

STM32F4 CMSIS

Feabhas

Published
with GitBook



Table of Contents

1. [Introduction](#)
2. [Building an Initial Cortex M4 Project](#)
 - i. [Creating a default Cortex-M4 Project](#)
 - ii. [Customising the project for the STM32F407VG Processor](#)
 - i. [mem.ld differences](#)
 - ii. [sections.ls differences](#)
 - iii. [Compiler Flags](#)
 - i. [Example main.d](#)
 - iv. [Linker Flags](#)
 - i. [nano.specs](#)
 - ii. [mem.ld](#)
 - iii. [sections.ld](#)
3. [Running the project](#)
4. [The Boot Sequence](#)
 - i. [_estack](#)
 - ii. [Reset Handler](#)
 - i. [__initialize_hardware_early](#)
 - ii. [__initialize_hardware](#)
 - iii. [Possible addition to boot sequence](#)
5. [SysTick](#)
6. [Standard IO and Debug IO](#)
7. [GPIO](#)
 - i. [GPIO Output](#)
 - ii. [GPIO Input](#)
8. [Serial IO](#)
 - i. [USART_init](#)
 - ii. [GPIOB AF Pins Setup](#)
9. [Interrupts](#)
 - i. [GPIO as an external Interrupt](#)
10. [FreeRTOS](#)
11. [feabOS](#)
 - i. [feabOS example main.c](#)

Building a Minimal C Project for the STM32F4 Processor using CMSIS

Most examples for the STM32 family utilise the ST HAL library. This code is very useful for seeing how peripherals are configured, but sometimes can get in the way of understanding really what is going on.

This book attempts to strip a C project back to its bare minimum using just CMSIS rather than the ST HAL.

It is built using an Eclipse project and the GNU Tools for ARM Embedded Processors (`arm-none-eabi-gcc`)

For setting up Eclipse and the GNU ARM toolchain, then I recommend following the wonderful instructions at the [GNU ARM Eclipse](#) website.

Building an initial Cortex M4 Project

gcc version 4.9.3

gcc version 4.9.3 20141119 (release) [ARM/embedded-4_9-branch revision 218278] (GNU Tools for ARM Embedded Processors)

Create Image Settings

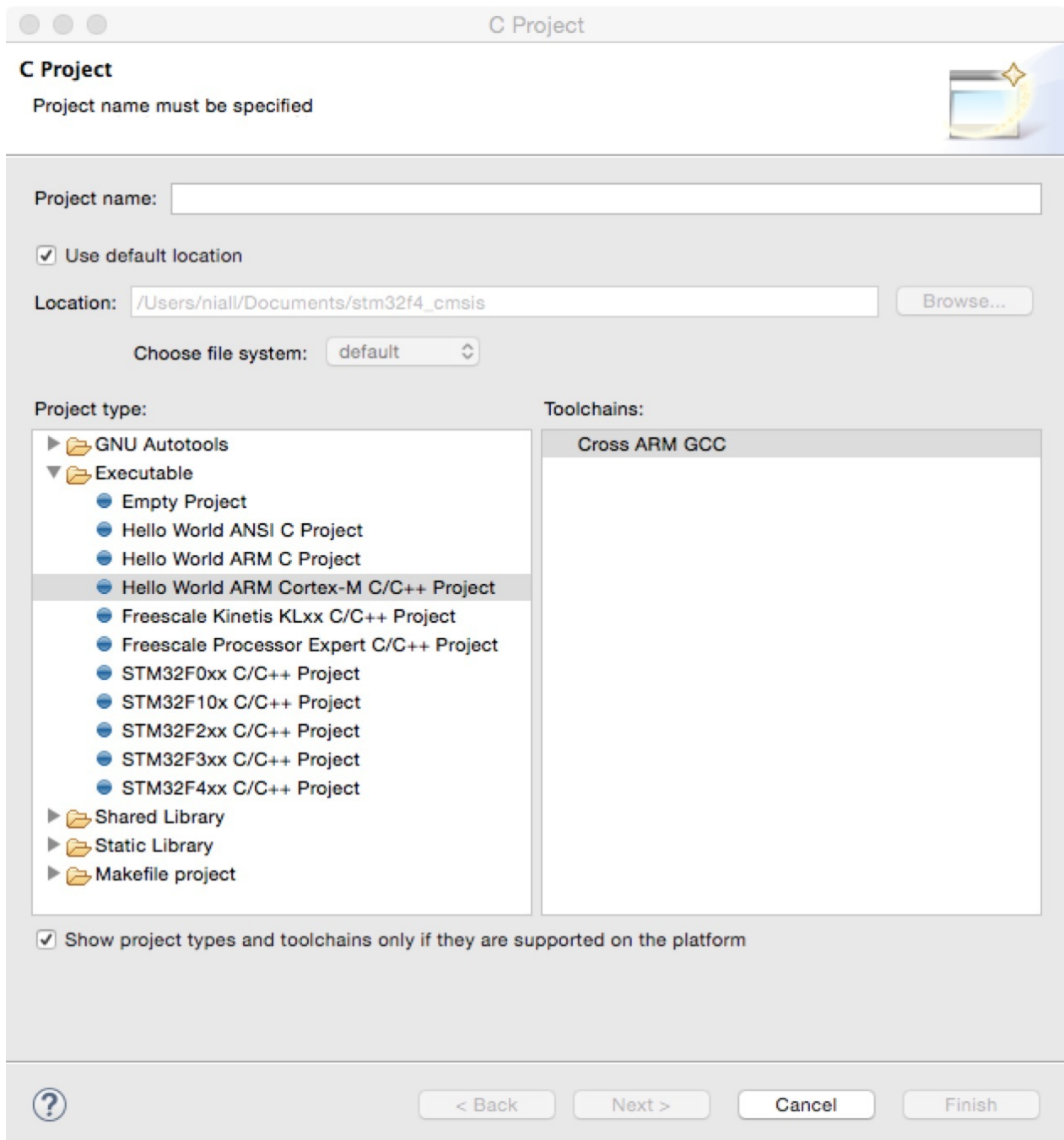
```
Invoking: Cross ARM GNU Create Flash Image  
arm-none-eabi-objcopy -O ihex "minimal.elf" "minimal.hex"  
Finished building: minimal.hex
```

```
-O ihex "minimal.elf"
```

Building a Default Cortex-M4 Project

Eclipse

- **File -> New -> C Project**
- Select *"Hello World ARM Cortex-M C/C++ Project"*
 - ensure that the toolchain is *"Cross ARM GCC"*



- Give the project a name (i.e. **minimal**)
- Use the **defaults** for the project as we'll override them later anyway for our processor
- Once finished you will have a project with the following file structure

• Project File Structure

```

.
├── files.txt
├── include
│   └── Timer.h
├── ldscripts
│   ├── libs.ld
│   ├── mem.ld
│   └── sections.ld
├── src
│   ├── Timer.c
│   ├── _write.c
│   └── main.c
└── system
    ├── include
    │   ├── DEVICE
    │   │   └── README_DRIVERS.txt
    │   ├── arm
    │   │   └── semihosting.h
    │   └── cmsis
    │       ├── DEVICE.h
    │       ├── README_CMSIS.txt
    │       ├── README_DEVICE.txt
    │       ├── arm_common_tables.h
    │       ├── arm_const_structs.h
    │       ├── arm_math.h
    │       ├── cmsis_device.h
    │       ├── core_cm0.h
    │       ├── core_cm0plus.h
    │       ├── core_cm3.h
    │       ├── core_cm4.h
    │       ├── core_cm4_simd.h
    │       ├── core_cmFunc.h
    │       ├── core_cmInstr.h
    │       └── core_sc000.h

```

```
| | | └─ core_sc300.h
| | | └─ system_DEVICE.h
| | └─ cortexm
| |   └─ ExceptionHandlers.h
| └─ diag
|   └─ Trace.h
└─ src
    └─ DEVICE
        └─ README_DRIVERS.txt
    └─ cmsis
        └─ README_DEVICE.txt
        └─ system_DEVICE.c
        └─ vectors_DEVICE.c
    └─ cortexm
        └─ _initialize_hardware.c
        └─ _reset_hardware.c
        └─ exception_handlers.c
    └─ diag
        └─ Trace.c
        └─ trace_impl.c
    └─ newlib
        └─ README.txt
        └─ _cxx.cpp
        └─ _exit.c
        └─ _sbrk.c
        └─ _startup.c
        └─ _syscalls.c
        └─ assert.c
```

16 directories, 45 files

Customising the project for the STM32F407VG Processor

The files that need replacing are:

```
.
├── system
│   ├── include
│   │   ├── cmsis
│   │   │   ├── DEVICE.h
│   │   │   └── system_DEVICE.h
│   └── src
│       ├── cmsis
│       │   ├── system_DEVICE.c
│       │   └── vectors_DEVICE.c
```

STM32F4 CMSIS Files

The files can be found [here](#) but are also part of the standard ARM GNU installation

Headers to add

- system_stmf4xx.h
- stm32f4xx.h
- stm32f407xx.h

Also change `cmsis_device.h` to include `stm32f4xx.h` rather than `DEVICE.h`

C Source Files to add

- system_stm32f4xx.c
- vectors_stm32f4xx.c

Also **exclude** from build

- *system_DEVICE.c*
- *vectors_DEVICE.c*

Overall changes to file tree

```
.
├── system
│   ├── include
│   │   ├── cmsis
│   │   │   ├── stm32f407xx.h
│   │   │   ├── stm32f4xx.h
│   │   │   └── system_stm32f4xx.h
│   └── src
│       ├── cmsis
│       │   ├── system_stm32f4xx.c
│       │   └── vectors_stm32f4xx.c
```

Other changes

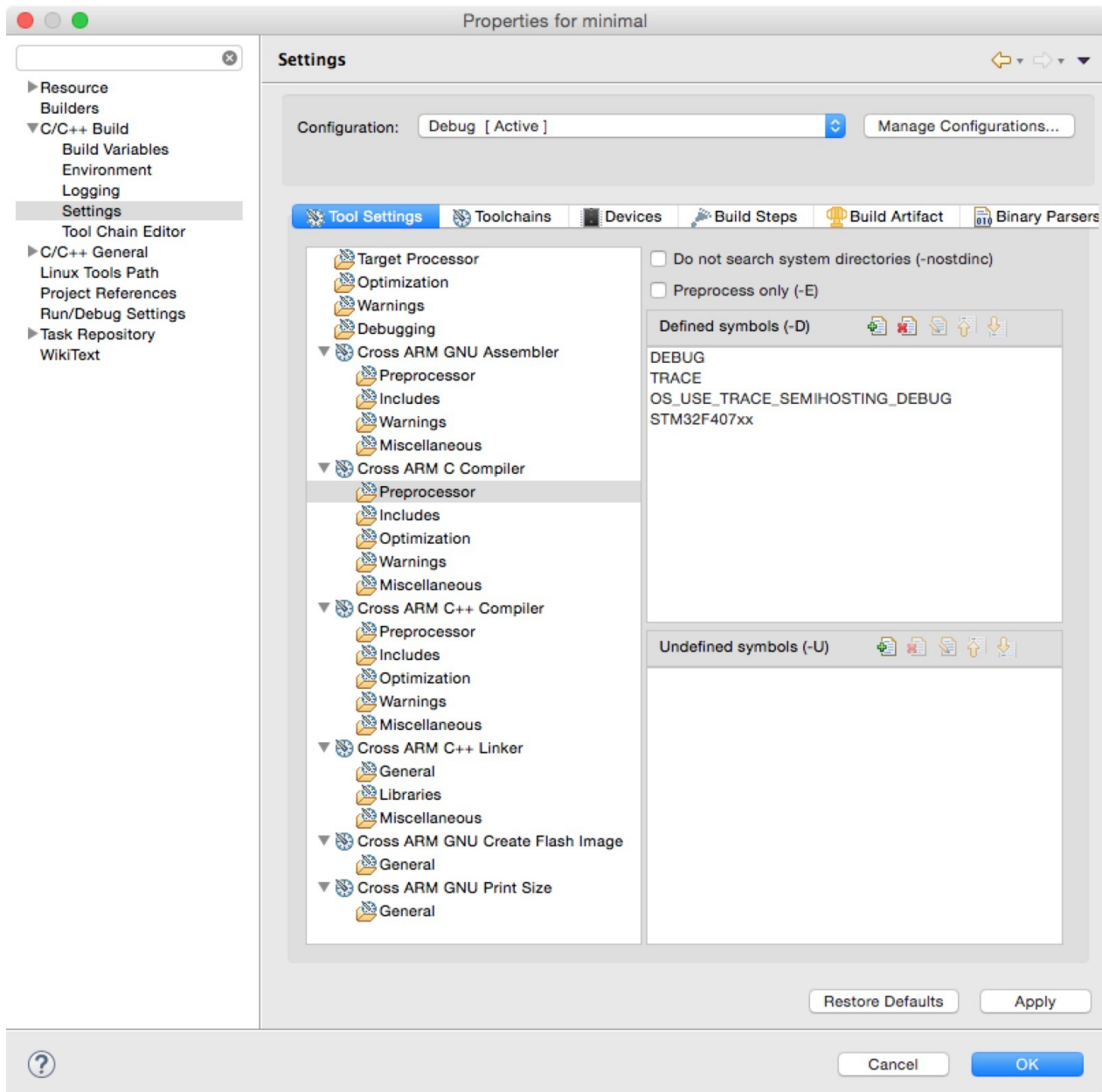
Linker Configuration Files to modify

Update files:

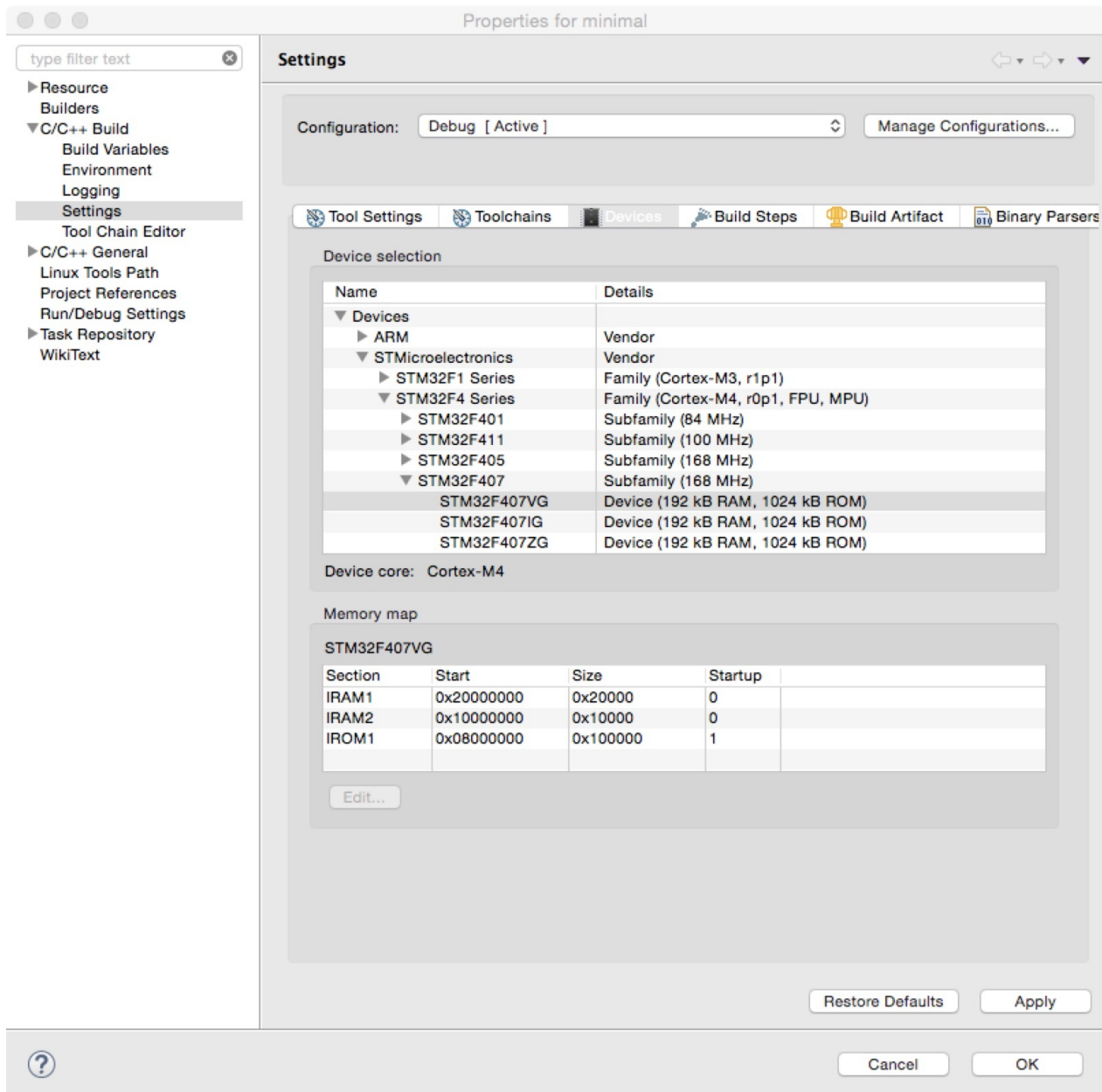
- mem.ld
- sections.ld

Project Configuration

- Add Preprocessor Directive
 - STM32F407xx



- Setup the Device
 - STM32F407VG



```

16,24c16,18
<  FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 128K
<  RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K
<
<  /*
<   * Optional sections; define the origin and length to match
<   * the the specific requirements of your hardware. The zero
<   * length prevents inadvertent allocation.
<   */
<  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 0
---
>  RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
>  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
>  FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
29a24
>  MEMORY_ARRAY (xrw) : ORIGIN = 0x20002000, LENGTH = 32
34,37c29,31
<  *  RAM (xrw) : ORIGIN = 0x64000000, LENGTH = 2048K
<  *
<  * For special RAM areas use something like:
<  *  MEMORY_ARRAY (xrw) : ORIGIN = 0x20002000, LENGTH = 32
---
>
>  RAM (xrw) : ORIGIN = 0x64000000, LENGTH = 2048K
>

```

```

56c57
<      .isr_vector :
---
>      .isr_vector : ALIGN(4)
58c59,61
<          . = ALIGN(4);
---
>          FILL(0xFF)
>
>      __vectors_start__ = ABSOLUTE(.) ;
70d72
<          . = ALIGN(4);
73c75
<      .inits :
---
>      .inits : ALIGN(4)
75c77,119
<          . = ALIGN(4);
---
>      /*
>      * Memory regions initialisation arrays.
>      *
>      * There are two kinds of arrays for each RAM region, one
>      * data and one for bss. Each is iterated at startup and
>      * region initialisation is performed.
>      *
>      * The data array includes:
>      * - from (LOADADDR())
>      * - region_begin (ADDR())
>      * - region_end (ADDR()+SIZEOF())
>      *
>      * The bss array includes:
>      * - region_begin (ADDR())
>      * - region_end (ADDR()+SIZEOF())
>      *
>      * WARNING: It is mandatory that the regions are word ali
>      * since the initialisation code works only on words.
>      */
>
>      __data_regions_array_start = .;
>
>      LONG(LOADADDR(.data));
>      LONG(ADDR(.data));

```

```

>         LONG(ADDR(.data)+SIZEOF(.data));
>
>         LONG(LOADADDR(.data_CCMRAM));
>         LONG(ADDR(.data_CCMRAM));
>         LONG(ADDR(.data_CCMRAM)+SIZEOF(.data_CCMRAM));
>
>         __data_regions_array_end = .;
>
>         __bss_regions_array_start = .;
>
>         LONG(ADDR(.bss));
>         LONG(ADDR(.bss)+SIZEOF(.bss));
>
>         LONG(ADDR(.bss_CCMRAM));
>         LONG(ADDR(.bss_CCMRAM)+SIZEOF(.bss_CCMRAM));
>
>         __bss_regions_array_end = .;
>
>         /* End of memory regions initialisation arrays. */
131d174
<         . = ALIGN(4);
139c182
<         .flashtext :
---
>         .flashtext : ALIGN(4)
141d183
<         . = ALIGN(4);
143d184
<         . = ALIGN(4);
151c192
<         .text :
---
>         .text : ALIGN(4)
153,154d193
<         . = ALIGN(4);
<
157c196,197
<         *(.rodata .rodata.*)          /* read-only data (constants) */
---
>         /* read-only data (constants) */
>         *(.rodata .rodata.* .constdata .constdata.*)
176c216
<         .ARM.extab :
---
```

```

>      .ARM.extab : ALIGN(4)
180a221
>      . = ALIGN(4);
182c223
<      .ARM.exidx :
---
>      .ARM.exidx : ALIGN(4)
192,197d232
<      /*
<      * This address is used by the startup code to
<      * initialise the .data section.
<      */
<      _sidata = _etext;
<
206a242,257
>      * The secondary initialised data section.
>      */
>      .data_CCMRAM : ALIGN(4)
>      {
>          FILL(0xFF)
>          *(.data.CCMRAM .data.CCMRAM.*)
>          . = ALIGN(4) ;
>      } > CCMRAM AT>FLASH
>
>      /*
>      * This address is used by the startup code to
>      * initialise the .data section.
>      */
>      _sidata = LOADADDR(.data);
>
>      /*
213c264
<      .data : AT ( _sidata )
---
>      .data : ALIGN(4)
215,216c266
<      . = ALIGN(4);
<
---
>          FILL(0xFF)
231c281
<      } >RAM
---
>      } >RAM AT>FLASH

```

```

235c285
<      * The uninitialised data section. NOLOAD is used to avoid
---
>      * The uninitialised data sections. NOLOAD is used to avoid
238c288,296
<      .bss (NOLOAD) :
---
>
>      /* The secondary uninitialised data section. */
>      .bss_CCMRAM (NOLOAD) : ALIGN(4)
>      {
>          *(.bss.CCMRAM .bss.CCMRAM.*)
>      } > CCMRAM
>
>      /* The primary uninitialised data section. */
>      .bss (NOLOAD) : ALIGN(4)
240d297
<          . = ALIGN(4);
254c311,316
<      .noinit (NOLOAD) :
---
>      .noinit_CCMRAM (NOLOAD) : ALIGN(4)
>      {
>          *(.noinit.CCMRAM .noinit.CCMRAM.*)
>      } > CCMRAM
>
>      .noinit (NOLOAD) : ALIGN(4)
256d317
<          . = ALIGN(4);
277c338
<      ._check_stack :
---
>      ._check_stack : ALIGN(4)
279,280d339
<          . = ALIGN(4);
<
282,283d340
<
<          . = ALIGN(4);
286,290d342
<      .bss_CCMRAM : ALIGN(4)
<      {
<          *(.bss.CCMRAM .bss.CCMRAM.*)
<      } > CCMRAM

```




```
<
296c348
<      .b1text :
---
>      .b1text : ALIGN(4)
310c362
<      .eb0text :
---
>      .eb0text : ALIGN(4)
318c370
<      .eb1text :
---
>      .eb1text : ALIGN(4)
326c378
<      .eb2text :
---
>      .eb2text : ALIGN(4)
334c386
<      .eb3text :
---
>      .eb3text : ALIGN(4)
```

Compiler flags

make

Example output from compilation step:

```
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Og -fmessage-length=0 -f
```



arm-none-eabi-gcc

This is our cross compiler

-m

-mcpu=cortex-m4

This specifies the name of the target ARM processor. GCC uses this name to derive the name of the target ARM architecture (as if specified by -march) and the ARM processor type for which to tune for performance (as if specified by -mtune).

Sets the target core to Cortex-M4. This is important if SIMD/FPU operations are required.

-mthumb

Select between generating code that executes in ARM and Thumb states. The default for most configurations is to generate code that executes in ARM state.

This may be redundant if the cpu is set to Cortex-M4?

-Og

In `gcc v4.8` a new general optimization level, `-Og`, was introduced. It

addressed the need for faster compilation and a superior debugging experience while providing a reasonable level of run-time performance. Overall experience for development should be better than the default optimization level `-O0`.

-f

-fmessage-length= 0

This diagnostic option causes `gcc` to format diagnostic messages so that they fit on lines of `n` characters or less, inducing line wrapping in messages that are greater than `n` characters. The default value for `n` is `72` characters for C++ messages and `0` for all other front ends supported by GCC. If `n` is `0`, no line wrapping will be done.

-fsigned-char

Permits the `char` type to be signed, as in the type `signed char`.

-ffunction-sections

This optimization option, for output targets that support arbitrary code sections, causes `gcc` to place each function into its own section in the output file. The name of the function determines the name of the section in the output file. This option is typically used on systems whose linkers can perform optimizations that improve the locality of reference in the instruction space.

Using this option causes the assembler and linker to create larger object and executable files that may therefore be slower on some systems.

Using this option also prevents the use of `gprof` on some systems and may cause problems if you are also compiling with the `-g` option.

-fdata-sections

This optimization option, for output targets that support arbitrary code sections, causes `gcc` to place each data item into its own section in the output file. The

name of the data item determines the name of the section in the output file.

Same issues as with `-ffunction-sections`

-fno-move-loop-invariants

This optimization option causes `gcc` to move all invariant computations in loops outside the loop.

This option was formerly known as `-fmove-all-movables`.

This will prevent the compiler to move some constant parts of the loop outside it and the execution flow will be more easy to follow during debugging.

Starting with 4.9, the `-fno-move-loop-invariants` is added automatically to `-Og`.

-W

-Wall

This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.

-Wextra

This enables some extra warning flags that are not enabled by `-Wall`.

This option used to be called ``-W``. The older name is still supported

-g3

Produce debugging information in the operating system's native format (`stabs`, `COFF`, `XCOFF`, or `DWARF 2`). GDB can work with this debugging information.

On most systems that use stabs format, `-g` enables use of extra debugging information that only `GDB` can use; this extra information makes debugging work better in `GDB` but probably makes other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use `-gstabs+` , `-gstabs` , `-gxcoff+` , `-gxcoff` , or `-gvms` .

`gcc` allows you to use `-g` with `-O` . The shortcuts taken by optimized code may occasionally produce surprising results:

- some variables you declared may not exist at all;
- flow of control may briefly move where you did not expect it;
- some statements may not be executed because they compute constant results or their values are already at hand;
- some statements may execute in different places because they have been moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

The following options are useful when `gcc` is generated with the capability for more than one debugging format.

-D

Project specific preprocessor directive used for conditional compilations.

- `-DDEBUG`
- `-DTRACE`
- `-DOS_USE_TRACE_SEMIHOSTING_DEBUG`
- `-DSTM32F407xx`

-I

Project specific header include paths for compiler searching

- `-I"./include"`
- `-I"./system/include"`
- `-I"./system/include/cmsis"`

- `-I"./system/include/DEVICE"`

-std

As of **v4.9.3**, `gcc` supports three versions of the C standard, although support for the most recent version is not yet complete.

The original ANSI C standard (`X3.159-1989`) was ratified in 1989 and published in 1990. This standard was ratified as an ISO standard (`ISO/IEC 9899:1990`) later in 1990. There were no technical differences between these publications, although the sections of the ANSI standard were renumbered and became clauses in the ISO standard. This standard, in both its forms, is commonly known as `c89` , or occasionally as `c90` , from the dates of ratification. The ANSI standard, but not the ISO standard, also came with a Rationale document. To select this standard in `gcc` , use one of the options `-ansi` , `-std=c90` or `-std=iso9899:1990` ; to obtain all the diagnostics required by the standard, you should also specify `-pedantic` (or `-pedantic-errors` if you want them to be errors rather than warnings).

Errors in the 1990 ISO C standard were corrected in two *Technical Corrigenda* published in 1994 and 1996. `gcc` does not support the uncorrected version.

An amendment to the 1990 standard was published in 1995. This amendment added digraphs and `__STDC_VERSION__` to the language, but otherwise concerned the library. This amendment is commonly known as **AMD1** ; the amended standard is sometimes known as `C94` or `C95` . To select this standard in `gcc` , use the option `-std=iso9899:199409` (with, as for other standard versions, `-pedantic` to receive all required diagnostics).

A new edition of the ISO C standard was published in 1999 as `ISO/IEC 9899:1999` , and is commonly known as `C99` . `gcc` has substantially complete support for this standard version; see [C99 Status](#) for details. To select this standard, use `-std=c99` or `-std=iso9899:1999` .

Errors in the 1999 ISO C standard were corrected in three *Technical Corrigenda* published in 2001, 2004 and 2007. `GCC` does not support the uncorrected version.

A fourth version of the C standard, known as **C11**, was published in 2011 as **ISO/IEC 9899:2011**. GCC has substantially complete support for this standard, enabled with **-std=c11** or **-std=iso9899:2011**.

-std=gnuXX

By default, **gcc** provides some extensions to the C language that on rare occasions conflict with the C standard. See [Extensions to the C Language Family](#). Use of the **-std** options listed above will disable these extensions where they conflict with the C standard version selected. You may also select an extended version of the C language explicitly with **-std=gnu90** (for C90 with GNU extensions), **-std=gnu99** (for C99 with GNU extensions) or **-std=gnu11** (for C11 with GNU extensions).

The **default**, if no C language dialect options are given, is **-std=gnu90**; *this will change to **-std=gnu11** in some future release when.*

- **-std=gnu99**

-M

-MMD

This preprocessor option causes **GCC** to generate the same output rule as that produced by the **-M** option, the difference being that the generated rules do not list user include files or other user include files that are included by user include files.

-MP

This preprocessor option causes **GCC** to add a fake target for each dependency other than the main file, causing each to depend on nothing through a dummy rule. These rules work around errors that may be generated by the make program if you remove header files without making corresponding Makefile updates.

-MF"src/main.d"

This preprocessor option along with the `-M` or `-MM` options identifies the name of a file to which GCC should write dependency information.

[Example main.d](#)

`-MT"src/main.o"`

This preprocessor option causes GCC to change the output target in the rule emitted by dependency-rule generation. Instead of following the standard extension-substitution naming convention, using this option sets the name of the output file to the exact filename that you specify.

`-C`

Compile the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

By default, the object file name for a source file is made by replacing the suffix `.c` , `.i` , `.s` , etc., with `.o` .

`-o "src/main.o"`

Example of generated `main.d`

```
src/main.o: ../src/main.c ../system/include/diag/Trace.h \  
  ../include/Timer.h ../system/include/cmsis/cmsis_device.h \  
  ../system/include/cmsis/stm32f4xx.h \  
  ../system/include/cmsis/stm32f407xx.h ../system/include/cmsis/core_cm4.h \  
  ../system/include/cmsis/core_cmInstr.h \  
  ../system/include/cmsis/core_cmFunc.h \  
  ../system/include/cmsis/core_cm4_simd.h \  
  ../system/include/cmsis/system_stm32f4xx.h \  
  ../system/include/cmsis/stm32f4xx.h ../src/gpio_port_d.h  
  
../system/include/diag/Trace.h:  
  
../include/Timer.h:  
  
../system/include/cmsis/cmsis_device.h:  
  
../system/include/cmsis/stm32f4xx.h:  
  
../system/include/cmsis/stm32f407xx.h:  
  
../system/include/cmsis/core_cm4.h:  
  
../system/include/cmsis/core_cmInstr.h:  
  
../system/include/cmsis/core_cmFunc.h:  
  
../system/include/cmsis/core_cm4_simd.h:  
  
../system/include/cmsis/system_stm32f4xx.h:  
  
../system/include/cmsis/stm32f4xx.h:  
  
../src/gpio_port_d.h:
```

Linker flags

Example output from linking step:

```
Invoking: Cross ARM C++ Linker
arm-none-eabi-g++ -mcpu=cortex-m4 -mthumb -Og -fmessage-length=0 -f
Finished building target: minimal.elf
```

Additional flags above and beyond the compiler flags:

```
-T mem.ld -T libs.ld -T sections.ld -nostartfiles -Xlinker --gc-sec
```

-T script

Use *script* as the linker script.

- *-T mem.ld*
- *-T libs.ld*
- *-T sections.ld*

-nostartfiles

Do not use the standard system startup files when linking. The standard system libraries are used normally, unless `-nostdlib` or `-nodefaultlibs` is used.

-Xlinker option

Pass *option* as an option to the linker. You can use this to supply system-specific linker options that GCC does not recognize.

If you want to pass an option that takes a separate argument, you must use `-Xlinker` twice, once for the option and once for the argument. *For example, to pass `-assert definitions`, you must write `-Xlinker -assert -Xlinker definitions`. It does not*

work to write `-Xlinker "-assert definitions"`, because this passes the entire string as a single argument, which is not what the linker expects.

When using the GNU linker, it is usually more convenient to pass arguments to linker options using the `option=value` syntax than as separate arguments. For example, you can specify `-Xlinker -Map=output.map` rather than `-Xlinker -Map -Xlinker output.map`.

-Xlinker --gc-sections

There are many *library* functions provided in common source files. All code and data is currently linked into every executable, and most of the images do not use most of the functions. Significant SRAM could be saved if only the required code and data are included.

Using the GCC `-ffunction-sections` and LD `--gc-sections` directives automatically only include used code and data for C sources. This requires linker scripts to include any new input sections.

-Ldir

Adds directory *dir* to the list of directories to be searched for by `-l`.

- `-L"./ldscripts"`

-Wl,option

Passes *option* as an option to the linker. If option contains commas, it is split into multiple options at the commas. You can use this syntax to pass an argument to the option.

For example, `-Wl, -Map,output.map` passes `-Map output.map` to the linker. When using the GNU linker, you can also get the same effect with `-Wl, -Map=output.map`.

-Wl,-Map,"minimal.map"

Generate map file called "minimal.map" based on a project name of *minimal*.

--specs=Spec file

The *spec* strings built into GCC can be overridden by using the `-specs=` command-line switch to specify a *spec file*.

Spec files are plain text files that are used to construct *spec strings*. They consist of a sequence of directives separated by blank lines.

--specs=nano.specs

By adding `-specs=nano.specs` to the gcc link command, a reduced-size libc is linked in (**libc_nano**). The effect of this on the final program size is significant.

Using **libc_nano** on a minimal project:

```
Invoking: Cross ARM GNU Print Size
arm-none-eabi-size --format=berkeley "minimal.elf"
  text      data      bss      dec      hex      filename
  4277       176       416     4869     1305     minimal.elf
Finished building: minimal.siz
```

Without using *libc_nano*:

```
Invoking: Cross ARM GNU Print Size
arm-none-eabi-size --format=berkeley "minimal.elf"
  text      data      bss      dec      hex      filename
 21472      2240       460    24172     5e6c     minimal.elf
Finished building: minimal.siz
```

This file (*nano.spec*) is part of the compiler toolchain.

[Example nano.spec file](#)

nano.specs

```
%rename link nano_link
%rename link_gcc_c_sequence nano_link_gcc_c_sequence

*nano_libc:
-lc_nano

*nano_libgloss:
%{specs=rdimon.specs:-lrdimon_nano} %{specs=nosys.specs:-lnosys}

*link_gcc_c_sequence:
%(nano_link_gcc_c_sequence) --start-group %G %(nano_libc) %(nano_li

*link:
%(nano_link) %:replace-outfile(-lc -lc_nano) %:replace-outfile(-lg

*lib:
%{!shared:%{g*:-lg_nano} %p:%{!pg:-lc_nano}}%{p:-lc_p}%{pg:-lc_p}
```

mem.ld

```

/*
 * Memory Spaces Definitions.
 *
 * Need modifying for a specific board.
 *   FLASH.ORIGIN: starting address of flash
 *   FLASH.LENGTH: length of flash
 *   RAM.ORIGIN: starting address of RAM bank 0
 *   RAM.LENGTH: length of RAM bank 0
 *
 * The values below can be addressed in further linker scripts
 * using functions like 'ORIGIN(RAM)' or 'LENGTH(RAM)'.
 */

MEMORY
{
  RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
  FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
  FLASHB1 (rx) : ORIGIN = 0x00000000, LENGTH = 0
  EXTMEMB0 (rx) : ORIGIN = 0x00000000, LENGTH = 0
  EXTMEMB1 (rx) : ORIGIN = 0x00000000, LENGTH = 0
  EXTMEMB2 (rx) : ORIGIN = 0x00000000, LENGTH = 0
  EXTMEMB3 (rx) : ORIGIN = 0x00000000, LENGTH = 0
  MEMORY_ARRAY (xrw) : ORIGIN = 0x20002000, LENGTH = 32
}

/*
 * For external ram use something like:
 *
  RAM (xrw) : ORIGIN = 0x64000000, LENGTH = 2048K
 */

```

sections.ld

```

/*
 * Default linker script for STM32Fxxx.
 */

/*
 * The '__stack' definition is required by crt0, do not remove it.
 */
__stack = ORIGIN(RAM) + LENGTH(RAM);

_estack = __stack;      /* STM specific definition */

/*
 * Default stack sizes.
 * These are used by the startup in order to allocate stacks
 * for the different modes.
 */

__Main_Stack_Size = 1024 ;

PROVIDE ( _Main_Stack_Size = __Main_Stack_Size ) ;

__Main_Stack_Limit = __stack - __Main_Stack_Size ;

/* "PROVIDE" allows to easily override these values from an
 * object file or the command line. */
PROVIDE ( _Main_Stack_Limit = __Main_Stack_Limit ) ;

/*
 * There will be a link error if there is not this amount of
 * RAM free at the end.
 */
_Minimum_Stack_Size = 256 ;

/*
 * Default heap definitions.
 * The heap start immediately after the last statically allocated
 * .sbss/.noinit section, and extends up to the main stack limit.
 */
PROVIDE ( _Heap_Begin = _end_noinit ) ;

```

```

PROVIDE ( _Heap_Limit = __stack - __Main_Stack_Size ) ;

/*
 * The entry point is informative, for debuggers and simulators,
 * since the Cortex-M vector points to it anyway.
 */
ENTRY(_start)

/* Sections Definitions */

SECTIONS
{
    /*
     * For Cortex-M devices, the beginning of the startup code is s
     * the .isr_vector section, which goes to FLASH.
     */
    .isr_vector : ALIGN(4)
    {
        FILL(0xFF)

        __vectors_start__ = ABSOLUTE(.) ;
        KEEP(*(.isr_vector))           /* Interrupt vectors */

        KEEP(*(.cfmconfig))           /* Freescale configuration w

    /*
     * This section is here for convenience, to store the
     * startup code at the beginning of the flash area, hoping
     * this will increase the readability of the listing.
     */
    *(.after_vectors .after_vectors.*) /* Startup code and I

} >FLASH

.inits : ALIGN(4)
{
    /*
     * Memory regions initialisation arrays.
     *
     * There are two kinds of arrays for each RAM region, one fo
     * data and one for bss. Each is iterated at startup and t
     * region initialisation is performed.
     *

```



```

* The data array includes:
* - from (LOADADDR())
* - region_begin (ADDR())
* - region_end (ADDR()+SIZEOF())
*
* The bss array includes:
* - region_begin (ADDR())
* - region_end (ADDR()+SIZEOF())
*
* WARNING: It is mandatory that the regions are word aligned
* since the initialisation code works only on words.
*/

__data_regions_array_start = .;

LONG(LOADADDR(.data));
LONG(ADDR(.data));
LONG(ADDR(.data)+SIZEOF(.data));

LONG(LOADADDR(.data_CCMRAM));
LONG(ADDR(.data_CCMRAM));
LONG(ADDR(.data_CCMRAM)+SIZEOF(.data_CCMRAM));

__data_regions_array_end = .;

__bss_regions_array_start = .;

LONG(ADDR(.bss));
LONG(ADDR(.bss)+SIZEOF(.bss));

LONG(ADDR(.bss_CCMRAM));
LONG(ADDR(.bss_CCMRAM)+SIZEOF(.bss_CCMRAM));

__bss_regions_array_end = .;

/* End of memory regions initialisation arrays. */

/*
* These are the old initialisation sections, intended to contain
* naked code, with the prologue/epilogue added by crt0.o/crti.o
* when linking with startup files. The standalone startup
* currently does not run these, better use the init arrays
*/
KEEP(*(.init))

```

```

KEEP(*(.fini))

. = ALIGN(4);

/*
 * The preinit code, i.e. an array of pointers to initialis
 * functions to be performed before constructors.
 */
PROVIDE_HIDDEN (__preinit_array_start = .);

/*
 * Used to run the SystemInit() before anything else.
 */
KEEP(*(.preinit_array_sysinit .preinit_array_sysinit.*))

/*
 * Used for other platform inits.
 */
KEEP(*(.preinit_array_platform .preinit_array_platform.*))

/*
 * The application inits. If you need to enforce some order
 * execution, create new sections, as before.
 */
KEEP(*(.preinit_array .preinit_array.*))

PROVIDE_HIDDEN (__preinit_array_end = .);

. = ALIGN(4);

/*
 * The init code, i.e. an array of pointers to static const
 */
PROVIDE_HIDDEN (__init_array_start = .);
KEEP(* (SORT(.init_array.*)))
KEEP(*(.init_array))
PROVIDE_HIDDEN (__init_array_end = .);

. = ALIGN(4);

/*
 * The fini code, i.e. an array of pointers to static destr
 */
PROVIDE_HIDDEN (__fini_array_start = .);

```

```

        KEEP(*(SORT(.fini_array.*)))
        KEEP(*( .fini_array))
        PROVIDE_HIDDEN (__fini_array_end = .);

} >FLASH

/*
 * For some STRx devices, the beginning of the startup code
 * is stored in the .flashtext section, which goes to FLASH.
 */
.flashtext : ALIGN(4)
{
    *(.flashtext .flashtext.*)    /* Startup code */
} >FLASH

/*
 * The program code is stored in the .text section,
 * which goes to FLASH.
 */
.text : ALIGN(4)
{
    *(.text .text.*)              /* all remaining code */

    /* read-only data (constants) */
    *(.rodata .rodata.* .constdata .constdata.*)

    *(vtable)                     /* C++ virtual tables */

    KEEP(*( .eh_frame*))

    /*
     * Stub sections generated by the linker, to glue together
     * ARM and Thumb code. .glue_7 is used for ARM code calling
     * Thumb code, and .glue_7t is used for Thumb code calling
     * ARM code. Apparently always generated by the linker, for
     * architectures, so better leave them here.
     */
    *(.glue_7)
    *(.glue_7t)

} >FLASH

/* ARM magic sections */

```

```

.ARM.extab : ALIGN(4)
{
  *(.ARM.extab* .gnu.linkonce.armextab.*)
} > FLASH

. = ALIGN(4);
__exidx_start = .;
.ARM.exidx : ALIGN(4)
{
  *(.ARM.exidx* .gnu.linkonce.armexidx.*)
} > FLASH
__exidx_end = .;

. = ALIGN(4);
_etext = .;
__etext = .;

/* MEMORY_ARRAY */
/*
.R0arraySection :
{
  *(.R0arraySection .R0arraySection.*)
} >MEMORY_ARRAY
*/

/*
 * The secondary initialised data section.
 */
.data_CCMRAM : ALIGN(4)
{
  FILL(0xFF)
  *(.data.CCMRAM .data.CCMRAM.*)
  . = ALIGN(4) ;
} > CCMRAM AT>FLASH

/*
 * This address is used by the startup code to
 * initialise the .data section.
 */
_sidata = LOADADDR(.data);

/*
 * The initialised data section.
 * The program executes knowing that the data is in the RAM

```

```

* but the loader puts the initial values in the FLASH (inidata
* It is one task of the startup to copy the initial values from
* FLASH to RAM.
*/
.data : ALIGN(4)
{
    FILL(0xFF)
    /* This is used by the startup code to initialise the .data
    _sdata = . ;          /* STM specific definition */
    __data_start__ = . ;
    *(.data_begin .data_begin.*)

    *(.data .data.*)

    *(.data_end .data_end.*)
    . = ALIGN(4);

    /* This is used by the startup code to initialise the .data
    _edata = . ;          /* STM specific definition */
    __data_end__ = . ;

} >RAM AT>FLASH


/*
* The uninitialised data sections. NOLOAD is used to avoid
* the "section `.bss' type changed to PROGBITS" warning
*/

/* The secondary uninitialised data section. */
.bss_CCMRAM (NOLOAD) : ALIGN(4)
{
    *(.bss.CCMRAM .bss.CCMRAM.*)
} > CCMRAM


/* The primary uninitialised data section. */
.bss (NOLOAD) : ALIGN(4)
{
    __bss_start__ = .;          /* standard newlib definition */
    _sbss = .;                  /* STM specific definition */
    *(.bss_begin .bss_begin.*)

    *(.bss .bss.*)
    *(COMMON)

```

```

        *(.bss_end .bss_end.*)
        . = ALIGN(4);
        __bss_end__ = .;           /* standard newlib definition */
        _ebss = . ;                /* STM specific definition */
    } >RAM

    .noinit_CCMRAM (NOLOAD) : ALIGN(4)
    {
        *(.noinit.CCMRAM .noinit.CCMRAM.*)
    } > CCMRAM

    .noinit (NOLOAD) : ALIGN(4)
    {
        _noinit = .;

        *(.noinit .noinit.*)

        . = ALIGN(4) ;
        _end_noinit = .;
    } > RAM

    /* Mandatory to be word aligned, _sbrk assumes this */
    PROVIDE ( end = _end_noinit ); /* was _ebss */
    PROVIDE ( _end = _end_noinit );
    PROVIDE ( __end = _end_noinit );
    PROVIDE ( __end__ = _end_noinit );

    /*
     * Used for validation only, do not allocate anything here!
     *
     * This is just to check that there is enough RAM left for the
     * stack. It should generate an error if it's full.
     */
    ._check_stack : ALIGN(4)
    {
        . = . + _Minimum_Stack_Size ;
    } >RAM

    /*
     * The FLASH Bank1.
     * The C or assembly source must explicitly place the code
     * or data there using the "section" attribute.
     */

```

```

.b1text : ALIGN(4)
{
    *(.b1text)                /* remaining code */
    *(.b1rodata)              /* read-only data (constants)
    *(.b1rodata.*)
} >FLASHB1

/*
 * The EXTMEM.
 * The C or assembly source must explicitly place the code or c
 * using the "section" attribute.
 */

/* EXTMEM Bank0 */
.eb0text : ALIGN(4)
{
    *(.eb0text)                /* remaining code */
    *(.eb0rodata)              /* read-only data (constants)
    *(.eb0rodata.*)
} >EXTMEMB0

/* EXTMEM Bank1 */
.eb1text : ALIGN(4)
{
    *(.eb1text)                /* remaining code */
    *(.eb1rodata)              /* read-only data (constants)
    *(.eb1rodata.*)
} >EXTMEMB1

/* EXTMEM Bank2 */
.eb2text : ALIGN(4)
{
    *(.eb2text)                /* remaining code */
    *(.eb2rodata)              /* read-only data (constants)
    *(.eb2rodata.*)
} >EXTMEMB2

/* EXTMEM Bank0 */
.eb3text : ALIGN(4)
{
    *(.eb3text)                /* remaining code */
    *(.eb3rodata)              /* read-only data (constants)
    *(.eb3rodata.*)
} >EXTMEMB3

```

```

/* After that there are only debugging sections. */

/* This can remove the debugging information from the standard
/*
DISCARD :
{
    libc.a ( * )
    libm.a ( * )
    libgcc.a ( * )
}
*/

/* Stabs debugging sections. */
.stab          0 : { *(.stab) }
.stabstr        0 : { *(.stabstr) }
.stab.excl      0 : { *(.stab.excl) }
.stab.exclstr   0 : { *(.stab.exclstr) }
.stab.index     0 : { *(.stab.index) }
.stab.indexstr  0 : { *(.stab.indexstr) }
.comment        0 : { *(.comment) }
/*
 * DWARF debug sections.
 * Symbols in the DWARF debugging sections are relative to the
 * of the section so we begin them at 0.
 */
/* DWARF 1 */
.debug          0 : { *(.debug) }
.line           0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo  0 : { *(.debug_srcinfo) }
.debug_sfnames  0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges  0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info     0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev   0 : { *(.debug_abbrev) }
.debug_line     0 : { *(.debug_line) }
.debug_frame    0 : { *(.debug_frame) }
.debug_str      0 : { *(.debug_str) }
.debug_loc      0 : { *(.debug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo) }

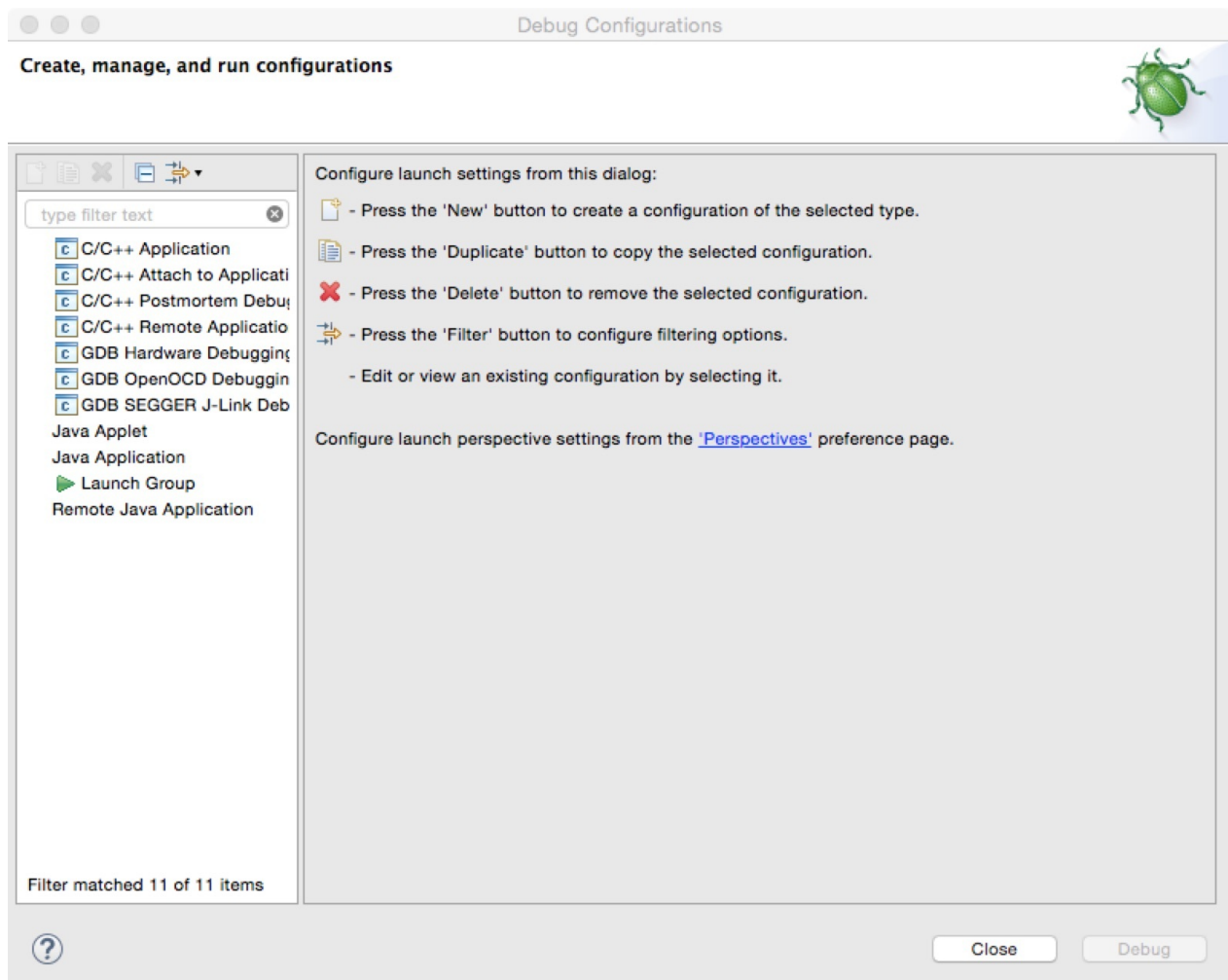
```



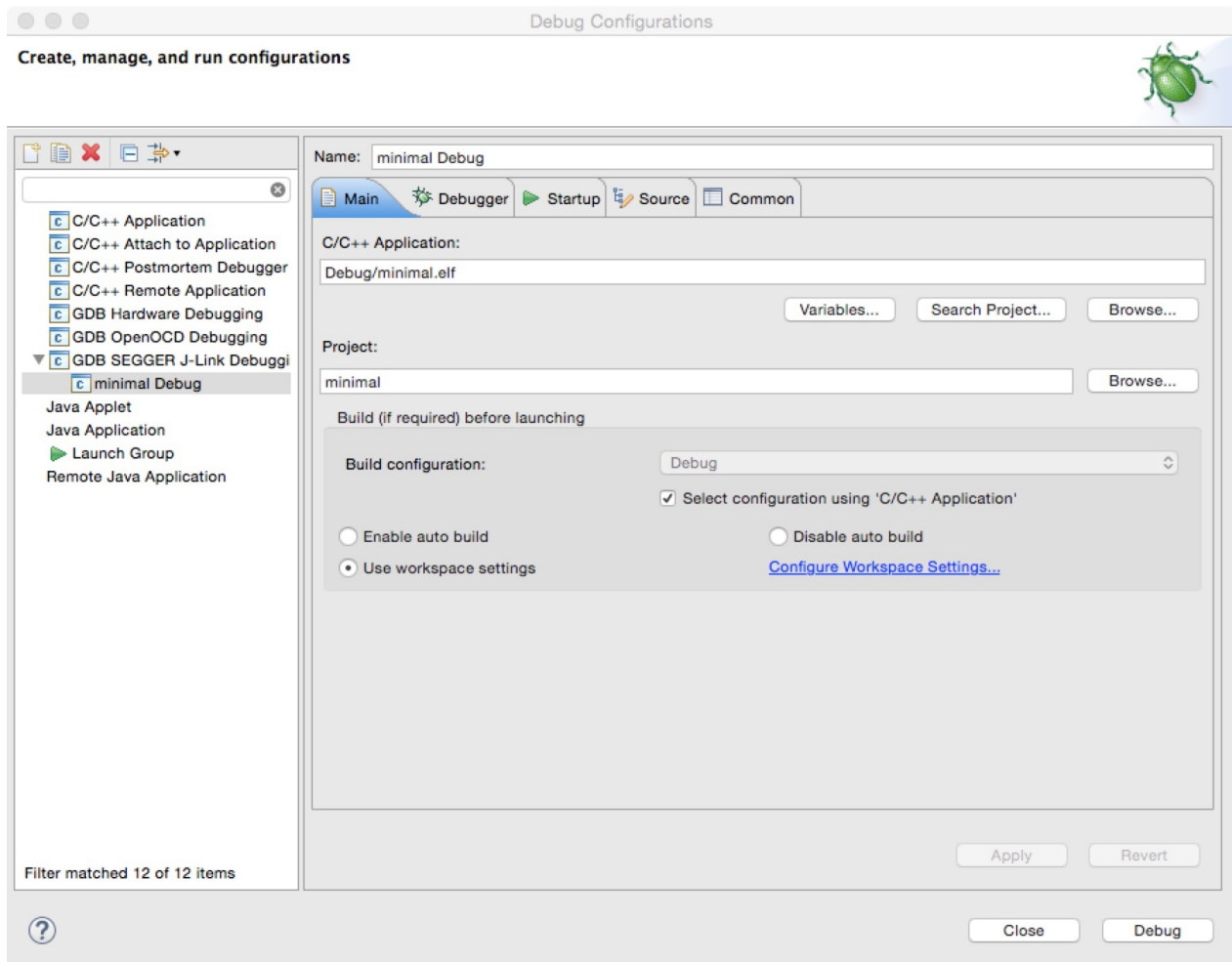
```
/* SGI/MIPS DWARF 2 extensions */  
.debug_weaknames 0 : { *(.debug_weaknames) }  
.debug_funcnames 0 : { *(.debug_funcnames) }  
.debug_typenames 0 : { *(.debug_typenames) }  
.debug_varnames 0 : { *(.debug_varnames) }  
}
```

Running the Project

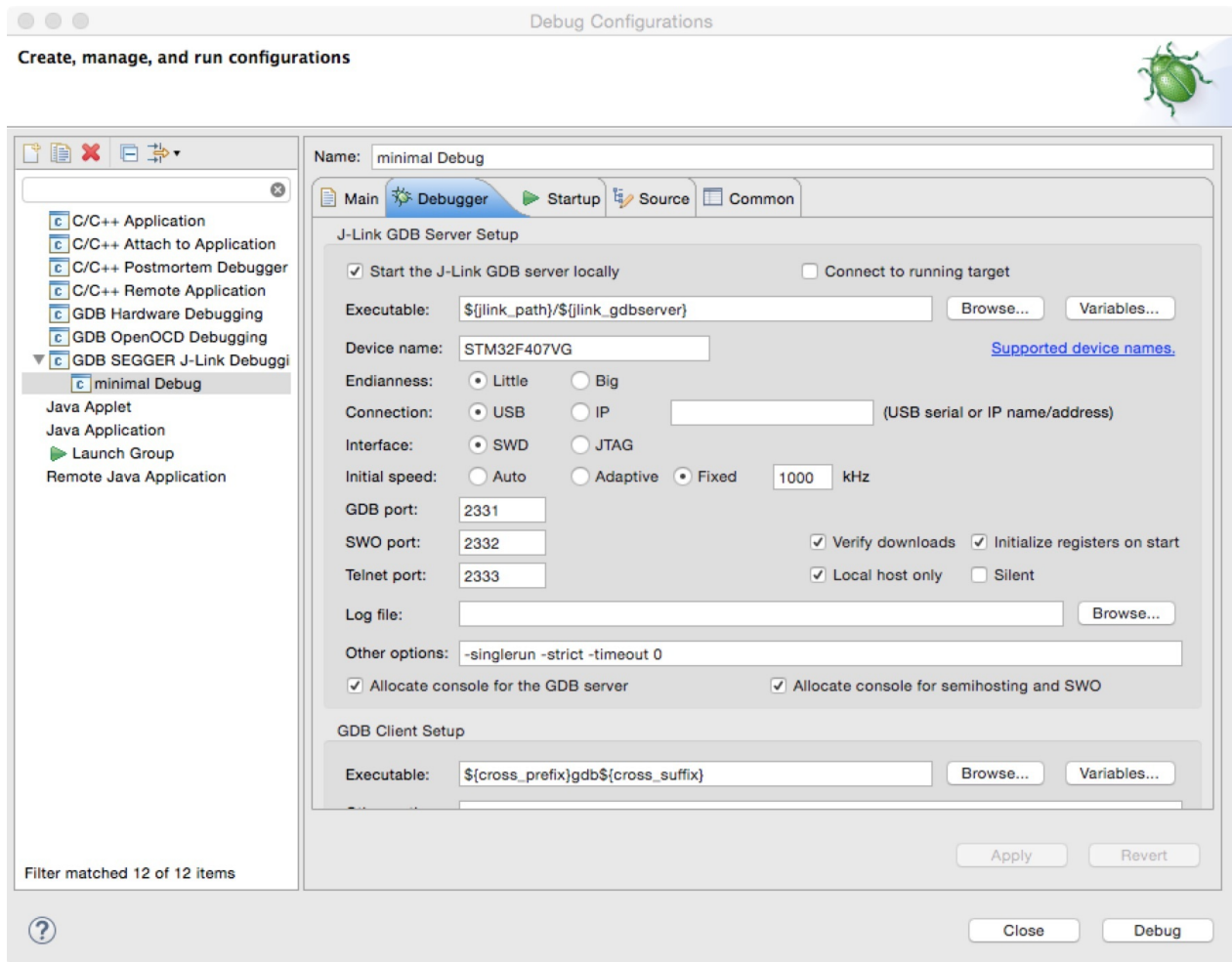
Run -> Debug Configurations...



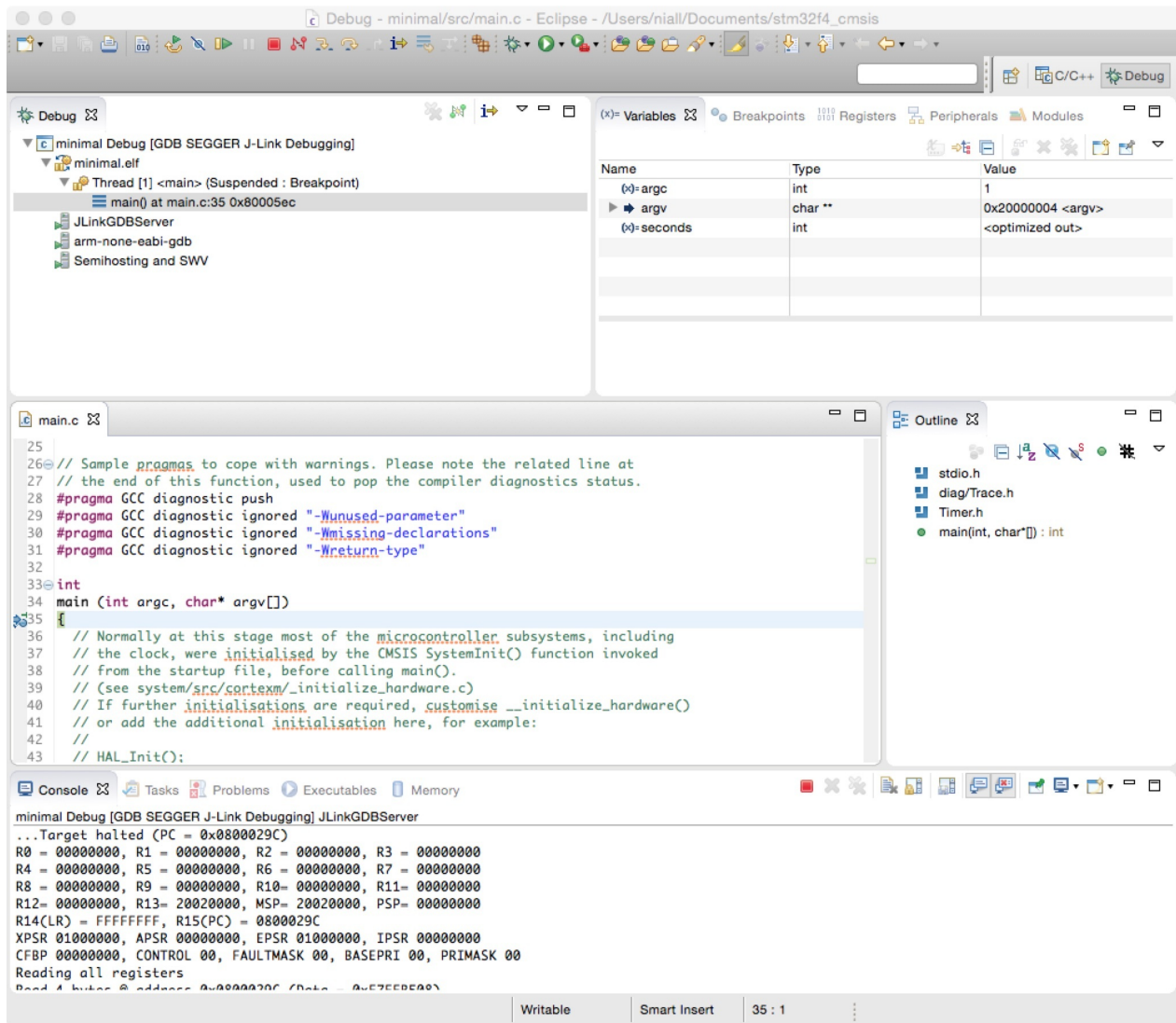
Create a New GDB SEGGER J-Link Debugging Configuration



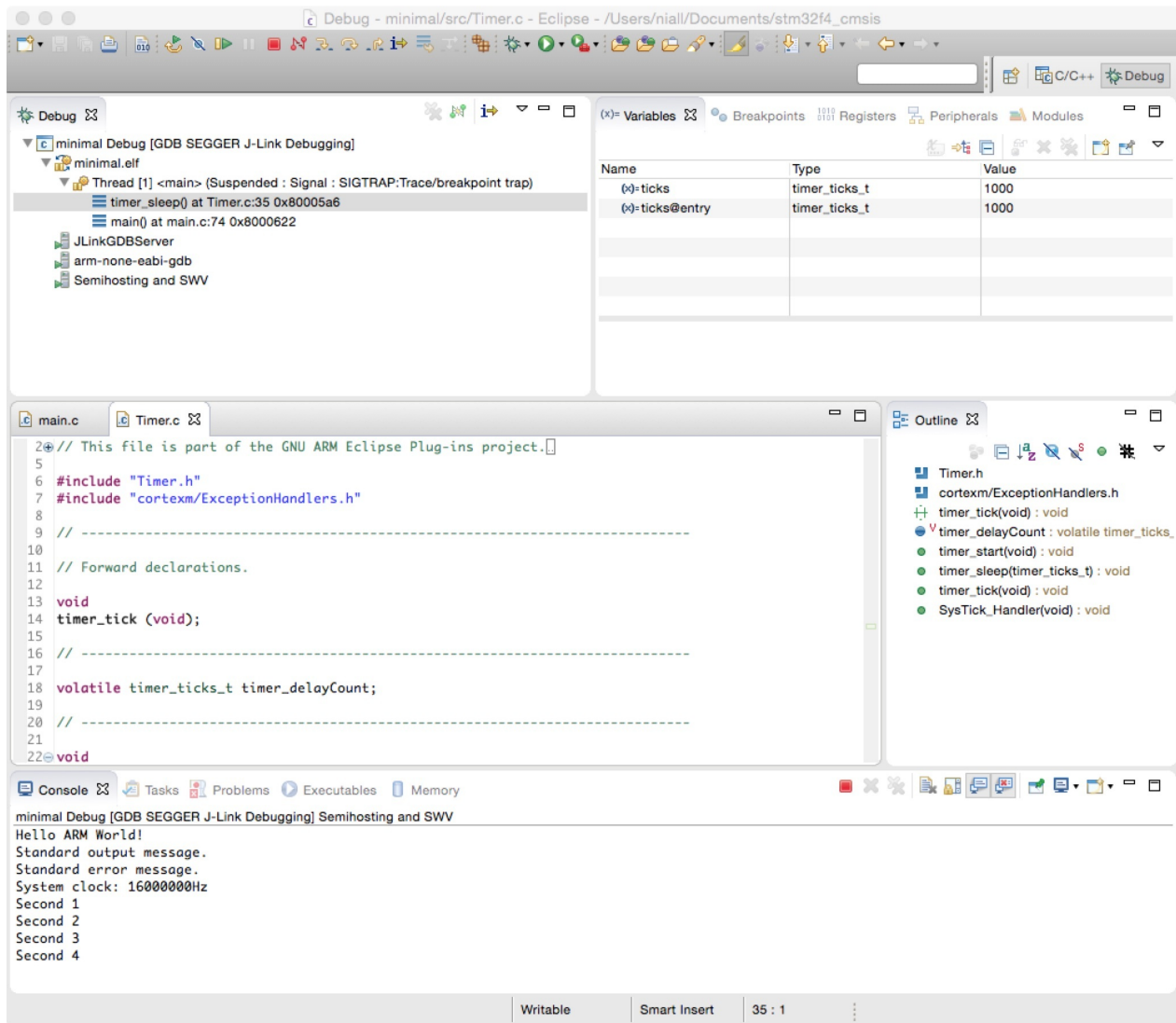
Ensure the *Device Name* is set to
STM32F407VG



Debug the project



Run the project



Power Up Sequence

IVT entry at `0x0000 0000` is the initial SP

IVT entry at `0x0000 0004` is the initial PC

Both are setup in `<project>/src/cmsis/vectors_stm32f4xx.c`

```
__attribute__((section(".isr_vector"),used))
pHandler __isr_vectors[] =
{
    // Core Level - CM4
    (pHandler) &_estack,           // The initial stack
    Reset_Handler,               // The reset handler
    ...
}
```

The Main Flash memory is automatically remapped at `0x0000 0000` as part of the boot process

`__estack`

`__estack` configured in `<project>/ldscripts/sections.ld`

```
/*
 * The '__stack' definition is required by crt0, do not remove it.
 */
__stack = ORIGIN(RAM) + LENGTH(RAM);
__estack = __stack;    /* STM specific definition */
```

The symbol `RAM` is configured in `<project>/ldscripts/mem.ld`

```
MEMORY
{
  RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
  FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
```

The IVT is placed in `FLASH` (configured as shown above) via the `mem.ld` file

```
/* Sections Definitions */

SECTIONS
{
  /*
   * For Cortex-M devices, the beginning of the startup code is s
   * the .isr_vector section, which goes to FLASH.
   */
  .isr_vector : ALIGN(4)
  {
    FILL(0xFF)

    __vectors_start__ = ABSOLUTE(.) ;
    KEEP(*(.isr_vector))    /* Interrupt vectors */

    KEEP(*(.cfmconfig))     /* Freescale configuration word

  /*
```



```

    * This section is here for convenience, to store the
    * startup code at the beginning of the flash area, hoping
    * this will increase the readability of the listing.
    */
    *(.after_vectors .after_vectors.*)    /* Startup code and I

```

```

} >FLASH

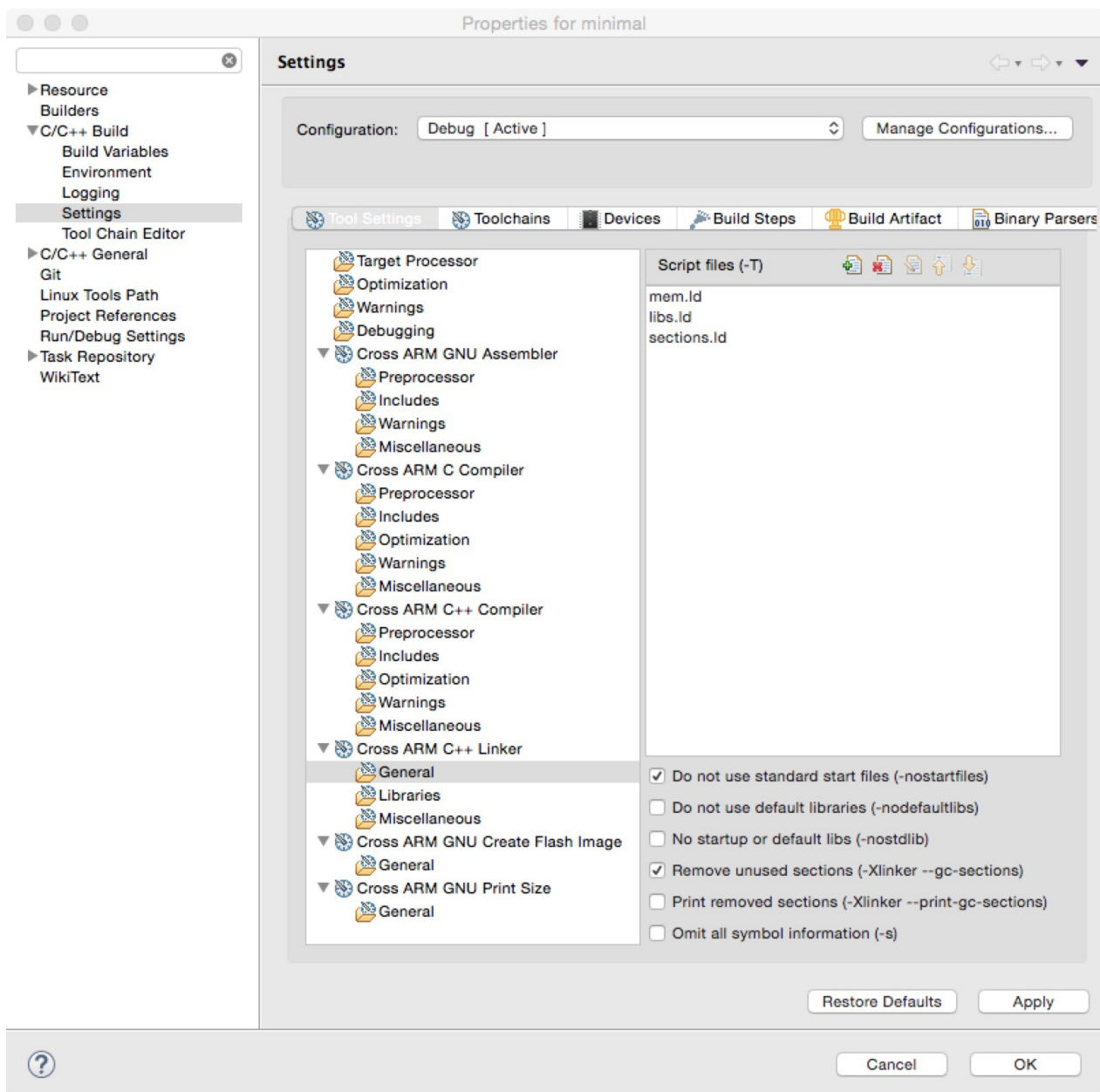
```

The linker files (`sections.ld` , `mem.ld`) to reference are configured (in eclipse) at:

```

Project Properties
  C/C++ Build
    Settings
      Tool Settings
        Cross ARM C++ Linker
          General
            Script files

```



Reset_Handler

The function `Reset_Handler` is defined in

`<project>/system/src/cortexm/exception_handlers.c`

The actual code included depends on whether the project is being built for `DEBUG` or not.

The preprocessor directive `DEBUG` is set in

```
Project Properties
  C/C++ Build
    Settings
      Tool Settings
        Cross ARM C Compiler
          Preprocessor
            Defined symbols
```

Assuming `DEBUG` mode, the definition of `Reset_Handler` is:

```
// The DEBUG version is not naked, to allow breakpoints at Reset_Handler
void __attribute__((section(".after_vectors"),noreturn))
Reset_Handler (void)
{
    _start ();
}
```

`_start` is found in file `<project>/system/newlib/_startup.c`

```
void __attribute__((section(".after_vectors"),noreturn))
_start (void)
{
```

The function `_start` does the following:

- calls function `__initialize_hardware_early ()`;
 - Initialise hardware right after reset, to switch clock to higher frequency

and have the rest of the initialisations run faster.

- This is mandatory on platforms like Kinetis, which start with the watch dog enabled and require an early sequence to disable it.
- Copies the data sections from flash to SRAM
- Zero fills all bss segments
- calls function `__initialize_hardware ()`;
 - secondary hardware initialisation
- Get the argc/argv (useful in semihosting configurations).
 - function `__initialize_args (&argc, &argv)`;
- Calls the standard library initialisation
 - mandatory for C++ to execute the constructors for the static objects
 - function `__run_init_array ()`;
- Call the main entry point, and save the exit code.
 - `int code = main (argc, argv)`;

If the `main` function returns, then it:

- Runs any C++ static destructors.
 - function `__run_fini_array ()`;
- call `_exit (code)`; using the `main` return code
 - `_exit()` should never return

__initialize_hardware_early

Function defined in

file: <project>/system/src/cortexm/_initialize_hardware.c

```
void
__attribute__((weak))
__initialize_hardware_early(void)
{
    // Call the CMSIS system initialisation routine.
    SystemInit();
}
```

__initialize_hardware

The default function is defined in:

<project>/system/src/cortexm/_initialize_hardware.c

```
void
__attribute__((weak))
__initialize_hardware(void)
{
    // Call the CMSIS system clock routine to store the clock frequency
    // in the SystemCoreClock global RAM location.
    SystemCoreClockUpdate();
}
```

FLASH Configuration

the default project doesn't initialise the flash access controller. However projects using the STM HAL code include this initialisation.

Flash access control register (`FLASH_ACR`)

- `FLASH->ACR |= FLASH_ACR_PRFTEN`
 - `#define FLASH_ACR_PRFTEN ((uint32_t)0x00000100)`
 - Bit 8 PRFTEN : Prefetch enable
- `FLASH->ACR |= FLASH_ACR_ICEN`
 - `#define FLASH_ACR_ICEN ((uint32_t)0x00000200)`
 - Bit 9 ICEN : Instruction cache enable
- `FLASH->ACR |= FLASH_ACR_DCEN`
 - `#define FLASH_ACR_DCEN ((uint32_t)0x00000400)`
 - Bit 10 DCEN : Data cache enable

This could be added to the `__initialize_hardware` function

SysTick

The default project initialises SysTick in the file `Timer.c`

```
//
// This file is part of the GNU ARM Eclipse Plug-ins project.
// Copyright (c) 2014 Liviu Ionescu.
//

#include "Timer.h"
#include "cortexm/ExceptionHandlers.h"

// -----

// Forward declarations.

void
timer_tick (void);

// -----

volatile timer_ticks_t timer_delayCount;

// -----

void
timer_start (void)
{
    // Use SysTick as reference for the delay loops.
    SysTick_Config (SystemCoreClock / TIMER_FREQUENCY_HZ);
}

void
timer_sleep (timer_ticks_t ticks)
{
    timer_delayCount = ticks;

    // Busy wait until the SysTick decrements the counter to zero.
    while (timer_delayCount != 0u)
        ;
}
```



```
void
timer_tick (void)
{
    // Decrement to zero the counter used by the delay routine.
    if (timer_delayCount != 0u)
    {
        --timer_delayCount;
    }
}
```

```
// ----- SysTick_Handler() -----
```

```
void
SysTick_Handler (void)
{
    timer_tick ();
}
```

```
// -----
```

Standard and Debug IO

Standard Output

the default project is configured through the preprocessor defines. If the `TRACE` macro is set, both `STDOUT` and `STDERR` are routed to the trace device

The project uses `newlib` and any call to `STDOUT/STDERR` will result in a function `_write` being called.

File: `src/_write.c`

```
ssize_t
_write (int fd __attribute__((unused)), const char* buf __attribute__((unused)),
        size_t nbyte __attribute__((unused)))
{
    #if defined(TRACE)
        // STDOUT and STDERR are routed to the trace device
        if (fd == 1 || fd == 2)
        {
            return trace_write (buf, nbyte);
        }
    #endif // TRACE

    errno = ENOSYS;
    return -1;
}
```

The implementation of `trace_write` can be found in file

- `system/src/diag/trace_impl.c`

Trace Output

The trace functions:

```
void
```

```
trace_initialize(void);

// Implementation dependent
ssize_t
trace_write(const char* buf, size_t nbyte);

// ----- Portable -----

int
trace_printf(const char* format, ...);

int
trace_puts(const char *s);

int
trace_putchar(int c);

void
trace_dump_args(int argc, char* argv[]);
```

also call on the `trace_write` function

General Purpose Input/Output

General Purpose Output

General Purpose Input

Setting Up GPIO for Output

This code uses CMSIS directives from `stm32f4xx.h`

Our 4 LEDs are connected to pins 8..11 on Port D

Steps based on ST Cube Initialization:

- Include header `#include "stm32f4xx.h"`
- Enable GPIO D IO Port Clock
 - RCC AHB1 peripheral clock enable register (`RCC_AHB1ENR`)
 - Bit3 (`GPIOEN`) : IO port D clock enable
- Mode = `GPIO_MODE_OUTPUT_PP`
 - GPIO port mode register (`GPIO_MODER`)
 - Reset values: `0x0000 0000` for Port D
 - `01` : General purpose output mode
 - `GPIO_MODER |= (0x01 << (pin * 2));`
- Pull = `GPIO_NOPULL`
 - GPIO port pull-up/pull-down register (`GPIO_PUPDR`)
 - Reset values: `0x0000 0000` for Port D
 - `00` : No pull-up, pull-down
 - *Does not need setting as reset value*
- Speed = `GPIO_SPEED_LOW`
 - GPIO port output speed register (`GPIO_OSPEEDR`)
 - Reset values: `0x0000 0000` for Port D
 - `00` : Low speed
 - *Does not need setting as reset value*

Mode Options

```
#define GPIO_MODE_OUTPUT_PP ((uint32_t)0x00000001) /*!< Output
```

Pull Options

```
#define GPIO_NOPULL          ((uint32_t)0x00000000)  /*!< No Pull-up or Pull-down resistor is connected
```

Speed Options

```
#define GPIO_SPEED_LOW       ((uint32_t)0x00000000)  /*!< Low speed mode
```

Driving the Output

Using the GPIO port output data register (`GPIOx_ODR`)

```
GPIO->ODR |= (1 << 8);  // Set bit 8 output high

GPIO->ODR &= ~(1 << 8); // Set bit 8 output low
```

Using the GPIO port bit set/reset register (`GPIOx_BSRR`)

The purpose of the `GPIOx_BSRR` register is to allow atomic read/modify accesses to any of the GPIO registers.

To each bit in `GPIOx_ODR`, correspond two control bits in `GPIOx_BSRR`: `BSRR(i)` and `BSRR(i+SIZE)`. When written to 1, bit `BSRR(i)` sets the corresponding `ODR(i)` bit. When written to 1, bit `BSRR(i+SIZE)` resets the `ODR(i)` corresponding bit.

Writing any bit to 0 in `GPIOx_BSRR` does not have any effect on the corresponding bit in `GPIOx_ODR`. If there is an attempt to both set and reset a bit in `GPIOx_BSRR`, the set action takes priority.

```
GPIO->BSRR = (1 << 8);  // Set bit 8 output high
```

```
GPIO->BSRRH = (1 << 8); // Set bit 8 output low
```

Setting Up GPIO for Input

By default, on power-up reset, all the GPIO ports are set for input, no specific initialisation code is required to configure them.

To ensure the inputs are set, you may want to reset the Port's mode back to the power on settings.

Steps based on ST Cube Initialization:

- Include header `#include "stm32f4xx.h"`
- Enable GPIO D IO Port Clock
 - RCC AHB1 peripheral clock enable register (`RCC_AHB1ENR`)
 - Bit3 (`GPIOEN`) : IO port D clock enable
- Mode = `GPIO_MODE_INPUT`
 - GPIO port mode register (`GPIO_MODER`)
 - Reset values: `0x0000 0000` for Port D
 - `00` : General purpose input mode
 - *Does not need setting as reset value*
- Pull = `GPIO_NOPULL`
 - GPIO port pull-up/pull-down register (`GPIO_PUPDR`)
 - Reset values: `0x0000 0000` for Port D
 - `00` : No pull-up, pull-down
 - *Does not need setting as reset value*

Mode Options

```
#define GPIO_MODE_INPUT      ((uint32_t)0x00000000)  /*!< Input
```

Pull Options

```
#define GPIO_NOPULL         ((uint32_t)0x00000000)  /*!< No Pull-u
```


Reading the Input

Using the GPIO port input data register (`GPIOx_IDR`)

These bits in `GPIOx_IDR` are read-only and must be accessed in word mode only. They contain the input value of the corresponding I/O port.

```
uint32_t inputValue = GPIOD->IDR;
```

Serial IO

UART3 Initialisation

The initialisation of UART3 takes the following steps:

1. Peripheral clock enable for UART3
2. Enable GPIO B IO Port Clock
3. Initialise the UART for out setup (115k, 8, 1, N)
4. Configure GPIO pins PB10 & PB11 for the alternative functions of UART3 Tx and Rx

Include header `#include "stm32f4xx.h"`

Peripheral clock enable for UART3

- RCC APB1 peripheral clock enable register (`RCC->APB1ENR`)
- Bit18 (`USART3EN`) : USART3 clock enable

Enable GPIO B IO Port Clock

- RCC APB1 peripheral clock enable register (`RCC->AHB1ENR`)
- Bit1 (`GPIOBEN`) : IO port B clock enable

Initialise the UART for out setup (115k, 8, 1, N)

Configure GPIO pins PB10 & PB11 for the alternative functions of UART3 Tx and Rx

USART Init

Disable the USART

Control register 1 (USART3->CR1)

- UE : USART enable [Bit 13]
 - When this bit is cleared, the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption.
 - This bit is set and cleared by software.
 - 0: USART prescaler and outputs disabled
 - 1: USART enabled

Set the UART Communication parameters

USART CR2 Configuration (USART3->CR2)

On reset, the USART defaults to 1 Stop Bit , so nothing actually needs configuring.

However, if you were to be safe, then the STOP bits should be reset back to 00 :

1. read CR2
 - i. `tmpreg = USART3->CR2;`
2. Clear STOP[13:12] bits
 - i. `tmpreg &= ~((uint32_t)USART_CR2_STOP);`
 - ii. `#define USART_CR2_STOP ((uint32_t)0x3000) /*!<STOP[1:0] bits
(STOP bits) */`
3. Write to USART CR2
 - i. `USART3->CR2 = (uint32_t)tmpreg;`

30.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

CR2 Default Settings

- STOP [13:12] 00 : 1 Stop bit **nothing needs setting**

USART CR1 Configuration

On reset, the USART defaults to 8 data bits and parity disabled. However the transmit and receive functions need enabling:

1. Clear M, PCE, PS, TE and RE bits

- i. `uint32_t tmpreg = USRT3->CR1;`
- ii. `tmpreg &= ~(uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1_TE | USART_CR1_RE | USART_CR1_OVER8));`

2. Configure the UART Word Length, Parity and mode:

- i. Set TE and RE bits according to Init Mode value
- ii. `tmpreg |= (uint32_t)(USART_CR1_RE | USART_CR1_TE);`

3. Write to USART CR1

- i. `USRT3->CR1 = (uint32_t)tmpreg;`

```
#define USART_CR1_RE ((uint32_t)0x0004)
#define USART_CR1_TE ((uint32_t)0x0008)

#define UART_MODE_RX ((uint32_t)USART_CR1_RE
#define UART_MODE_TX ((uint32_t)USART_CR1_TE
#define UART_MODE_TX_RX ((uint32_t)(USART_CR1_RE | USART_CR1_TE)
```

30.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

CR1 Default Settings

- M : Wordlength [12]
 - 0 : 1 Start bit, 8 Data bits, n Stop bit
 - *nothing needs setting*
- PCE : Paritycontrolenable [10]
 - 0: Parity control disabled
 - *nothing needs setting*
- PS : Parity selection [9]
 - *nothing needs setting as PCE is 0*
- TE : Transmitter enable [3]
 - 1: Transmitter is enabled
 - **needs setting**
- RE : Receiver enable [2]
 - 1: Receiver is enabled
 - **needs setting**
- OVER8 : Oversampling mode
 - 0: oversampling by 16
 - *nothing needs setting*

USART CR3 Configuration (USART3->CR3)

Hardware flow control is disabled on reset, therefore does not need configuring.

CR3 Default Settings

- CTSE : CTS enable [9]
 - 0: CTS hardware flow control disabled
 - *nothing needs setting*

- RTSE : RTSenable [8]
 - 0: RTS hardware flow control disabled
 - *nothing needs setting*

USART Baud Rate Register (USARTn->BRR)

Baud Rate Register calculation:

- $BRR = clk / (2 \times 8 \times \text{baud})$
- $BRR = 16\text{MHz} / (16 \times 115200)$
- $BRR = 8.6875$
- $BRR = 0x8B = 8 + (0xB / 16)$ in [12:4] format
- $BRR = 139 \quad 0x8B \quad 0b1000 \quad 1011$

Bits15:4 DIV_Mantissa[11:0]:mantissa of USARTDIV

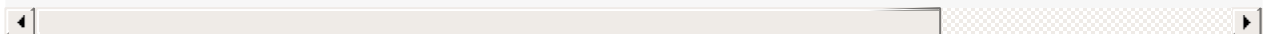
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits3:0 DIV_Fraction[3:0]:fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV)

When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

```
USART3->BRR = __UART_BRR_SAMPLING16(HAL_RCC_GetPCLK1Freq(), huart
```



In asynchronous mode

the following bits must be kept cleared (*They are clear on reset*):

- LINEN and CLKEN bits in the USART_CR2 register,
- SCEN, HDSEL and IREN bits in the USART_CR3 register.

```
USART3->CR2 &= ~(USART_CR2_LINEN | USART_CR2_CLKEN);
```

```
USART3->CR3 &= ~(USART_CR3_SCEN | USART_CR3_HDSEL | USART_CR3_IREN)
```

Enable the UART (USART3->CR1)

- UE: USART enable [Bit 13]
 - 1: USART enabled

CMSIS

```
// file      stm32f407xx.h

/**
 * @brief Universal Synchronous Asynchronous Receiver Transmitter
 */

typedef struct
{
    __IO uint32_t SR;          /*!< USART Status register,
    __IO uint32_t DR;          /*!< USART Data register,
    __IO uint32_t BRR;         /*!< USART Baud rate register,
    __IO uint32_t CR1;         /*!< USART Control register 1,
    __IO uint32_t CR2;         /*!< USART Control register 2,
    __IO uint32_t CR3;         /*!< USART Control register 3,
    __IO uint32_t GTPR;        /*!< USART Guard time and prescaler reg
} USART_TypeDef;
```

GPIO B Alternative Function Setup

- USART3 GPIO Configuration
 - PB10 -> USART3_TX
 - PB11 -> USART3_RX

```
#define GPIO_PIN_10      ((uint16_t)0x0400) /* Pin 10 selected
#define GPIO_PIN_11      ((uint16_t)0x0800) /* Pin 11 selected
```

- Configure **Alternate function** mapped with the current IO (GPIOx->AFRL/H)
- Configure **IO Direction mode** (Alternate) (GPIOx->MODER)
- Configure the **IO Speed** (GPIOx->OSPEEDR)
- Configure the **IO Output Type** (GPIOx->OTYPER)
- Activate the **Pull-up or Pull down** resistor for the current IO (GPIOx->PUPDR)

```
Pin = GPIO_PIN_10|GPIO_PIN_11;
Mode = GPIO_MODE_AF_PP;
Pull = GPIO_PULLUP;
Speed = GPIO_SPEED_HIGH;
Alternate = GPIO_AF7_USART3;
```

GPIO Alternate function registers (GPIOx->AFR[n] n=0..1)

There are 4 bits per pin. The *low* register configures bits 0..7, and the *high* register configures bits 8..15.

- GPIO alternate function low register (GPIOx->AFR[0])
 - Bits 31:0 AFR_{Ly} : Alternate function selection for port x bit y (y=0..7)
- GPIO alternate function high register (GPIOx->AFR[1])
 - Bits 31:0 AFR_H**y** : Alternate function selection for port x bit y (**y=8..15**)
- GPIO AF7 (USART1..3)


```
#define GPIO_AF7_USART3 ((uint8_t)0x07) /* USART3 Alternate Fu
```

GPIO *mode* register (GPIOx->MODER)

- GPIO B Reset value: 0x0000 0280
- Bits 2y:2y+1 MODERy [1:0] : Port x configuration bits (y=0..15)
 - 0b10 : Alternate function mode

```
#define GPIO_MODE_AF_PP ((uint32_t)0x00000002) /*!< Alternate
```

```
GPIOB->MODER |= (GPIO_MODE_AF_PP << (pin * 2));
```

GPIO *output speed* register (GPIOx->OSPEEDR)

- GPIO B Reset value: 0x0000 00C0
- Bits 2y:2y+1 OSPEEDRy [1:0] : Port x configuration bits (y=0..15)
 - 0b11 : Very high speed

```
#define GPIO_SPEED_HIGH ((uint32_t)0x00000003) /*!< High speed
```

```
GPIOB->OSPEEDR |= (GPIO_SPEED_HIGH << (pin * 2));
```

GPIO *output type* register (GPIOx->OTYPER)

This register can be ignored as reset state is push-pull

- Bits 15:0 OTy : Port x configuration bits (y=0..15)
 - 0: Output push-pull (reset state)

GPIO port *pull-up/pull-down* register (GPIOx->PUPDR)

- GPIO B Reset value: 0x0000 0100
- Bits 2y:2y+1 PUPDRy[1:0] : Port x configuration bits (y=0..15)
 - 0b01 : Pull-up

```
#define GPIO_PULLUP          ((uint32_t)0x00000001) /*!< Pull-up e

GPIOB->PUPDR |= (GPIO_PULLUP << (pin * 2));
```

CMSIS

```
typedef struct
{
    __IO uint32_t MODER; /*!< GPIO port mode register,
    __IO uint32_t OTYPER; /*!< GPIO port output type register,
    __IO uint32_t OSPEEDR; /*!< GPIO port output speed register,
    __IO uint32_t PUPDR; /*!< GPIO port pull-up/pull-down register,
    __IO uint32_t IDR; /*!< GPIO port input data register,
    __IO uint32_t ODR; /*!< GPIO port output data register,
    __IO uint16_t BSRRL; /*!< GPIO port bit set/reset low register,
    __IO uint16_t BSRRH; /*!< GPIO port bit set/reset high register,
    __IO uint32_t LCKR; /*!< GPIO port configuration lock register,
    __IO uint32_t AFR[2]; /*!< GPIO alternate function registers,
} GPIO_TypeDef;
```

Interrupts

The STM32F407VG only uses 4 of the possible 8 bits for interrupt priority.

Configuring the binary point is done via the CMSIS function

`NVIC_SetPriorityGrouping`

The default appears to be (using 4 bits) 16 unique Group priorities.

The priority grouping is configured via the `NVIC` Application interrupt and reset control register (`AIRCR`).

- Bits [10:8] `PRIGROUP` : Interrupt priority grouping field

CMSIS

CMSIS provides a number of functions for NVIC control, including:

CMSIS interrupt control function	Description
<code>void NVIC_SetPriorityGrouping(uint32_t priority_grouping)</code>	Set the priority grouping
<code>void NVIC_EnableIRQ(IRQn_t IRQn)</code>	Enable IRQn
<code>void NVIC_DisableIRQ(IRQn_t IRQn)</code>	Disable IRQn
<code>uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)</code>	Return true (IRQ-Number) if IRQn is pending
<code>void NVIC_SetPendingIRQ (IRQn_t IRQn)</code>	Set IRQn pending
<code>void NVIC_ClearPendingIRQ (IRQn_t IRQn)</code>	Clear IRQn pending status
<code>uint32_t NVIC_GetActive (IRQn_t IRQn)</code>	Return the IRQ number of the active interrupt
<code>void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)</code>	Set priority for IRQn
<code>uint32_t NVIC_GetPriority (IRQn_t IRQn)</code>	Read priority of IRQn
<code>void NVIC_SystemReset (void)</code>	Reset the system

Interrupt priority level value, `PRI_N[7:4]`

PRIGROUP [2:0]	Binary point ¹	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b011	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

¹ PRI_n[7:4] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

e.g.

```
NVIC_SetPriorityGrouping(5); // [2:2]
```

will set the priority grouping with 2 bits for both the Priority and SubPriority, giving a range of 0..3 for each.

This is then used to set a particular interrupt sources priority, e.g.:

```
prioritygroup = NVIC_GetPriorityGrouping(); // will return 5
priority = NVIC_EncodePriority(prioritygroup, 1, 1 ); // Pri=1
NVIC_SetPriority(EXTI4_IRQn, priority);
```

GPIO as an external Interrupt

GP Inputs as external Interrupt (EXTI)

EXTI 1..4

Basic principle:

- Configure the Port Pin as input
- It is worth setting the GPIO speed to HIGH
- Ensure the SYSCFG clock enable
- Configure source Port for the particular EXTI
 - Each Pin is or'ed for all Ports.
 - So Pin 4 on Port A and Pin 4 on port D both map to EXTI4
 - The EXTICR registers configure the mapping
- Unmask the EXTI interrupt
- Select whether the interrupt will be generated on the rising or falling edge (or both)
- Set the priority for the EXTI
- Enable the IRQ for the EXTI

```
// make sure Port D pin 4 is configured as input
// RCC->AHB1ENR |= (1 << 3);
// set port D pin 4 as EXTI4 interrupt source
  GPIOD->OSPEEDR |= (0x03 << (2 * 4));    // high speed
  RCC->APB2ENR |= (1 << 14);                // SYSCFG clock enable
  SYSCFG->EXTICR[1] |= 0x03;                // EXTI4 set to Port D
  EXTI->IMR |= (1 << 4);                    // unmask EXTI0 interrupt
  EXTI->RTSR |= (1 << 4);                    // rising edge
// EXTI->FTSR |= (1 << 4);                    // falling edge

uint32_t prioritygroup = NVIC_GetPriorityGrouping();
// Highest user int priority (0), 1 sub-pri
uint32_t priority = NVIC_EncodePriority(prioritygroup, 0, 1);
NVIC_SetPriority(EXTI4_IRQn, priority);
NVIC_EnableIRQ(EXTI4_IRQn);
```

```

void EXTI4_IRQHandler(void)
{
    EXTI->PR |= (1 << 4);           // ack int
    // ISR code
}

```

EXTI 5..9

The interrupt sources share a common ISR

```

// set port D pin 5 as EXTI5 interrupt source (ACCEPT Key)
GPIO->OSPEEDR |= (0x03 << (2 * 5));           // high speed
// RCC->APB2ENR |= (1 << 14);           // SYSCFG clock enable
SYSCFG->EXTICR[1] |= (0x03 << 4);           // EXTI5 set to Port D
EXTI->IMR |= (1 << 5);                       // unmask EXTI0 interrupt
EXTI->RTSR |= (1 << 5);                       // rising edge
// EXTI->FTSR |= (1 << 5);                 // falling edge

// set port D pin 6 as EXTI6 interrupt source (Motor Feedback)
GPIO->OSPEEDR |= (0x03 << (2 * 6));           // high speed
// RCC->APB2ENR |= (1 << 14);           // SYSCFG clock enable
SYSCFG->EXTICR[1] |= (0x03 << 8);           // EXTI6 set to Port D
EXTI->IMR |= (1 << 6);                       // unmask EXTI0 interrupt
EXTI->RTSR |= (1 << 6);                       // rising edge
// EXTI->FTSR |= (1 << 6);                 // falling edge

prioritygroup = NVIC_GetPriorityGrouping();
priority = NVIC_EncodePriority(prioritygroup, 1, 0);
NVIC_SetPriority(EXTI9_5_IRQn, priority);
NVIC_EnableIRQ(EXTI9_5_IRQn);

```

```

void EXTI9_5_IRQHandler(void)
{
    if(EXTI->PR & (1 << 5)) {                 // if EXTI5 source
        EXTI->PR |= (1 << 5);                 // ack int
        // ISR code
    }
    if(EXTI->PR & (1 << 6)) {                 // if EXTI6 source
        EXTI->PR |= (1 << 6);                 // ack int
    }
}

```

```
        // ISR code  
    }  
}
```

FreeRTOS Integration

Basic steps:

1. copied FreeRTOS source code directory into project under
 - i. Middleware/Third_Party/FreeRTOS
 - ii. Set up include paths
2. copied FreeRTOSConfig.h into project include folder
3. Under FreeRTOS/Source/portable
 - i. excluded IAR part of directory tree
 - ii. /MemManger
 - i. **excluded all** but heap_3.c
4. Changed C/C++ Build - Settings - Tools
 - i. Float ABI -> FP instructions (hard)
 - ii. FPU Type -> fpv4-sp-d16

This should now compile and build

1. In SysTick_Handler
 - i. added call to xPortSysTickHandler();
2. in main
 - i. create tasks that use vTaskDelay(ticks);
 - ii. call vTaskStartScheduler();

FreeRTOS required extra Includes

```

"../include"
"../system/include"
"../system/include/cmsis"
"../system/include/DEVICE"
"${workspace_loc}/${ProjName}/Middlewares/Third_Party/FreeRTOS/Sour
"${workspace_loc}/${ProjName}/Middlewares/Third_Party/FreeRTOS/Sour

```


FeabhOS

Basic steps

1. Copy feabhOS source code into folder `feabhos`
2. Exclude folders `feabhos/C`
 - i. `POSIX`
 - ii. `Win32`
3. Add `feabhas` project include
4. Build and run

Includes

```
"../include"  
"../system/include"  
"../system/include/cmsis"  
"../system/include/DEVICE"  
../Middlewares/Third_Party/FreeRTOS/Source/include  
../Middlewares/Third_Party/FreeRTOS/Source/portable/GCC/ARM_CM4F  
../feabhos/C/common/inc  
../feabhos/C/FreeRTOS/inc
```

[example main.c](#)

```

#include <stdio.h>
#include <assert.h>

#include "diag/Trace.h"
#include "Timer.h"

#include "stm32f4xx.h"

#include "gpio_port_d.h"

#include "feabhOS_task.h"
#include "feabhOS_scheduler.h"

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"

typedef enum { D6 = 8, D5, D4, D3 } LEDS;

/* Task to be created. */
int D5taskCode( void * pvParameters )
{
    for( ;; ) {
        /* Task code goes here. */
        port_d_toggle_bit(D5);
        feabhOS_task_sleep(500);
    }
    return 0;
}

int D4taskCode( void * pvParameters )
{
    for( ;; ) {
        /* Task code goes here. */
        port_d_toggle_bit(D4);
        feabhOS_task_sleep(750);
    }
    return 0;
}

int
main (int argc, char* argv[])
{

```

```

    feabhOS_scheduler_init();

    feabhOS_TASK   D5Handle;
    feabhOS_TASK   D4Handle;

    feabhOS_error error;

    trace_printf("System clock: %uHz\n", SystemCoreClock);

    timer_start ();

    port_d_set_as_output(D6);
    port_d_set_as_output(D5);
    port_d_set_as_output(D4);
    port_d_set_as_output(D3);

    // Just to make rest of the WMS quite
    port_d_set_as_output(12);
    port_d_set_as_output(13);
    port_d_set_as_output(14);
    port_d_set_as_output(15);

    error = feabhOS_task_create( &D5Handle, D5taskCode, NULL, STACK
    assert( error == ERROR_OK );
    error = feabhOS_task_create( &D4Handle, D4taskCode, NULL, STACK
    assert( error == ERROR_OK );

    // Start the scheduler.
    feabhOS_scheduler_start();

    // Will only get here if there was insufficient memory to creat
    // and/or timer task.
}

#pragma GCC diagnostic pop

// -----

```