPU's listed under "Supported CPU's" has to be defined. |

## 9.3.7. IAR-compiler

### 9.3.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.10a |
| Linker | 4.59j |
| Assembler | 3.10a |
| Workbench | 4.0 |

### 9.3.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLConf.H` and `FLASH_Config.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

Link file: User.xcl

The M32C port for IAR contains a linker command file for the application program, USER.xcl. This linker command file should be used as a starting point. It may be changed if using an other target.

### 9.3.7.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M32C_IAR_V310A.dep` | IAR project settings |
| `M32C_IAR_V310A.ewd` | IAR project settings |
| `M32C_IAR_V310A.ewp` | IAR project file |
| `M32C_IAR_V310A.eww` | IAR workspace file |
| `FLASH\FLASH_M32C.c` | Flash routines for the internal flash of M32C CPUs |
| `PORT\BTL.xcl` | Linker file for the BTL |
| `PORT\cpum32c.h` | Special function register definitions for M32C |
| `PORT\USER.xcl` | Linker file for application program |

## 9.3.8. Renesas NC308-compiler

### 9.3.8.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.41r1 |
| Linker | 4.04.02 |
| Assembler | 4.03r1 |
| Workbench | - |

### 9.3.8.2. Compiling and linking

The configuration files `BTLConf.H` and `FLASH_Config.H` needs to be configured to your hardware. `PREP.BAT` needs to be modified to set the right path for the compiler. `M.BAT` will call `PREP.BAT` to enhance the PATH-variable and sets the environment variables used by NC308. Please adapt the following line of `PREP.BAT`:

```
SET TOOLPATH=C:\Tool\C\HEW4\Tools\Renesas\nc308wa\v541r01
```

Now you can modify `BTLConf.H` and `FLASH_Config.H` and rebuild the BTL by executing `M.BAT`. The executable file `BTLM32C.MOT` will be stored in the `EXE`-folder.

### 9.3.8.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M.bat` | Batch file for rebuilding the BTL |
| `Prep.bat` | Batch file called by `M.bat` for setting the toolpath |
| `FLASH\FLASH_M32C.c` | Flash routines for the internal flash of M32C CPUs |
| `PORT\cpum32c.h` | Special function register definitions for M32C |
| `PORT\fixvect.a30` | Pass on fixed vectors |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |
| `PORT\ncrt0.a30` | Startup code BTL |
| `PORT\ncrt0_user.a30` | Startup code USER application, can be modified |
| `PORT\sect308.inc` | Sector definitions BTL |
| `PORT\sect308_user.inc` | Sector definitions USER application, can be modified |

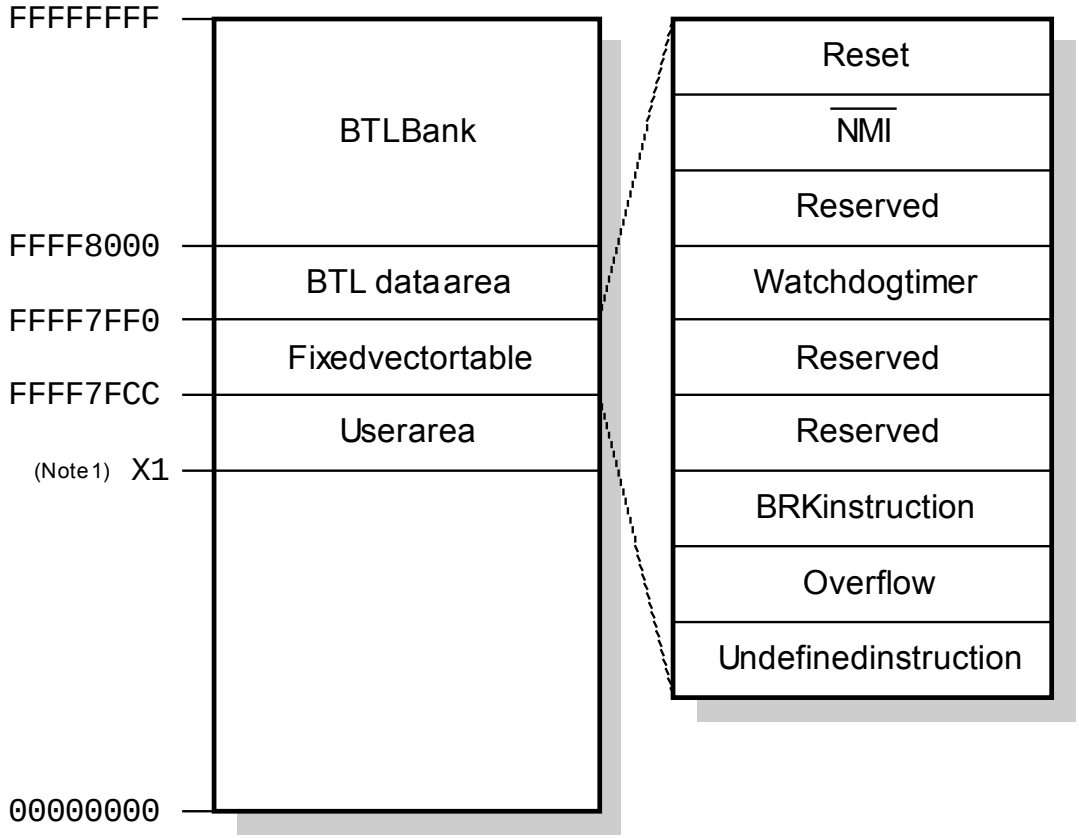
# 9.4. Renesas R32C

## 9.4.1. Supported CPU's:

| | | | |
|---|---|---|---|
| R5F6411E, | R5F64110, | R5F64114, | R5F6411F, |
| R5F64111, | R5F641x5, | R5F64112, | R5F641x6, |
| R5F641x7, | R5F64514, | R5F6451M, | R5F641x8, |
| R5F64515, | R5F6451N, | R5F641x9 | |

## 9.4.2. Memory map

The diagram below shows the memory map of the R32C memory.




Note 1: The beginning of the user area depends on the target. The address X1 is typically the first address of the flash area, for example 0xFFFE000 for a target with 128K of flash memory.

The BTL resides in the top bank of CPUs internal FLASH. Unfortunately this bank is 32kb in size (the BTL uses only approx. 5kb), but you loose the entire bank for your application program. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x8010 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations as shown above.

### 9.4.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            64000000    /* [Hz]                              */
#define UARTSEL            0           /* select uart                       */
#define BAUDRATE           115200L     /* baudrate                          */

/* Data needs to be programmed in blocks of 8 bytes */
#define BTL_WRITE_BLOCK_SIZE  8

/* Common defines */
#define APPNAME            "BTLR32C " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* wait time after reset          */
                                       /* before app. is started [ms]     */
```

### 9.4.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. Sample:<br>10000000 for 10MHz<br>16000000 for 16 MHz |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication<br>(1200 ... 115200) |

### 9.4.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
#ifndef FLASH_CONFIG_H
#define FLASH_CONFIG_H

/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE huge

/* Define CPU type */
#define R5F6411E

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"

#endif /* Avoid multiple inclusion */
```

### 9.4.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| R5F6411E | Definition of the used CPU type. A list of CPU defines can be found in the upper section of the file “FLASH_Select.h”. Depending on the used CPU the BTL uses target depending default values for the user flash area. Furthermore it tells the BTL what kind of flash memory is used. |

## 9.4.7. IAR-compiler

### 9.4.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 1.30E |
| Linker | 4.61J |
| Assembler | 1.30A |
| Workbench | 5.4.0.832 |

### 9.4.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

### 9.4.7.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `R32C_IAR.ewd` | Project file |
| `R32C_IAR.ewp` | Project file |
| `R32C_IAR_V*.eww` | Workspace file |
| `PORT\BTL.xcl` | Linker command file for the BTL |
| `PORT\CPU_R32C.h` | Special function register definitions for R32C |
| `PORT\FIXVECT.asm` | Pass on fixed vectors, may need to be modified |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |

## 9.4.8. RENESAS-compiler

### 9.4.8.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 1.01.00 |
| Linker | 1.00.02 |
| Assembler | 1.01.00 |
| Workbench | 4.07.00 |

### 9.4.8.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

### 9.4.8.3. Additional program modules

| File | Explanation |
|---|---|
| `BTL_RSK_R32C111.*` | Workspace files |
| `RSK_R32C111\*` | Project files |
| `R32C_IAR_V*.eww` | Workspace file |
| `RSK_R32C111\PORT\sect100.inc` | Linker sections file for the BTL and pass on fixed vectors, may need to be modified |
| `RSK_R32C111\PORT\CPU_R32C.h` | Special function register definitions for R32C |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |

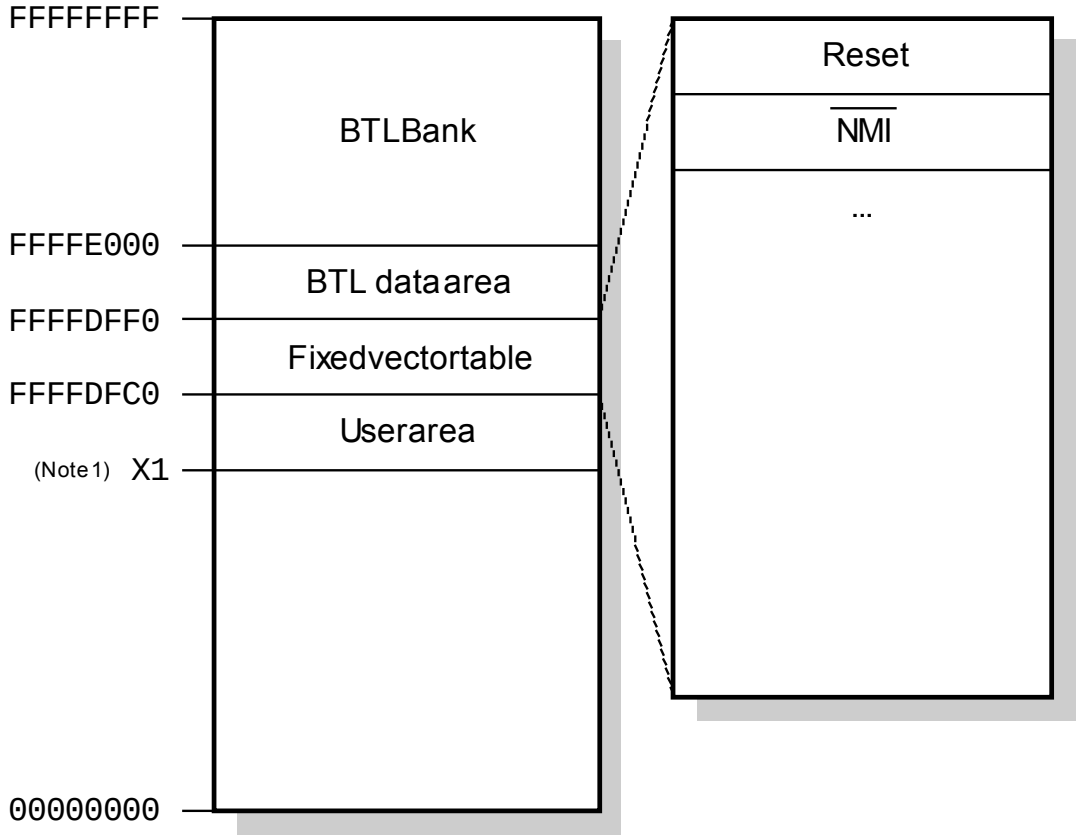
# 9.5. Renesas RX

## 9.5.1. Supported CPU's:

| | | | |
|---|---|---|---|
| R5F562x6, | R5F562x7, | R5F562x8, | R5F563x6, |
| R5F563x7, | R5F563x8, | R5F563xA, | R5F563xB, |
| R5F563xD, | R5F563xE, | R5F563xM, | R5F563xN, |
| R5F563xP, | R5F52103, | R5F52104, | R5F52105, |
| R5F52106, | R5F52107, | R5F52108, | R5F5210A, |
| R5F5210B | | | |

## 9.5.2. Memory map

The diagram below shows the memory map of the RX memory.




Note 1: The beginning of the user area depends on the target. The address X1 is typically the first address of the flash area, for example 0xFFFC0000 for a target with 256K of flash memory.

The BTL resides in the two top banks of CPUs internal FLASH that consists of two 4kByte blocks. Since the RESET vector is located in this banks, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x2010 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations as shown above.

### 9.5.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            96000000    /* [Hz]                              */
#define UARTSEL            2           /* select uart                       */
#define BAUDRATE           115200L     /* baudrate                          */

/* Data needs to be programmed in blocks of 256 bytes */
#define BTL_WRITE_BLOCK_SIZE  256uL

/* Common defines */
#define APPNAME            "BTLRX " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* wait time after reset          */
                                       /* before app. is started [ms]     */
```

### 9.5.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. Sample: 96000000 for 96MHz |
| UARTSEL | Selects the UART used for communication. Should be: 2: UART 2 |
| BAUDRATE | Baudrate used for serial communication (9600 ... 115200) |

### 9.5.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
#ifndef FLASH_CONFIG_H
#define FLASH_CONFIG_H

/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE BTL_HUGE

/* Define CPU type */
#define R5F562x8

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"

#endif /* Avoid multiple inclusion */
```

### 9.5.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| R5F562x8 | Definition of the used CPU type. A list of CPU defines can be found in the upper section of the file “FLASH_Select.h”. Depending on the used CPU the BTL uses target depending default values for the user flash area. Furthermore it tells the BTL what kind of flash memory is used. |

## 9.5.7. RENESAS-compiler

### 9.5.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 1.02.01 |
| Linker | 1.02.00 |
| Assembler | 1.02.00 |
| Workbench | 4.09.00.007 |

### 9.5.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder of the configuration.

### 9.5.7.3. Additional program modules

| File | Explanation |
|---|---|
| `BTL_YRDK_RX62N.*` | Workspace files |
| `BTL_YRDK_RX62N\*` | Project files |
| `BTL_YRDK_RX62N.hws` | Workspace file |
| `BTL_YRDK_RX62N \PORT\CPU_RX.h` | Special function register definitions for RX |
| `BTL_YRDK_RX62N \PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |

## 9.5.8. IAR-compiler

### 9.5.8.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 2.40.1.50509 |
| Linker | 2.40.1.50509 |
| Assembler | 2.40.1.50509 |
| Workbench | 6.4.0.2310 |

### 9.5.8.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder of the configuration.

### 9.5.8.3. Additional program modules

| File | Explanation |
|---|---|
| `BTL_YRDK_RX62N.*` | Project files |
| `BTL_YRDK_RX62N_V*.eww` | Workspace file |
| `PORT\CPU_RX.h` | Special function register definitions for RX |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |

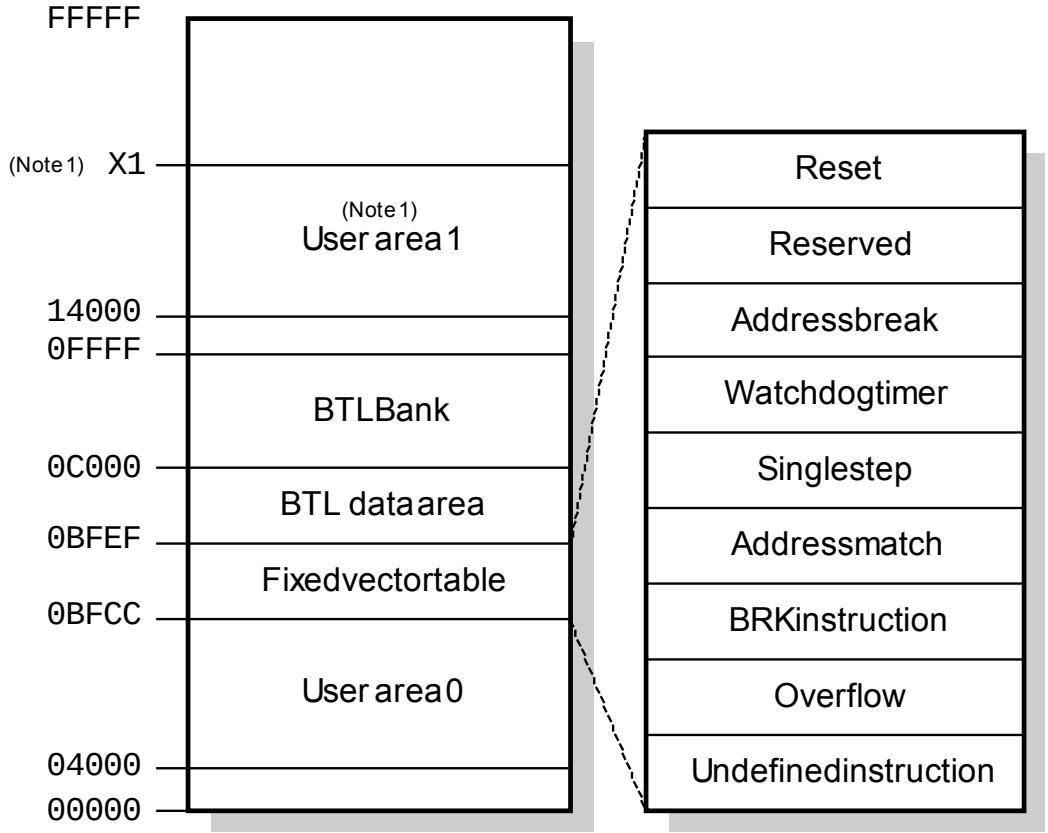
# 9.6. Renesas R8C

## 9.6.1. Supported CPU's:

R5F212x7, R5F212x8, R5F212xA, R5F212xC

## 9.6.2. Memory map

The diagram below shows the memory map of the R8C memory.




Note 1: This user area does only exist for targets with 96kByte or more flash. The end of this user area depends on the target. The address X1 is typically the last address of the flash blocks after the flash block containing the BTL, for example 0x23FFF for a target with 128kByte of flash memory.

The BTL resides in the bank of CPUs internal FLASH that contains the reset vector located at 0x0FFFF. Unfortunately this bank is 32kByte in size (the BTL uses only approx. 5kb), but you loose the entire bank for your application program. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The RESET vector of the application program is moved down in memory by 0x4010 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory locations in the XCL-file as shown above.

### 9.6.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            20000000    /* [Hz]                              */
#define UARTSEL            2           /* select uart                       */
#define BAUDRATE           115200L     /* baudrate                          */

/* Common defines */
#define APPNAME            "BTLR8C " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* wait time after reset             */
                                       /* before app. is started [ms]       */
#define BTL_HUGE           __far
#define BTL_RW_U32NO       0
```

### 9.6.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. Sample:<br>10000000 for 10MHz<br>16000000 for 16 MHz |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1<br>2: UART 2 |
| BAUDRATE | Baudrate used for serial communication<br>(1200 ... 115200) |

### 9.6.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
#ifndef FLASH_CONFIG_H
#define FLASH_CONFIG_H

/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32  U32
#define FLASH_U16  U16
#define FLASH_U8   U8
#define FLASH_HUGE huge

/* Define CPU type */
#define R5F212xA

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"

#endif /* Avoid multiple inclusion */
```

### 9.6.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| R5F212xA | Definition of the used CPU type. A list of CPU defines can be found in the upper section of the file “FLASH_Select.h”. Depending on the used CPU the BTL uses target depending default values for the user flash area. Furthermore it tells the BTL what kind of flash memory is used. |

## 9.6.7. IAR-compiler

### 9.6.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.30D |
| Linker | 4.61G |
| Assembler | 3.30A |
| Workbench | 5.2.9.580.8668 |

### 9.6.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in the EXE-subfolder.

### 9.6.7.3. Additional program modules

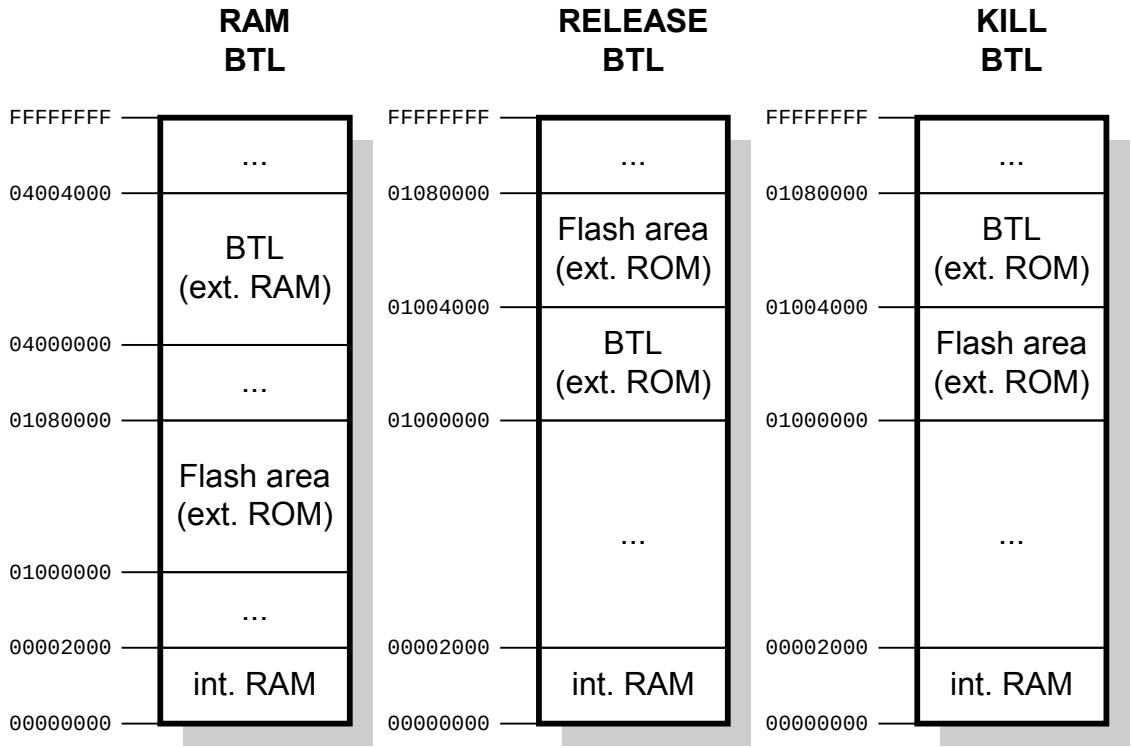
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `R8C_IAR.ewd` | Project file |
| `R8C_IAR.ewp` | Project file |
| `R8C_IAR_V*.eww` | Workspace file |
| `PORT\BTL.xcl` | Linker command file for the BTL |
| `PORT\CPU_R8C.h` | Special function register definitions for R32C |
| `PORT\FixVect.s34` | Pass on fixed vectors, may need to be modified |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area and flash type |

# 9.7. ARM AT91M40800

## 9.7.1. Supported CPU's:

AT91M40800

## 9.7.2. Memory map




The project contains 3 targets:

RAM_BTL

This target is used to run the BTL in RAM using a wiggler. Use the RAM_BTL to program the RELEASE_BTL into the external flash.

RELEASE_BTL

The release target.

KILL_BTL

This target can be used to program a new release version of the BTL into the external flash without using a wiggler.

## 9.7.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
#ifndef BTLCONF_H  // Avoid multiple inclusion
#define BTLCONF_H

/* common defines for BTL */
#define PASSWORD              ""
#define BTL_WAIT0_MS          500        // wait time after reset
                                         // before app. is started [ms]

/* cpu and target board specific defines */
#define UPCLOCK               7372800
#define UARTSEL               1          // select uart
#define BAUDRATE              115200     // baudrate

/* Type of external flash */
#define FLASH_29LV400B        1

#if   defined(RAM_BTL)
  #define APPNAME             "RAM BTL ARM AT91"
  #define FLASH_BASEADR       0x01000000
  #define FLASH_USER_START    0x01000000 // Start of application program
  #define FLASH_USER_LEN      0x00080000 // Length of user area
#elif defined(RELEASE_BTL)
  #define APPNAME             "BTL ARM AT91"
  #define FLASH_BASEADR       0x01000000
  #define FLASH_USER_START    0x01004000 // Start of application program
  #define FLASH_USER_LEN      0x0007C000 // Length of user area
#elif defined(KILL_BTL)
  #define APPNAME             "KILLER BTL ARM AT91"
  #define FLASH_BASEADR       0x01000000
  #define FLASH_USER_START    0x01000000 // Start of application program
  #define FLASH_USER_LEN      0x00004000 // Length of user area
#else
  #error No BTL selected!
#endif

#endif // defined BTLCONF_H
```

## 9.7.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication (1200 ... 115200) |
| FLASH_29LV400B | Activate the FLASH routines |

## 9.7.5. IAR-compiler

### 9.7.5.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.30a |
| Linker | 4.55d |
| Assembler | 3.30a |
| Workbench | 3.4a |

### 9.7.5.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.7.5.3. Additional program modules

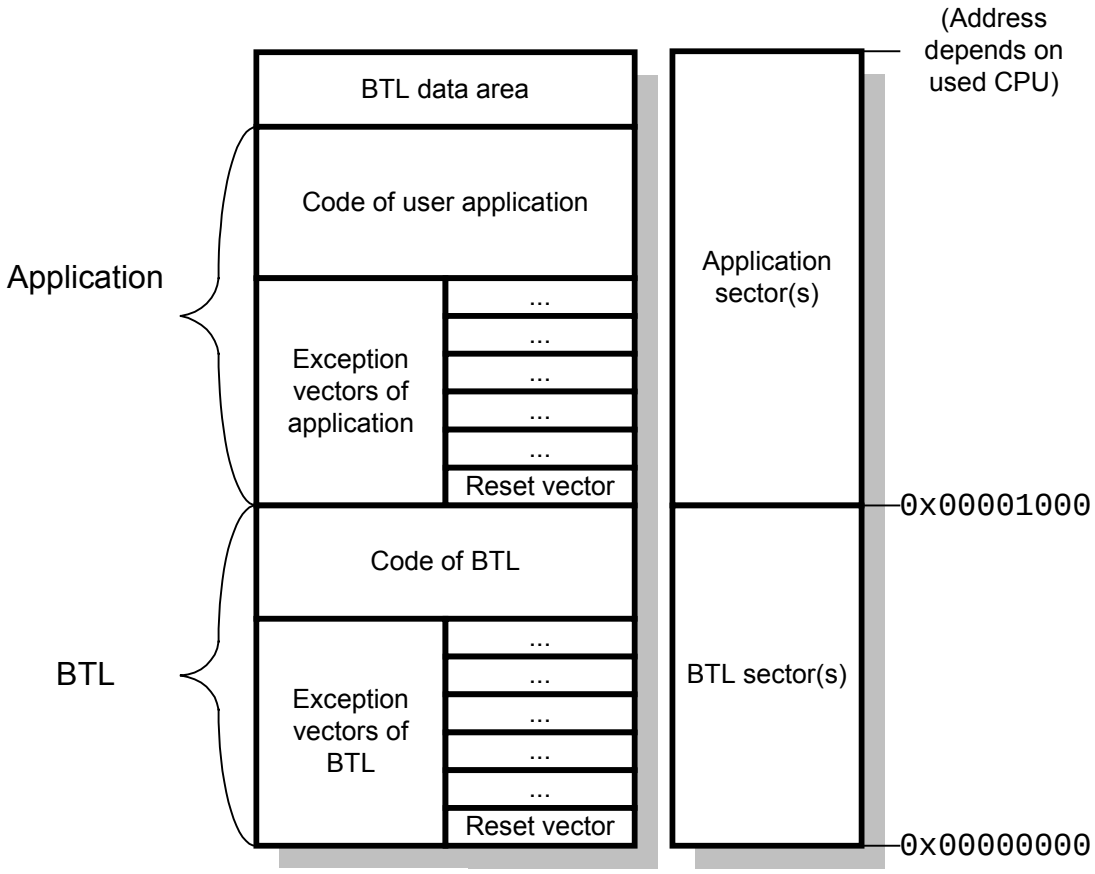
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `ARM_AT91_IAR_V330A.pew` | Project file for IAR IDE |
| `PORT\at91_cstartup.s79` | Modified startup containing the remapping |
| `PORT\FixVect.asm` | Pass on fixed vectors |
| `PORT\JTAG_AT91.mac` | Macro for RAM_BTL using IAR IDE with JTAG |
| `PORT\JTAG_AT91.xcl` | Linker file for RAM_BTL |
| `PORT\KILL_AT91.xcl` | Linker file for KILL_BTL |
| `PORT\LowLevelInit.c` | Contains __low_level_init |
| `PORT\RELEASE_AT91.xcl` | Linker file for RELEASE_BTL |

# 9.8. ARM AT91SAM7

## 9.8.1. Supported CPU's:

| | | | |
|---|---|---|---|
| AT91SAM7A3 | AT91SAM7SE32 | AT91SAM7SE256 | AT91SAM7SE512 |
| AT91SAM7S128 | AT91SAM7S128A | AT91SAM7S256 | AT91SAM7S256A |
| AT91SAM7S32 | AT91SAM7S321 | AT91SAM7S64 | AT91SAM7X128 |
| AT91SAM7X256 | | | |

## 9.8.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.8.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            47923200L   /* [Hz]                              */
#define UARTSEL            0           /* select uart                       */
#define BAUDRATE           115200L     /* baudrate                          */

/* Data needs to be programmed in blocks of 256 bytes */
#define BTL_WRITE_BLOCK_SIZE 256

/* Flash user area definition */
#define FLASH_USER_START  0x101000     /* Start adress of flash user area */
#define FLASH_USER_LEN    0x03F000     /* Length of flash user area       */

/* common defines */
#define APPNAME            "BTL AT91SAM7 " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset           */
                                       /* Before app. is started [ms]     */
```

### 9.8.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 0, 1: UART 1, 2: UART 2 (only SAM7A3) |
| BAUDRATE | Baudrate used for serial communication |

### 9.8.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* FLASH selection */
#define FLASH_AT91SAM7 1
```

### 9.8.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_AT91SAM7 | Use the AT91SAM7 flash module. |

## 9.8.7. IAR-compiler

### 9.8.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 4.31a |
| Linker | 4.59w |
| Assembler | 4.31a |
| Workbench | 4.6B |

### 9.8.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.8.7.3. Additional program modules

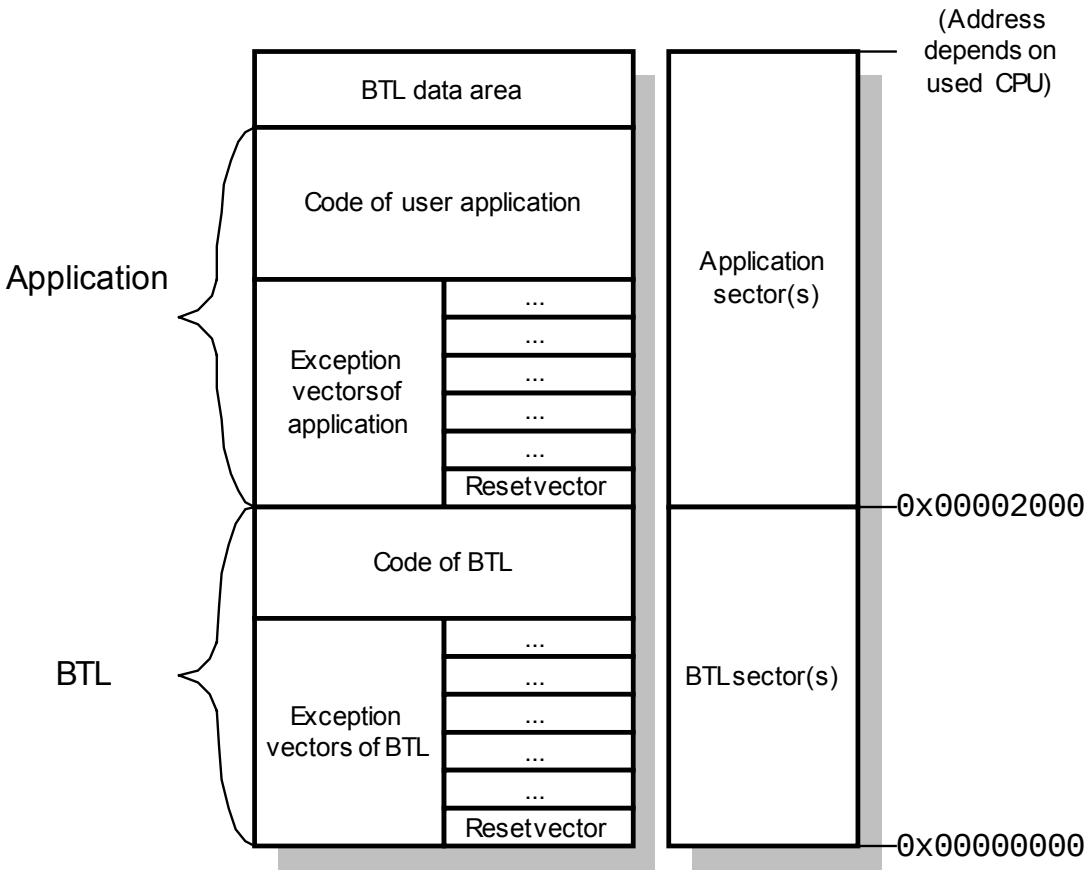
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_AT91SAM7_V431A.dep` | Project for IAR IDE |
| `BTL_AT91SAM7_V431A.ewd` | “” |
| `BTL_AT91SAM7_V431A.ewp` | “” |
| `BTL_AT91SAM7_V431A.eww` | “” |
| `AT91SAM7S256_FLASH.xcl` | Linker file for release configuration |
| `AT91SAM7S256_RAM.xcl` | Linker file for debug configuration |
| `AT91SAM7_Cstartup.s79` | Startup code |
| `SAM7_FLASH.mac` | Macro file used to debug the FLASH build |
| `SAM7_RAM.mac` | Macro file used to debug the RAM build |

# 9.9. ARM AT91SAM7L

## 9.9.1. Supported CPU's:

AT91SAM7L64 AT91SAM7L128

## 9.9.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x2000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.9.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            29982720L   /* [Hz]                              */
#define UARTSEL            2           /* select uart                       */
#define BAUDRATE           230400L     /* baudrate                          */

/* Data needs to be programmed in blocks of 256 bytes */
#define BTL_WRITE_BLOCK_SIZE 256

/* Flash user area definition */
#define FLASH_USER_START  0x102000     /* Start adress of flash user area */
#define FLASH_USER_LEN    0x01E000     /* Length of flash user area       */

/* common defines */
#define APPNAME            "BTL AT91SAM7L " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset           */
                                       /* Before app. is started [ms]     */
```

### 9.9.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 0, 1: UART 1, 2: DBGU |
| BAUDRATE | Baudrate used for serial communication |

### 9.9.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* FLASH selection */
#define FLASH_AT91SAM7L 1
```

### 9.9.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_AT91SAM7L | Use the AT91SAM7L flash module. |

## 9.9.7. IAR-compiler

### 9.9.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.20.3.51064 |
| Linker | 5.20.3.51064 |
| Assembler | 5.20.3.51064 |
| Workbench | 5.3.0.622 |

### 9.9.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.9.7.3. Additional program modules

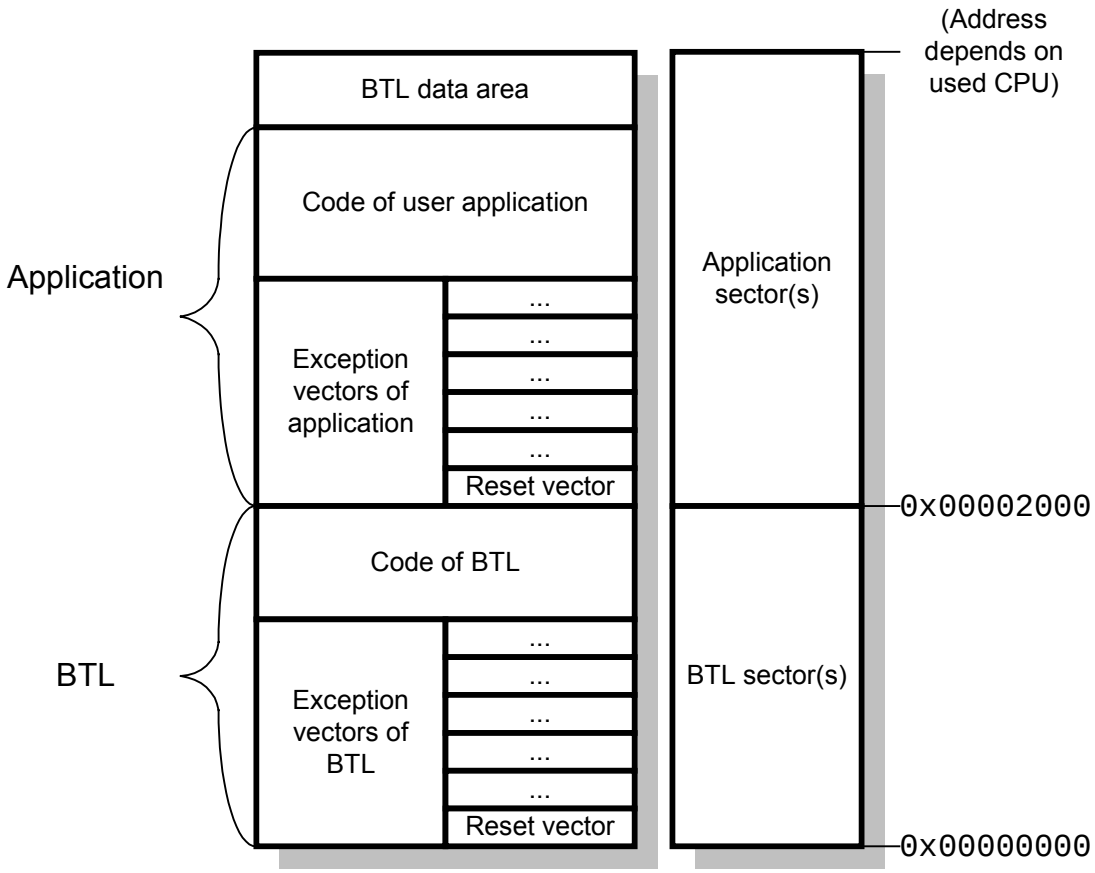
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_AT91SAM7L_V5.ewd` | Project for IAR IDE |
| `BTL_AT91SAM7L_V5.ewp` | “” |
| `BTL_AT91SAM7L_V520.eww` | “” |
| `AT91SAM7L128_FLASH.icf` | Linker file for flash configuration |
| `AT91SAM7_Startup.s` | Startup code |
| `SAM7L_FLASH.mac` | Macro file used to debug the FLASH build |

# 9.10. ARM LH754XX

## 9.10.1. Supported CPU's:

LH75400 LH75401 LH75410 LH75401

## 9.10.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x2000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.10.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* CPU specific definitions */
#define BTL_RW_U32NO           1
#define BTL_WRITE_16BIT_ALIGNED 1

/* Application, harware specific definitions */
#define UPCLOCK     14175000
#define UART        0
#define BAUDRATE    115200L

/* FLASH specific definitions */
#if defined(TARGET_JTAG_RELEASE)
  #define FLASH_USER_START   0x40002000
#elif defined (TARGET_RELEASE)
  #define FLASH_USER_START   0x00002000
#endif
#define FLASH_USER_LEN     0x003FE000

/* common defines */
#define APPNAME            "LH754XX " __DATE__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Dwell time after reset          */
                                       /* Before app. is started [ms]     */
```

### 9.10.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 0, 1: UART 1 |
| BAUDRATE | Baudrate used for serial communication |

### 9.10.5. FLASH specific configuration file

The flash area is configured by the `FLASH_Config.h` file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* FLASH specific definitions */
#if defined(TARGET_JTAG_RELEASE)
  #define FLASH_BASEADR      0x40000000
#elif defined (TARGET_RELEASE)
  #define FLASH_BASEADR      0x00000000
#endif

/* External area: Flash driver selection */
#define FLASH_29XX         1

/* External area: Sector definition */
#define FLASH_SA0  0x002000
...
```

### 9.10.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_29XX | Definition of the used kind of Flash. |

## 9.10.7. IAR-compiler

### 9.10.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 4.30a |
| Linker | 4.59n |
| Assembler | 4.30a |
| Workbench | 4.5 |

### 9.10.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.10.7.3. Additional program modules

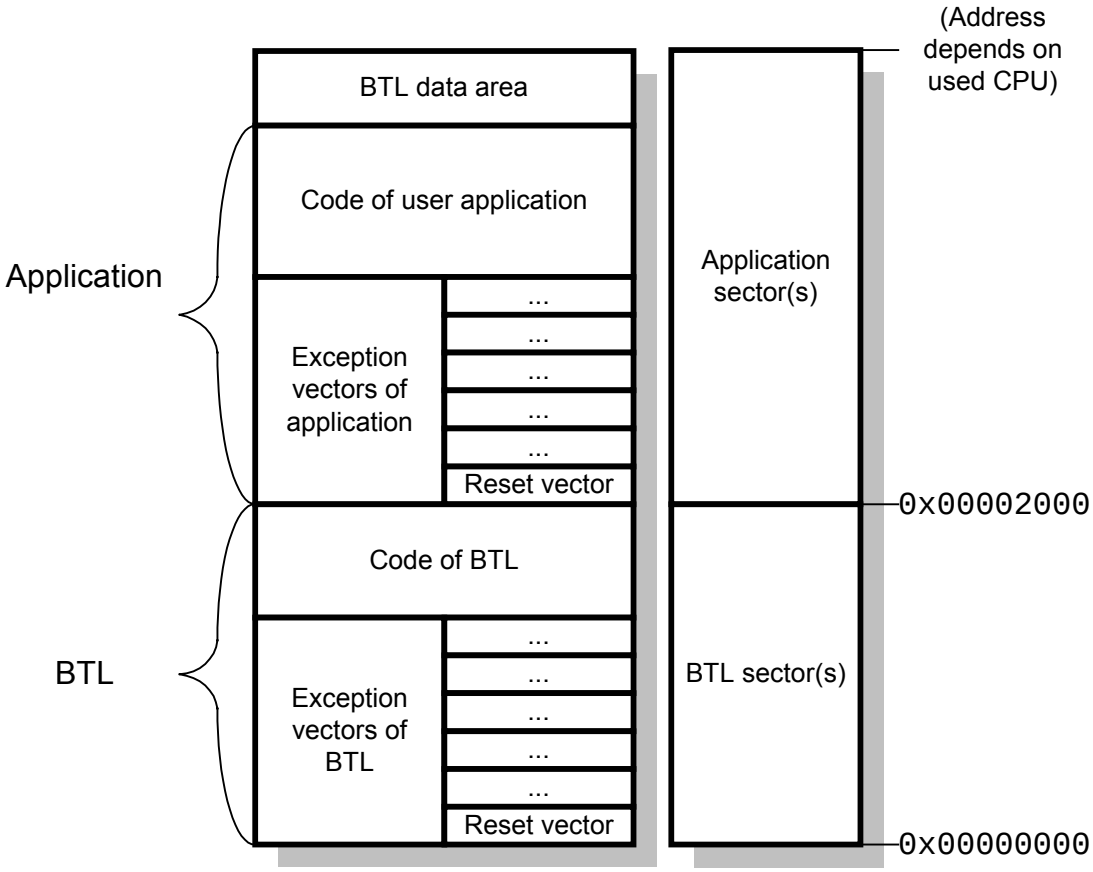
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_LH754XX_V420.dep` | Project for IAR IDE |
| `BTL_LH754XX_V420.ewd` | “” |
| `BTL_LH754XX_V420.ewp` | “” |
| `BTL_LH754XX_V420.eww` | “” |
| `BTL_Release_LH754XX.xcl` | Linker file for release configuration |
| `JTAG_Release_LH754XX.xcl` | Linker file for JTAG configuration |
| `USER_Release_LH754XX.xcl` | Linker file for application |

# 9.11. ARM LPC2XXX

## 9.11.1. Supported CPU's:

| | | | |
|---|---|---|---|
| LPC2104 | LPC2105 | LPC2106 | LPC2114 |
| LPC2119 | LPC2124 | LPC2129 | LPC2131 |
| LPC2132 | LPC2134 | LPC2136 | LPC2138 |
| LPC2194 | LPC2212 | LPC2214 | LPC2292 |
| LPC2294 | | | |

## 9.11.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x2000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.11.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU specific definitions */
#define BTL_USE_PARA(Para)   Para = Para  /* Avoid warnings */
#define BTL_WRITE_BLOCK_SIZE 512

/* Application, harware specific definitions */
#define UPCLOCK     14745600L
#define UART        1
#define BAUDRATE    115200L

#define APPNAME     "BTL LPC2138 " __DATE__

/* common defines */
#define BTL_WAIT0_MS       500         /* Dwell time after reset          */
                                       /* Before app. is started [ms]     */
```

### 9.11.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication |

### 9.11.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8
#define FLASH_USE_PARA(Para) Para = Para /* Avoid warnings */

/* Define CPU type */
#define FLASH_LPC_2138 1

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"
```

### 9.11.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_LPC_2138 | Definition of the used CPU type. One of the CPU's listed under "Supported CPU's" has to be defined. If for example a LPC2138 should be used, the following line needs to be included:<br><br>#define FLASH_LPC_2138  1 |

## 9.11.7. Keil-compiler

### 9.11.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 2.23a |
| Linker | 2.23a |
| Assembler | 2.22 |
| Workbench | µVision3 V3.12f |

### 9.11.7.2. Compiling and linking

BTL

The project file `ARM_LPC21XX_BTL.Uv2` should be opened by double click from the Windows Explorer. Now you can modify `BTLConf.H` and `FLASH_Config.H` and rebuild the BTL. The actual BTL will be in the subfolder `Output\BTL\Obj`.

Sample application

The emLoad shipment contains a sample project similar to the 'Blinky' sample shipped with the Keil compiler. The project file `ARM_LPC21XX_APP.Uv2` should be opened by double click from the Windows Explorer. After rebuilding it the output file `ARM_LPC2XXX_APP.hex` will be in the subfolder `Output\APP\Obj`.

### 9.11.7.3. Additional program modules

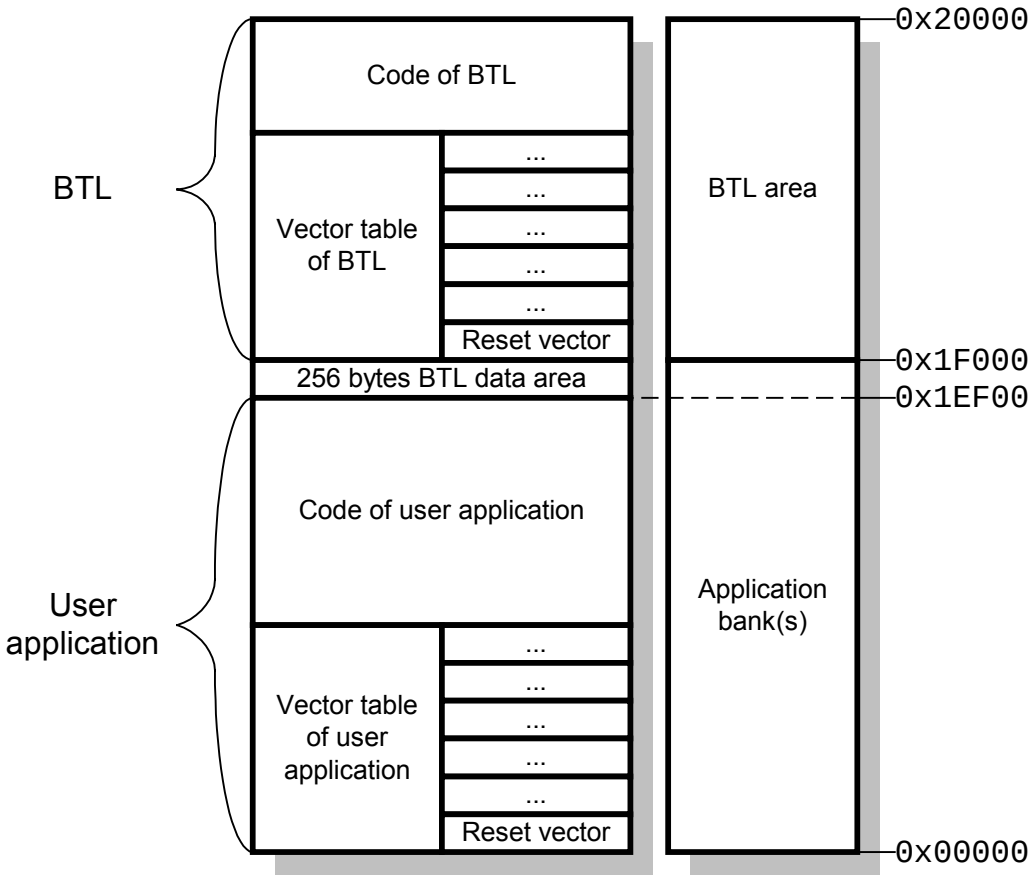
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `ARM_LPC21XX_APP.Opt` | Project settings of sample application |
| `ARM_LPC21XX_APP.Uv2` | Project settings of sample application |
| `ARM_LPC21XX_BTL.Opt` | Project settings of BTL |
| `ARM_LPC21XX_BTL.Uv2` | Project settings of BTL |
| `FLASH\FLASH_LPC2xxx.c` | Flash routines for the internal flash of LPC2XXX CPUs |
| `PORT\FLASH_Select.h` | Defines defaults for flash user area depending of defined CPU |
| `PORT\MainAPP.c` | Sample application |
| `PORT\StartupAPP.S` | Startup code sample application |
| `PORT\StartupBTL.S` | Startup code BTL |

# 9.12. ATMEL ATmega128

## 9.12.1. Supported CPU's:

ATmega128

## 9.12.2. Memory map




## 9.12.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
#ifndef BTLCONF_H  // Avoid multiple inclusion
#define BTLCONF_H

/* Common defines for BTL */
#define PASSWORD              ""
#define APPNAME               "emLoad ATMEGA"
#define BTL_WAIT0_MS          500        // Wait time after reset
#define BTL_RW_U32NO          0
#define BTL_WRITE_BLOCK_SIZE 256
#define BTL_HUGE              __hugeflash
                                         // Before app. is started [ms]
/* CPU and target board specific defines */
#define UPCLOCK               7372800    // Oszillator frequency
#define UARTSEL               0          // Select uart
#define BAUDRATE              115200     // Baudrate

/* Type of external flash */
#define FLASH_ATMEGA          1
#define FLASH_USER_START      0x00000    // Start of application program
#define FLASH_USER_LEN        0x1F000    // Length of user area
#define FLASH_USER_RESBYTES   256
```

### 9.12.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | Selects the UART used for communication.<br>Should be:<br>0: UART 0<br>1: UART 1 |
| BAUDRATE | Baudrate used for serial communication (1200 ... 115200) |
| FLASH_ATMEGA | Activate the ATmega128 flash routines |

## 9.12.5. IAR-compiler

### 9.12.5.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 3.10b |
| Linker | 4.56f |
| Assembler | 3.10b |
| Workbench | 3.0b |

### 9.12.5.2. Compiling and linking

The BTL can be rebuild using the batch file M.bat in the main folder or by using the project file.
The project file should be opened by double click from the Windows Explorer or by opening it with the IAR workbench. It contains 2 targets: the debug and the release target. To build the target executable the release target should be selected. Now you can modify `BTLCONF.H` and rebuild the BTL.
Before the project could be compiled by the batch file M.bat it should be adapted to the customers tool path by modifying the following line:

```
SET TOOLPATH=C:\Tool\C\IAR\AVR_V310B
```

The actual BTL will be in the subfolder `Output\Release\Exe`.

Fuses

Please note that the BTL does not work if the ATmega128 runs in ATmega103 compatibility mode. Further the fuses should enable the use of the reset vector of the BTL. The following table shows the settings required for the BTL:

| Fuse byte | Bit | Value |
|---|---|---|
| Extended Fuse Byte | M103C | 1 |
| Fuse High Byte | BOOTSZ1 | 0 |
| | BOOTSZ0 | 1 |
| | BOOTRST | 0 |

### 9.12.5.3. Additional program modules

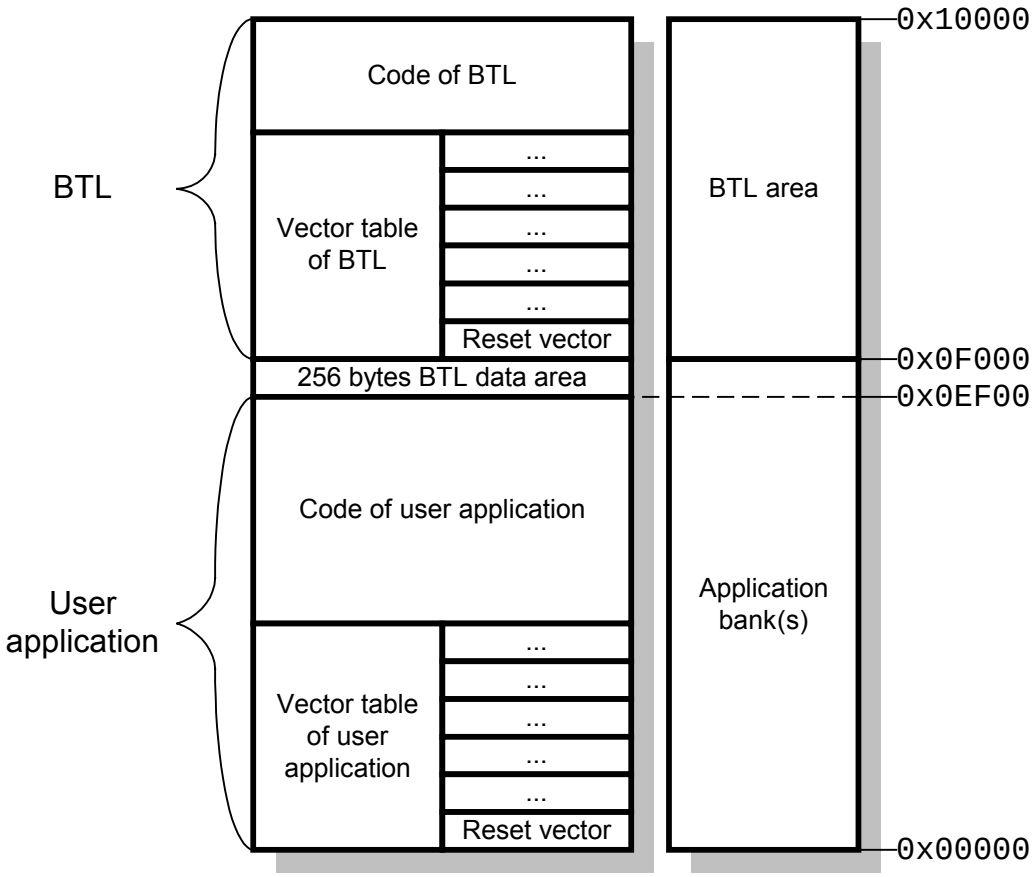
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `M.bat` | Batch file to build the target executable |
| `AVR_IAR.eww` | Workspace file for IAR IDE |
| `AVR_IAR.ewp` | Project file for IAR IDE |
| `FLASH\FLASH_ATMEGA.c` | Flash routines for ATmega128 |
| `FLASH\FLASH_ATMEGA_HELP.s90` | Assembler helper routines |
| `PORT\BTL.xcl` | Linker file for BTL |

# 9.13. ATMEL ATmega644

## 9.13.1. Supported CPU's:

ATmega644

## 9.13.2. Memory map




## 9.13.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* Common defines for BTL */
#define PASSWORD              ""
#define APPNAME               "emLoad ATMEGA"
#define BTL_RW_U32NO          0
#define BTL_WRITE_BLOCK_SIZE 256
#define BTL_HUGE              __flash
#define FEEDWATCHDOG()        asm("WDR")
#define BTL_WAIT0_MS          500        // Wait time after reset
                                         // Before app. is started [ms]

/* CPU and target board specific defines */
#define UPCLOCK               6140000    // Oszillator frequency
#define BAUDRATE              38400      // Baudrate

/* Type of external flash */
#define FLASH_ATMEGA          1
#define FLASH_USER_START      0x00000    // Start of application program
#define FLASH_USER_LEN        0x0F000    // Length of user area
#define FLASH_USER_RESBYTES   256
```

## 9.13.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| BAUDRATE | Baudrate used for serial communication (1200 ... 115200) |
| FLASH_ATMEGA | Activate the Atmega644 flash routines |

## 9.13.5. IAR-compiler

### 9.13.5.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 4.20A |
| Linker | 4.59Z |
| Assembler | 4.20A |
| Workbench | 4.7 |

### 9.13.5.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer or by opening it with the IAR workbench. It contains 2 targets: the debug and the release target. To build the target executable the release target should be selected. Now you can modify `BTLCONF.H` and rebuild the BTL.

The actual BTL will be in the subfolder `Output\Release\Exe`.

Fuses

Please note that the BTL does not work if the Atmega644 runs in ATmega103 compatibility mode. Further the fuses should enable the use of the reset vector of the BTL. The following table shows the settings required for the BTL:

| Fuse byte | Bit | Value |
|---|---|---|
| Fuse High Byte | BOOTSZ1 | 0 |
| | BOOTSZ0 | 1 |
| | BOOTRST | 0 |

### 9.13.5.3. Additional program modules

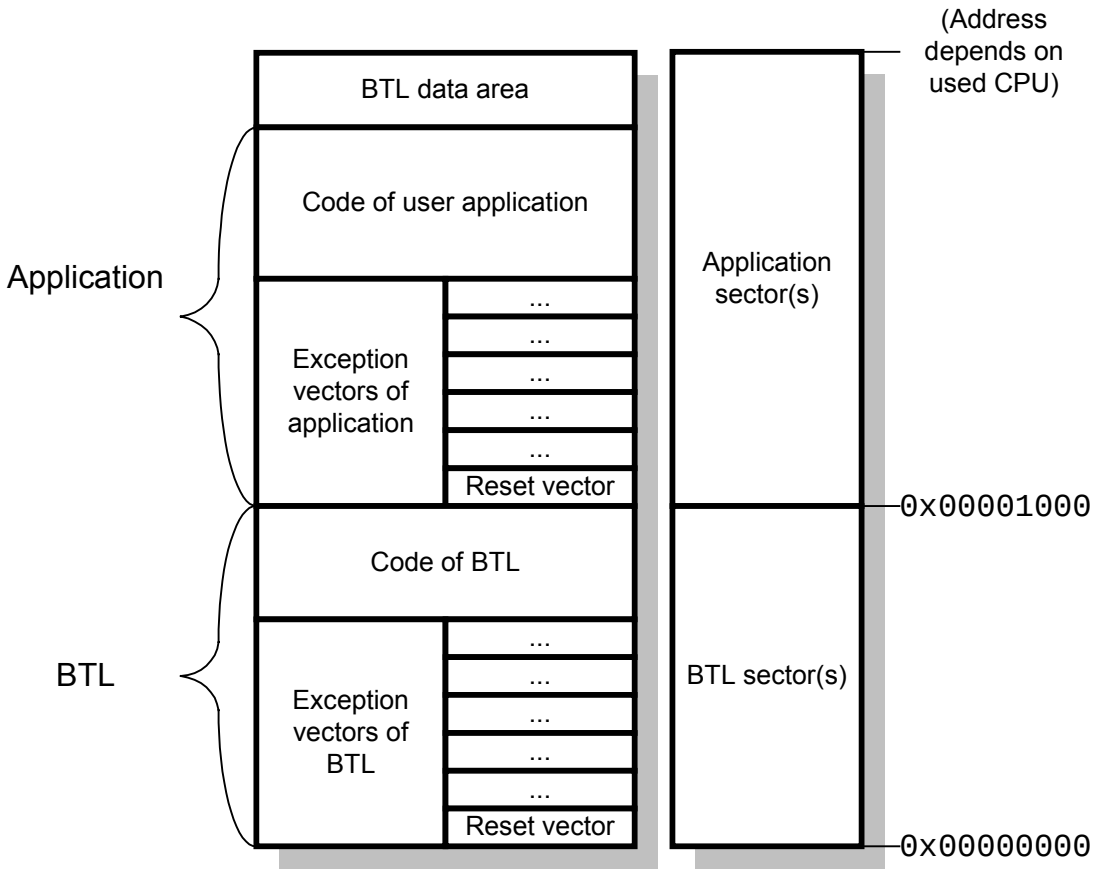
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `AVR_IAR_V420A.eww` | Workspace file for IAR IDE |
| `AVR_IAR_V420A.ewp` | Project file for IAR IDE |
| `FLASH\FLASH_ATMEGA.c` | Flash routines for Atmega644 |
| `FLASH\FLASH_ATMEGA_HELP.s90` | Assembler helper routines |
| `PORT\ATmega644.h` | SFR definitions for Atmega644 |
| `PORT\BTL.xcl` | Linker file for BTL |

# 9.14. Cortex-M3 Luminary

## 9.14.1. Supported CPU's:

All Cortex-M3 based Luminary CPUs.

## 9.14.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.14.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            8000000L    /* [Hz]                              */
#define UARTSEL            0           /* select uart                       */
#define BAUDRATE           230400L     /* baudrate                          */

/* Data needs to be programmed in blocks of 1024 bytes */
define BTL_WRITE_BLOCK_SIZE 1024

/* common defines */
#define APPNAME            "BTL Luminary LM3 " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset             */
                                       /* Before app. is started [ms]       */
```

### 9.14.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 0 |
| BAUDRATE | Baudrate used for serial communication |

### 9.14.5. FLASH specific configuration file

The flash area is configured by the `FLASH_Config.h` file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define LM3

/* FLASH selection */
#define LM3_16K  1

/* Include the file FLASH_Select.h after the CPU type definition */
#include "FLASH_Select.h"
```

### 9.14.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| LM3 | Use the Luminary LM3 flash module. |
| LM3_16K | Select the flash sector information for Luminary CPUs with 16K flash.<br>Available flash configurations are:<br>LM3_16K<br>LM3_21K<br>LM3_64K<br>LM3_96K<br>LM3_128K<br>LM3_256K |

## 9.14.7. IAR-compiler

### 9.14.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.41.2.51793 |
| Linker | 5.41.2.51793 |
| Assembler | 5.41.2.51793 |
| Workbench | 5.6.2.1418 |

### 9.14.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.14.7.3. Additional program modules

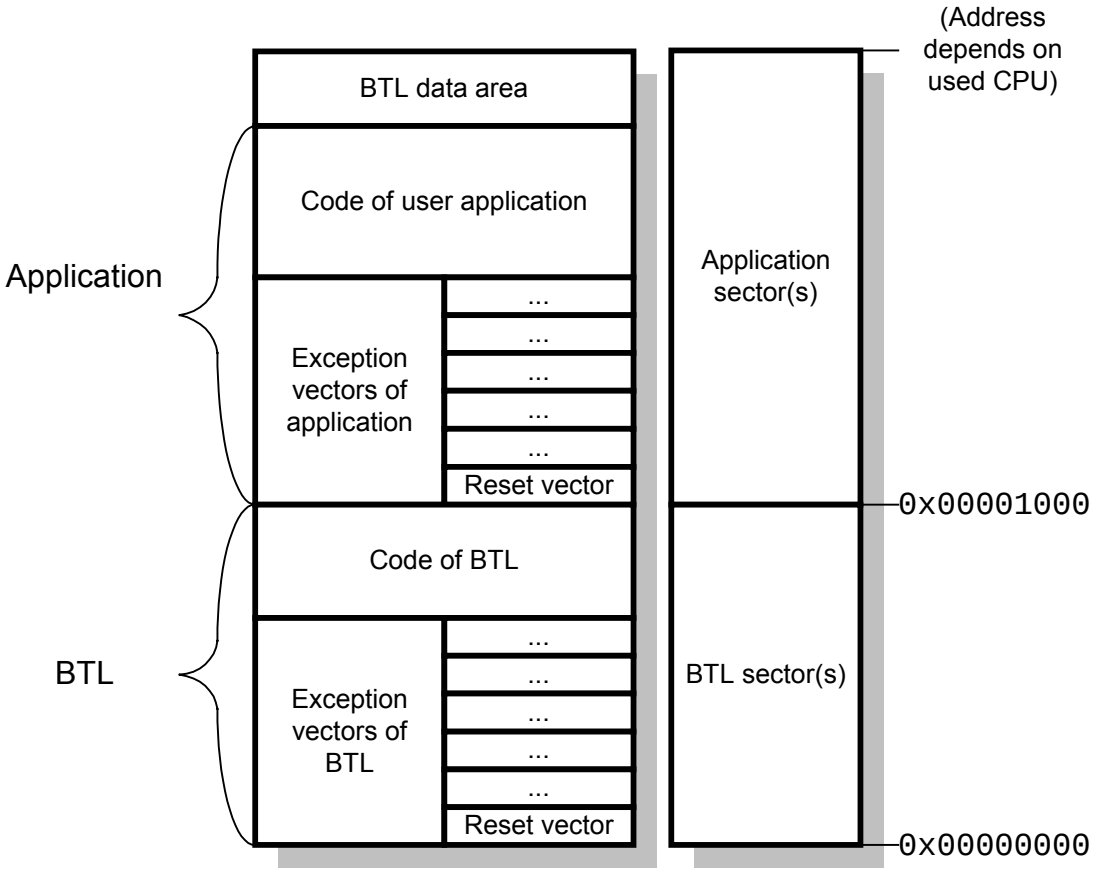
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_LM3.ewd` | Project for IAR IDE |
| `BTL_LM3.ewp` | “” |
| `BTL_LM3_V541B.eww` | “” |
| `FLASH.icf` | Linker file for debug and release configuration |
| `CPU_ASM_IAR_V5.s` | Firmware start code |

## 9.15. Cortex-M3 LPC17xx

### 9.15.1. Supported CPU's:

| | | | |
|---|---|---|---|
| LPC1751 | LPC1752 | LPC1754 | LPC1756 |
| LPC1758 | LPC1764 | LPC1765 | LPC1766 |
| LPC1768 | | | |

### 9.15.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

## 9.15.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            72000000L   /* [Hz]                              */
#define UARTSEL            0           /* select uart                       */
#define BAUDRATE           230400L     /* baudrate                          */

/* Data needs to be programmed in blocks of 256 bytes */
#define BTL_WRITE_BLOCK_SIZE 256

/* common defines */
#define APPNAME            "BTL LPC17xx " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset             */
                                       /* Before app. is started [ms]       */
```

## 9.15.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 0 |
| BAUDRATE | Baudrate used for serial communication |

## 9.15.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define LPC17X1

/* FLASH selection */
#define FLASH_LPC17XX 1
```

## 9.15.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_LPC17XX | Use the LPC17xx flash module. |
| LPC17X1 | Select the flash sector information for LPC17x1 CPUs.<br>Available flash configurations are:<br>LPC17X1<br>LPC17X2<br>LPC17X4<br>LPC17X5<br>LPC17X6<br>LPC17X8 |

## 9.15.7. IAR-compiler

### 9.15.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.41.2.51793 |
| Linker | 5.41.2.51793 |
| Assembler | 5.41.2.51793 |
| Workbench | 5.6.2.1418 |

### 9.15.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.15.7.3. Additional program modules

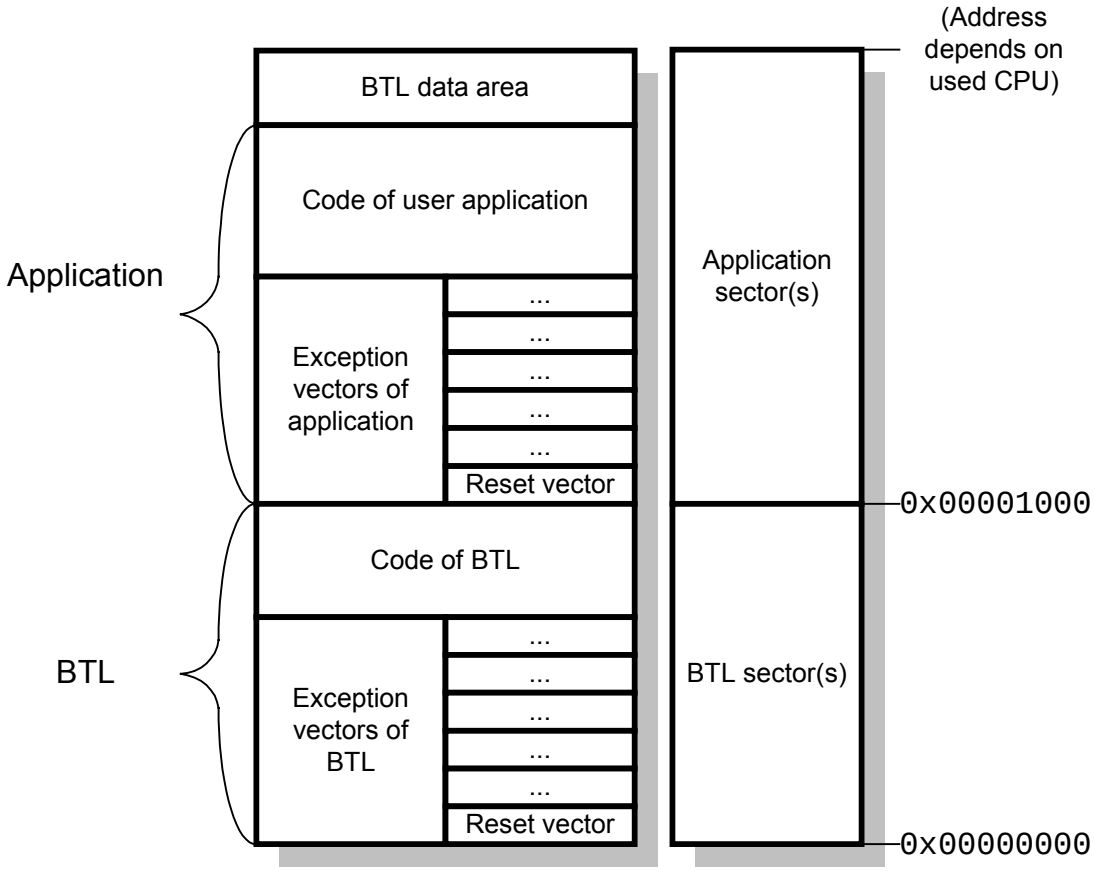
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_LPC17xx.ewd` | Project for IAR IDE |
| `BTL_LPC17xx.ewp` | “” |
| `BTL_LPC17xx_V541B.eww` | “” |
| `FLASH.icf` | Linker file for debug and release configuration |
| `CPU_ASM_IAR_V5.s` | Firmware start code |
| `FLASH.mac` | Macro file used to debug the FLASH build |

## 9.16. Cortex-M3 STM32F10x

### 9.16.1. Supported CPU's:

| | | | |
|---|---|---|---|
| STM32F10xx4 | STM32F10xx6 | STM32F10xx8 | STM32F10xxB |
| STM32F10xxC | STM32F10xxD | STM32F10xxE | |

### 9.16.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.16.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK             72000000L  // [Hz]
#define UARTSEL             0          // Select UART, STM3210E-EVAL:
                                       // 0: UART1, 1: UART2
#define UART2_NOT_REMAPPED  1          // On STM3210E-EVAL UART2 is not
                                       // remapped
#define BAUDRATE            230400L    // Baudrate

//
// Data needs to be programmed in a multiple of 16 bytes.
// Greater block size means less overhead by unlock/lock operations.
//
#define BTL_WRITE_BLOCK_SIZE  256

//
// Common defines
//
#define APPNAME            "BTL STM32F10x " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset           */
                                       /* Before app. is started [ms]     */
```

### 9.16.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART 1 on STM3210E-EVAL eval board.<br>1: UART 2 on STM3210E-EVAL eval board |
| UART2_NOT_REMAPPED | Port pins do not need to be remapped to alternate function for UART. |
| BAUDRATE | Baudrate used for serial communication |

### 9.16.5. FLASH specific configuration file

The flash area is configured by the `FLASH_Config.h` file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define STM32F10XXE

/* FLASH selection */
#define FLASH_STM32F10X  1
```

### 9.16.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_STM32F10X | Use the STM32F10x flash module. |

| | |
|---|---|
| STM32F10XXE | Select the flash sector information for STM32F10xxE CPUs.<br>Available flash configurations are:<br>STM32F10XX4<br>STM32F10XX6<br>STM32F10XX8<br>STM32F10XXB<br>STM32F10XXC<br>STM32F10XXD<br>STM32F10XXE |

## 9.16.7. IAR-compiler

### 9.16.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.50.5.51995 |
| Linker | 5.50.5.51995 |
| Assembler | 5.50.5.51995 |
| Workbench | 5.8.0.1623 |

### 9.16.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.16.7.3. Additional program modules

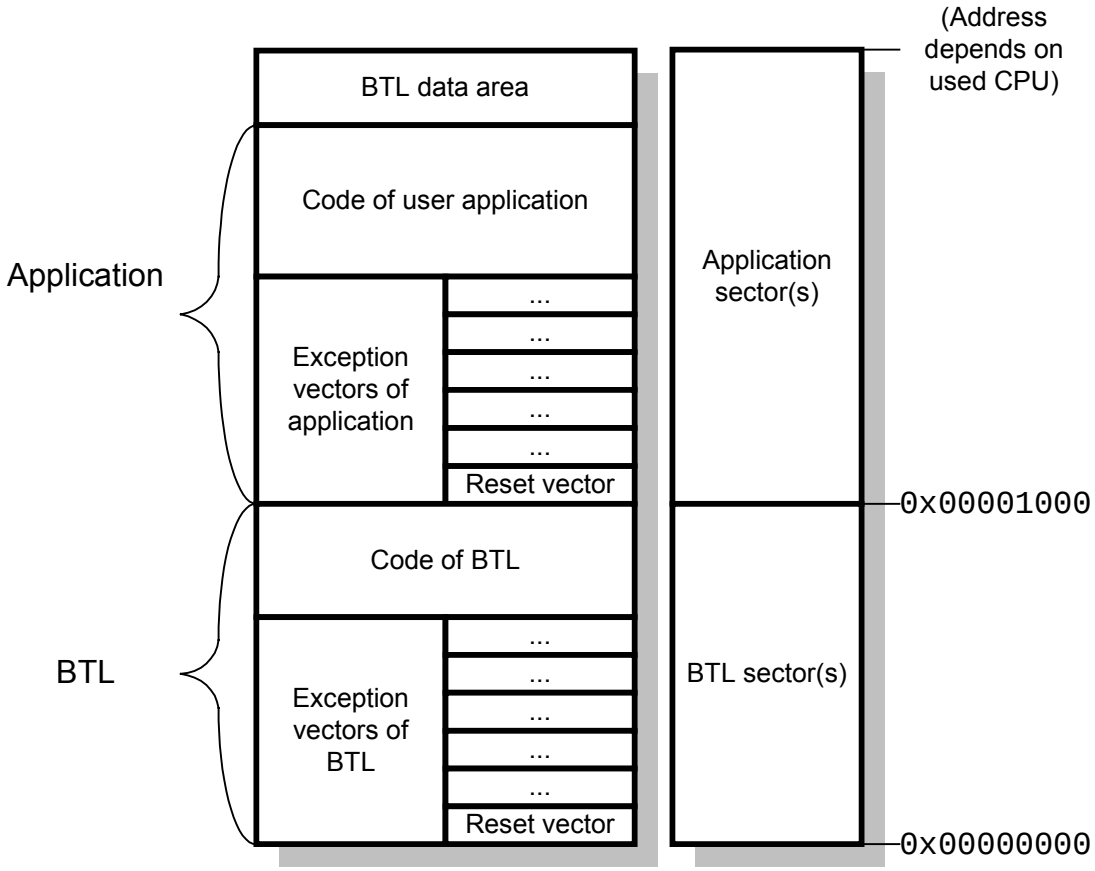
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_STM32F10x.ewd` | Project for IAR IDE |
| `BTL_STM32F10x.ewp` | “” |
| `BTL_STM32F10x_V5505.eww` | “” |
| `FLASH.icf` | Linker file for debug and release configuration |
| `USER.icf` | Linker sample file for the user firmware. |
| `CPU_ASM_IAR_V5.s` | Firmware start code |

## 9.17. Cortex-M3 STM32L15x

### 9.17.1. Supported CPU's:

STM32L15xx6 STM32L15xx8 STM32L15xxB STM32L15xxC
STM32L15xxD

### 9.17.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

## 9.17.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK            32000000L   /* [Hz]                              */
#define UARTSEL            2           /* select uart                       */
#define BAUDRATE           230400L     /* baudrate                          */

/* Data needs to be programmed in blocks of 256 bytes */
#define BTL_WRITE_BLOCK_SIZE 256

/* common defines */
#define APPNAME            "BTL STM32L15x " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset             */
                                       /* Before app. is started [ms]       */
```

## 9.17.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>2: UART 2 on STM32L152-EVAL eval board |
| BAUDRATE | Baudrate used for serial communication |

## 9.17.5. FLASH specific configuration file

The flash area is configured by the `FLASH_Config.h` file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define STM32L15XXB

/* FLASH selection */
#define FLASH_STM32L15X  1
```

## 9.17.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_STM32L15X | Use the STM32L15x flash module. |
| STM32L15XXB | Select the flash sector information for STM32L15xxB CPUs.<br>Available flash configurations are:<br>STM32L15XX6<br>STM32L15XX8<br>STM32L15XXB<br>STM32L15XXC<br>STM32L15XXD |

## 9.17.7. IAR-compiler

### 9.17.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.50.5.51995 |
| Linker | 5.50.5.51995 |
| Assembler | 5.50.5.51995 |
| Workbench | 5.8.0.1623 |

### 9.17.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.17.7.3. Additional program modules

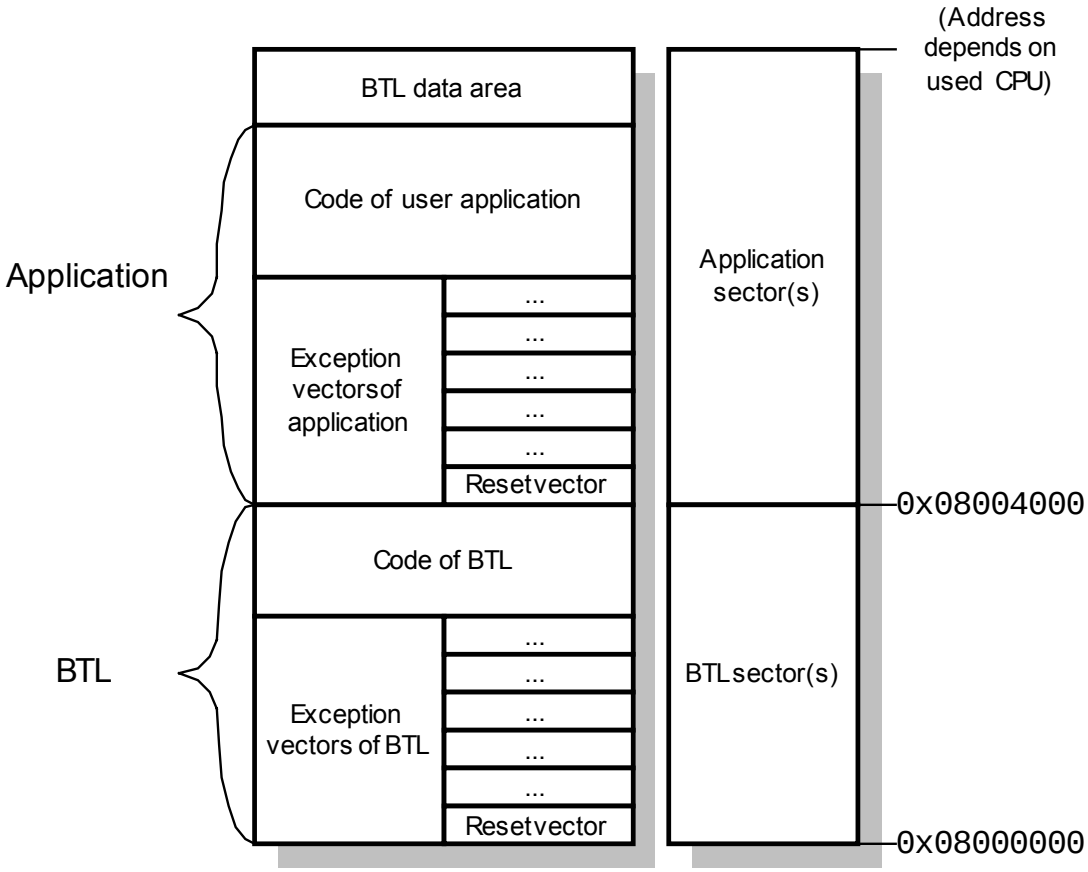
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_STM32L15x.ewd` | Project for IAR IDE |
| `BTL_STM32L15x.ewp` | “” |
| `BTL_STM32L15x_V5505.eww` | “” |
| `FLASH.icf` | Linker file for debug and release configuration |
| `USER.icf` | Linker sample file for the user firmware. |
| `CPU_ASM_IAR_V5.s` | Firmware start code |

# 9.18. Cortex-M3 STM32F20x

## 9.18.1. Supported CPU's:

STM32F20xxB STM32F20xxC STM32F20xxE STM32F20xxF
STM32F20xxG

## 9.18.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x4000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

## 9.18.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK             120000000L  // [Hz]
#define UARTSEL             0           // Select UART, STM3220G-EVAL:
                                        // 0: UART3
#define BAUDRATE            230400L     // Baudrate

//
// Data needs to be programmed in a multiple of 16 bytes.
// Greater block size means less overhead by unlock/lock operations.
//
#define BTL_WRITE_BLOCK_SIZE  256

//
// Common defines
//
#define APPNAME             "BTL STM32F20x " __DATE__ " " __TIME__
#define PASSWORD            ""
#define BTL_WAIT0_MS        500         /* Wait time after reset          */
                                        /* Before app. is started [ms]    */
```

## 9.18.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART3 on STM3220G-EVAL eval board. |
| BAUDRATE | Baudrate used for serial communication |

## 9.18.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define STM32F20XXG

/* FLASH selection */
#define FLASH_STM32F20X  1
```

## 9.18.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_STM32F20X | Use the STM32F20x flash module. |
| STM32F20XXG | Select the flash sector information for STM32F20xxG CPUs.<br>Available flash configurations are:<br>STM32F20XXB<br>STM32F20XXC<br>STM32F20XXE<br>STM32F20XXF<br>STM32F20XXG |

## 9.18.7. KEIL MDK ARM-compiler

### 9.18.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.03.0.69 |
| Linker | 5.03.0.69 |
| Assembler | 5.03.0.69 |
| uVision | 4.71.2.0 |

### 9.18.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL.

### 9.18.7.3. Additional program modules

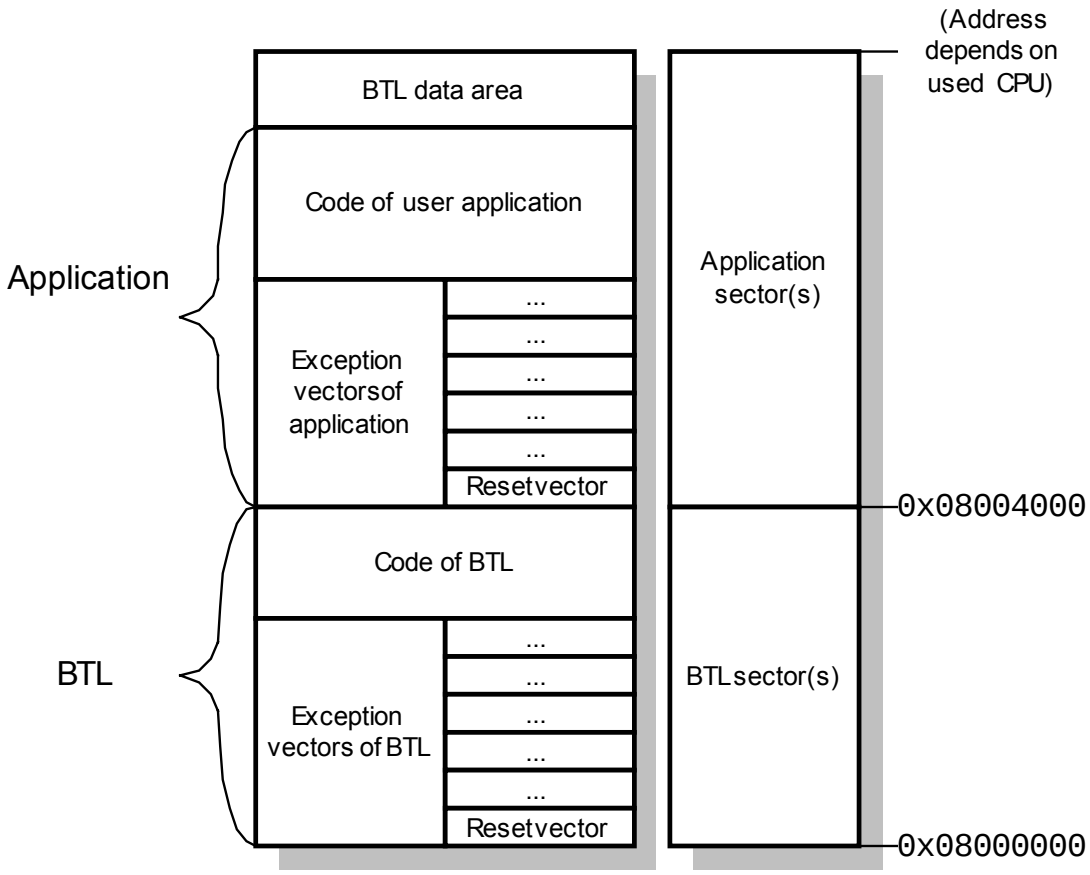
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_STM32F20x.*` | Project for KEIL MDK ARM |
| `FLASH.sct` | Linker file for debug and release configuration |
| `USER.sct` | Linker sample file for the user firmware. |

# 9.19. Cortex-M3 STM32F40x

## 9.19.1. Supported CPU's:

STM32F40xxE STM32F40xxG

## 9.19.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x4000 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

### 9.19.3. CPU specific configuration file

The BTL is configured by the `BTLCONF.H` file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK             168000000L  // [Hz]
#define UARTSEL             0           // Select UART, STM3240G-EVAL:
                                        // 0: UART3
#define BAUDRATE            230400L     // Baudrate

//
// Data needs to be programmed in a multiple of 16 bytes.
// Greater block size means less overhead by unlock/lock operations.
//
#define BTL_WRITE_BLOCK_SIZE  256

//
// Common defines
//
#define APPNAME            "BTL STM32F40x " __DATE__ " " __TIME__
#define PASSWORD           ""
#define BTL_WAIT0_MS       500         /* Wait time after reset           */
                                       /* Before app. is started [ms]     */
```

### 9.19.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>0: UART3 on STM3240G-EVAL eval board. |
| BAUDRATE | Baudrate used for serial communication |

### 9.19.5. FLASH specific configuration file

The flash area is configured by the `FLASH_Config.h` file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define STM32F40XXG

/* FLASH selection */
#define FLASH_STM32F20X  1
```

### 9.19.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_STM32F20X | Use the STM32F20x flash module (same for STM32F40x). |
| STM32F40XXG | Select the flash sector information for STM32F40xxG CPUs.<br>Available flash configurations are:<br>STM32F40XXE<br>STM32F40XXG |

## 9.19.7. IAR-compiler

### 9.19.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 6.30.6.53336 |
| Linker | 6.30.6.53336 |
| Assembler | 6.30.6.53336 |
| Workbench | 6.30.6.3387 |

### 9.19.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL. The actual BTL will be in one of the EXE-subfolders.

### 9.19.7.3. Additional program modules

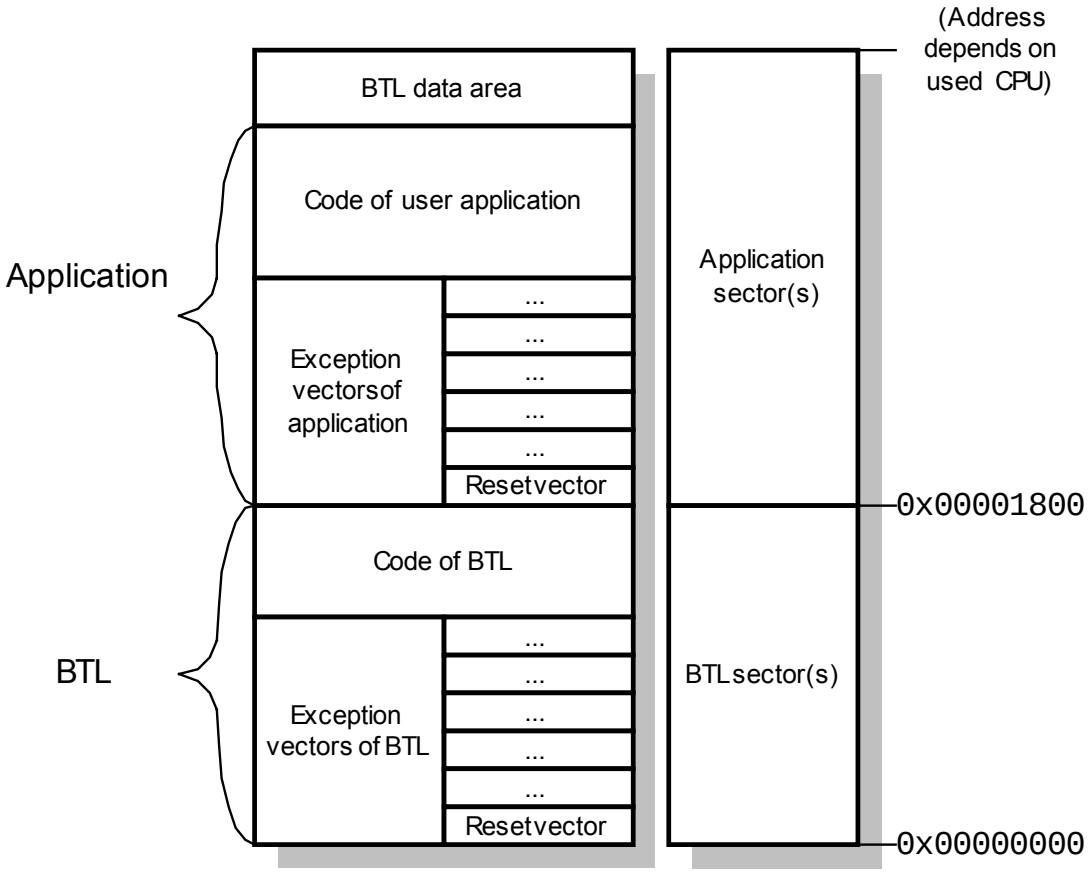
| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_STM32F40x.ewd` | Project for IAR IDE |
| `BTL_STM32F40x.ewp` | "" |
| `BTL_STM32F40x_V6306.eww` | "" |
| `FLASH.icf` | Linker file for debug and release configuration |
| `USER.icf` | Linker sample file for the user firmware. |
| `BTL_CPU_CM3_IAR_V6_ASM.s` | Firmware start code |

# 9.20. Cortex-M3 Freescale Kinetis K10

## 9.20.1. Supported CPU's:

| | | | |
|---|---|---|---|
| K10X32X5 | K10X64X5 | K10X64X7 | K10X128X5 |
| K10X128X7 | K10X256X7 | K10X512X10 | K10X512X12 |
| K10X1M0X12 | | | |

## 9.20.2. Memory map




The BTL resides in the bottom sector(s) of CPUs internal FLASH. Since the RESET vector is located in this bank, the BTL is automatically started after RESET.
The application program is moved up in memory by 0x1800 bytes. The application program can be compiled and linked the same way as without BTL; you only have to change the memory location.
The BTL data area resides at the top of the application sectors.

## 9.20.3. CPU specific configuration file

The BTL is configured by the BTLCONF.H file.
This file is self explaining and may look like the following:

```
/* CPU and UART specific defines */
#define UPCLOCK             41943040L   // [Hz]
#define UARTSEL             1           // Select UART: 1: UART1
#define BAUDRATE            230400L     // Baudrate

//
// Data needs to be programmed in a multiple of 16 bytes.
// Greater block size means less overhead by unlock/lock operations.
//
#define BTL_WRITE_BLOCK_SIZE  64

//
// Common defines
//
#define APPNAME             "BTL Kinetis Kxx " __DATE__ " " __TIME__
#define PASSWORD            ""
#define BTL_WAIT0_MS        500         /* Wait time after reset          */
                                        /* Before app. is started [ms]    */
```

## 9.20.4. CPU specific configuration parameters:

| Parameter | Meaning |
|---|---|
| UPCLOCK | Microprocessor clock frequency [Hz]. |
| UARTSEL | UART Selection. Should be:<br>1: UART1. |
| BAUDRATE | Baudrate used for serial communication |

## 9.20.5. FLASH specific configuration file

The flash area is configured by the FLASH_Config.h file.
This file is self explaining and may look like the following:

```
/* Use BTL.h for the basic type definition */
#include "BTL.h"

/* FLASH specific data types */
#define FLASH_U32 U32
#define FLASH_U16 U16
#define FLASH_U8  U8

/* Define CPU type */
#define K10X128X5

/* FLASH selection */
#define FLASH_KINETIS_KXX  1
```

## 9.20.6. FLASH specific configuration parameters:

| Parameter | Meaning |
|---|---|
| FLASH_KINETIS_KXX | Use the Freescale Kinetis Kxx flash module. |
| K10X128X5 | Select the flash sector information for Kinetis Freescale K10 CPUs.<br>Available flash configurations are:<br>K10X32X5<br>K10X64X5<br>K10X64X7<br>K10X128X5<br>K10X128X7<br>K10X256X7<br>K10X512X10<br>K10X512X12<br>K10X1M0X12 |

## 9.20.7. KEIL MDK ARM-compiler

### 9.20.7.1. Used tools

| Tool | Version |
|---|---|
| Compiler | 5.03.0.69 |
| Linker | 5.03.0.69 |
| Assembler | 5.03.0.69 |
| uVision | 4.71.2.0 |

### 9.20.7.2. Compiling and linking

The project file should be opened by double click from the Windows Explorer. Select the target which should be build. Now you can modify `BTLCONF.H` and rebuild the BTL.

### 9.20.7.3. Additional program modules

| File | Explanation |
|---|---|
| `Clean.bat` | Removes the compiler output |
| `BTL_Kinetis_Kxx.*` | Project for KEIL MDK ARM |
| `FLASH.sct` | Linker file for debug and release configuration |
| `USER.sct` | Linker sample file for the user firmware. |

# 10. Index