./

Learning Report – Embedded C - Hardware + Programming + Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 1.1 | 30-Sept-2020 | Anand A Warrier |  | Dr. Vivek Kaundal, Bhargav N | Initial draft report for review |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

Contents

[List of figures 4](#_Toc52375900)

[Activity 1 – Writing linker scripts 5](#_Toc52375901)

[Activity 2 – Semi Hosting technique 7](#_Toc52375902)

[Activity 3 – In class activity 8](#_Toc52375903)

[Activity 4 – Debugging techniques 10](#_Toc52375904)

[Activity 5 – Completing MCU specific header files 15](#_Toc52375905)

# List of figures

[Figure 1 - Linker script 5](#_Toc52370921)

[Figure 2 - Semi hosting technique to print "Hello World" 6](#_Toc52370922)

[Figure 3 - RCC bit enabled for SPI1 register 8](#_Toc52370923)

[Figure 4 - SPE bit of SPI control register enabled 8](#_Toc52370924)

# Activity 1 – Writing linker scripts

**Requirement:** Create one memory location in the linker script.

**Code:**

//Entry point

ENTRY(Reset\_Handler)

//Memories definition

MEMORY

{

FLASH(rx):ORIGIN =0x08000000,LENGTH =1024K

SRAM(rwx):ORIGIN =0x20000000,LENGTH =128K

}

//Sections

SECTIONS

{

.text :

{

\*(.isr\_vector)

\*(.text)

\*(.text.\*)

\*(.init)

\*(.fini)

\*(.rodata)

\*(.rodata.\*)

. = ALIGN(4);

\_etext = .;

}> FLASH

\_la\_data = LOADADDR(.data);

.data :

{

\_sdata = .;

\*(.data)

\*(.data.\*)

. = ALIGN(4);

\_edata = .;

}> SRAM AT> FLASH

.bss :

{

\_sbss = .;

\_\_bss\_start\_\_ = \_sbss;

\*(.bss)

\*(.bss.\*)

\*(COMMON)

. = ALIGN(4);

\_ebss = .;

\_\_bss\_end\_\_ = \_ebss;

. = ALIGN(4);

end = .;

\_\_end\_\_ = .;

}> SRAM

} //close\_Sections

**Screenshot:**

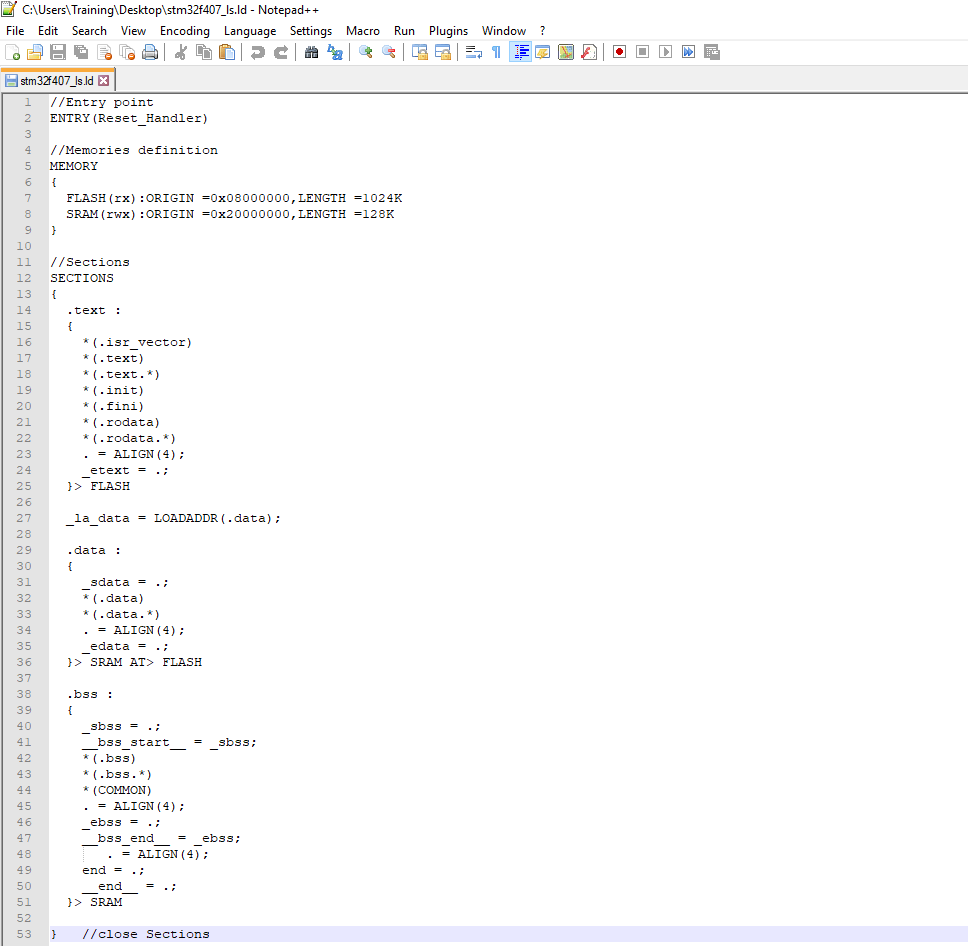


Figure 1 - Linker script

# Activity 2 – Semi Hosting technique

**Requirement:**

**Code:**

**#include** <stdio.h>

**extern** **void** **initialise\_monitor\_handles**(**void**);

**int** **main**(**void**)

{

initialise\_monitor\_handles();

**printf**("Hello World\n");

**for**(;;);

}

**Screenshot:**

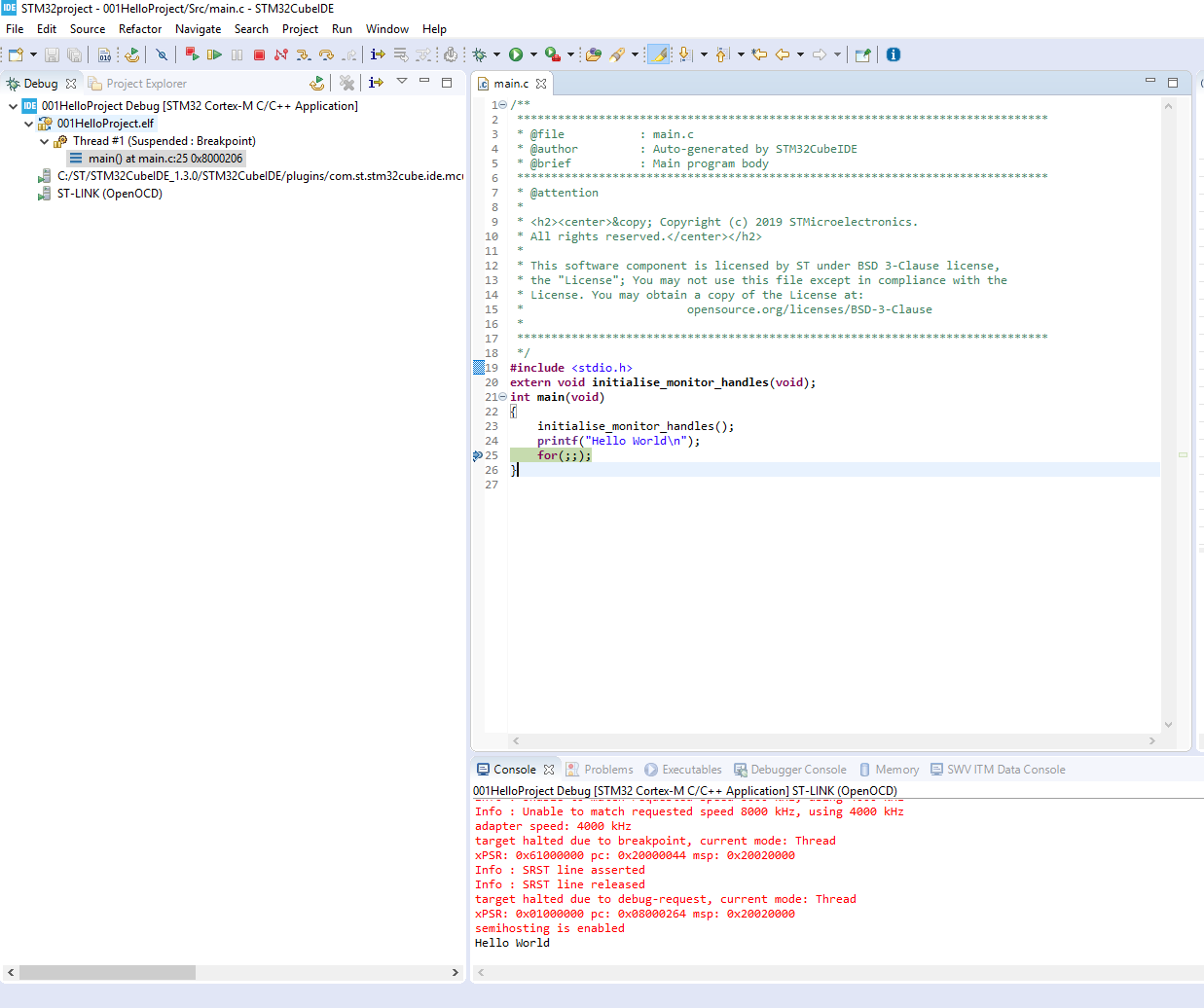


Figure 2 - Semi hosting technique to print "Hello World"

# Activity 3 – In class activity

**Requirement:** Enable SPE bit position of SPI control register 1 (SPI\_CR1) by writing embedded C code.

**Code:**

**#include** <stdint.h>

**#define** SPI\_BASE\_ADDR 0x40013000UL

**#define** SPI\_CR1\_REG\_OFFSET 0x00UL

**#define** SPI\_CR1\_REG\_ADDR (SPI\_BASE\_ADDR + SPI\_CR1\_REG\_OFFSET)

**#define** RCC\_BASE\_ADDR 0x40023800UL

**#define** RCC\_APB2ENR\_REG\_OFFSET 0x44UL

**#define** RCC\_APB2ENR\_REG\_ADDR (RCC\_BASE\_ADDR + RCC\_APB2ENR\_REG\_OFFSET)

**int** **main**(**void**)

{

uint32\_t \*pSpiCr1Reg = (uint32\_t\*) SPI\_CR1\_REG\_ADDR;

uint32\_t \*pRccApb2enrReg = (uint32\_t\*) RCC\_APB2ENR\_REG\_ADDR;

\*pRccApb2enrReg |= (1<<12); //enable the clock

\*pSpiCr1Reg |= (1<<6); //set the 6th bit i.e SPE bit

/\* Loop forever \*/

**for**(;;);

}

**Screenshot:**

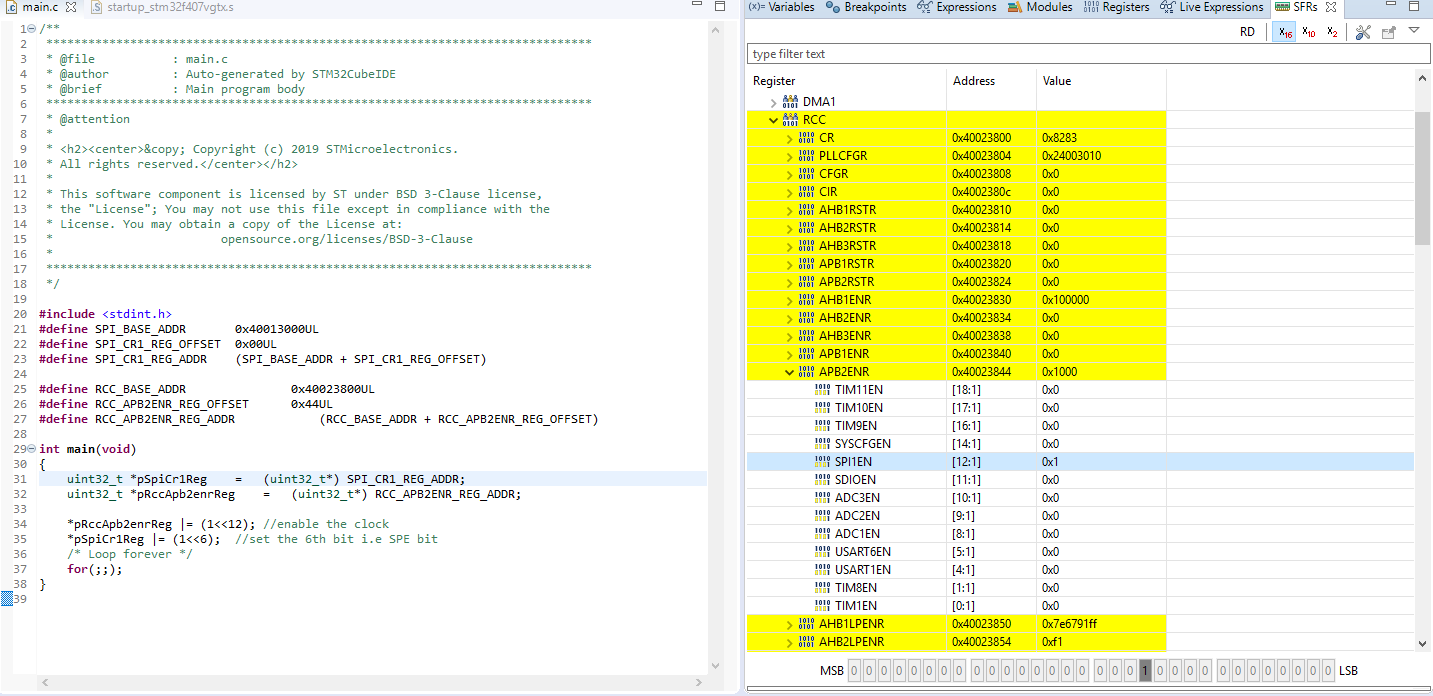


Figure 3 - RCC bit enabled for SPI1 register

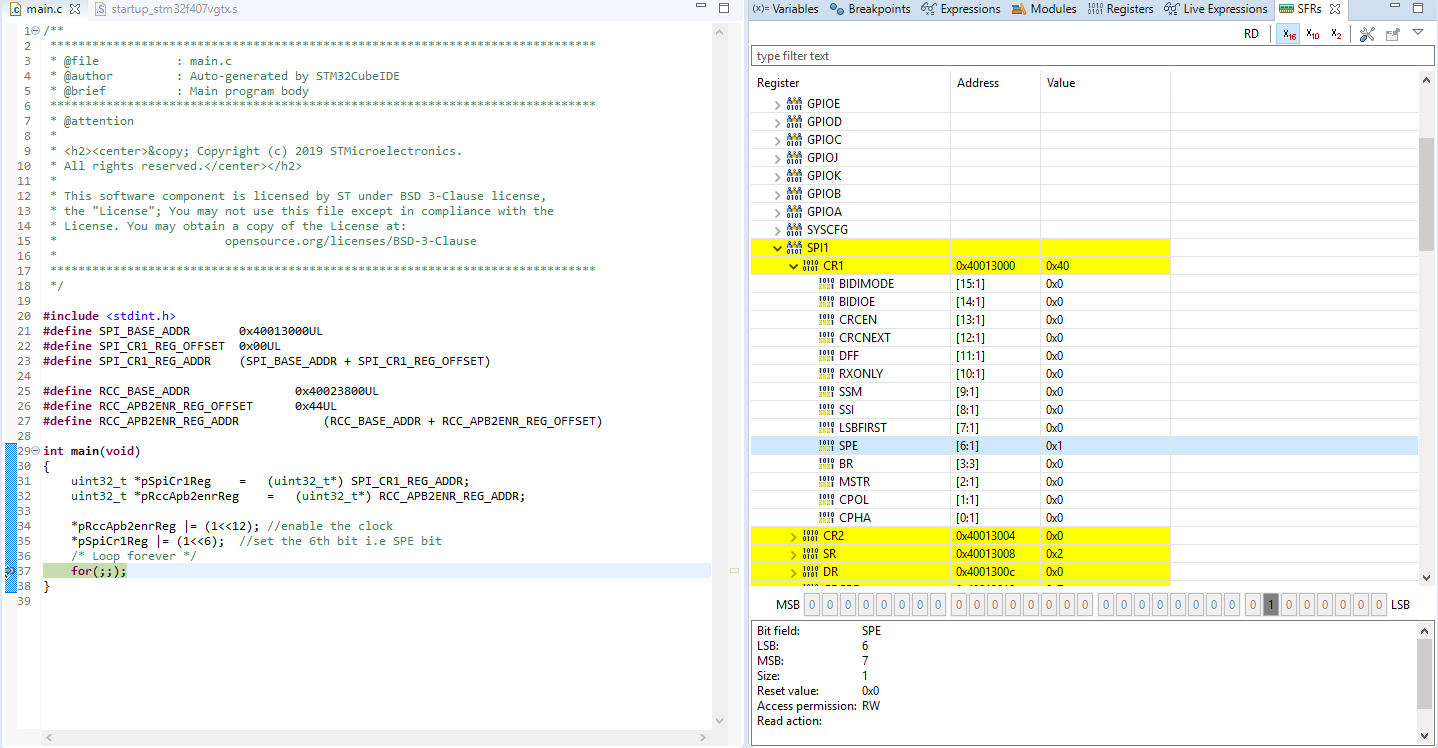


Figure 4 - SPE bit of SPI control register enabled

# Activity 4 – Debugging techniques

**Requirement:** Take any C code and try these debugging options :

* Serial wire viewer and data tracing (printf style debugging)
* Single stepping, stepping over and stepping out.
* Breakpoints (inserting, deleting and skipping breakpoints).
* Disassembly
* Call stack
* Expression and variable window
* Memory browser
* Data watch points

**Code:**

**main.c**

**#include** <stdio.h>

**int** **main**(**void**)

{

**printf**("Hello World\n");

**for**(;;);

}

**syscall.c**

**#include** <sys/stat.h>

**#include** <stdlib.h>

**#include** <errno.h>

**#include** <stdio.h>

**#include** <signal.h>

**#include** <time.h>

**#include** <sys/time.h>

**#include** <sys/times.h>

/////////////////////////////////////////////////////////////////////////////////////////////////////////

// Implementation of printf like feature using ARM Cortex M3/M4/ ITM functionality

// This function will not work for ARM Cortex M0/M0+

// If you are using Cortex M0, then you can use semihosting feature of openOCD

/////////////////////////////////////////////////////////////////////////////////////////////////////////

//Debug Exception and Monitor Control Register base address

**#define** DEMCR \*((**volatile** uint32\_t\*) 0xE000EDFCU )

/\* ITM register addresses \*/

**#define** ITM\_STIMULUS\_PORT0 \*((**volatile** uint32\_t\*) 0xE0000000 )

**#define** ITM\_TRACE\_EN \*((**volatile** uint32\_t\*) 0xE0000E00 )

**void** **ITM\_SendChar**(uint8\_t ch)

{

//Enable TRCENA

DEMCR |= ( 1 << 24);

//enable stimulus port 0

ITM\_TRACE\_EN |= ( 1 << 0);

// read FIFO status in bit [0]:

**while**(!(ITM\_STIMULUS\_PORT0 & 1));

//Write to ITM stimulus port0

ITM\_STIMULUS\_PORT0 = ch;

}

/\* Variables \*/

//#undef errno

**extern** **int** errno;

**extern** **int** **\_\_io\_putchar**(**int** ch) **\_\_attribute\_\_**((weak));

**extern** **int** **\_\_io\_getchar**(**void**) **\_\_attribute\_\_**((weak));

**register** **char** \* stack\_ptr **asm**("sp");

**char** \*\_\_env[1] = { 0 };

**char** \*\*environ = \_\_env;

/\* Functions \*/

**void** **initialise\_monitor\_handles**()

{

}

**int** **\_getpid**(**void**)

{

**return** 1;

}

**int** **\_kill**(**int** pid, **int** sig)

{

errno = EINVAL;

**return** -1;

}

**void** **\_exit** (**int** status)

{

\_kill(status, -1);

**while** (1) {} /\* Make sure we hang here \*/

}

**\_\_attribute\_\_**((weak)) **int** **\_read**(**int** file, **char** \*ptr, **int** len)

{

**int** DataIdx;

**for** (DataIdx = 0; DataIdx < len; DataIdx++)

{

\*ptr++ = \_\_io\_getchar();

}

**return** len;

}

**\_\_attribute\_\_**((weak)) **int** **\_write**(**int** file, **char** \*ptr, **int** len)

{

**int** DataIdx;

**for** (DataIdx = 0; DataIdx < len; DataIdx++)

{

//\_\_io\_putchar(\*ptr++);

ITM\_SendChar(\*ptr++);

}

**return** len;

}

**int** **\_close**(**int** file)

{

**return** -1;

}

**int** **\_fstat**(**int** file, **struct** stat \*st)

{

st->st\_mode = S\_IFCHR;

**return** 0;

}

**int** **\_isatty**(**int** file)

{

**return** 1;

}

**int** **\_lseek**(**int** file, **int** ptr, **int** dir)

{

**return** 0;

}

**int** **\_open**(**char** \*path, **int** flags, ...)

{

/\* Pretend like we always fail \*/

**return** -1;

}

**int** **\_wait**(**int** \*status)

{

errno = ECHILD;

**return** -1;

}

**int** **\_unlink**(**char** \*name)

{

errno = ENOENT;

**return** -1;

}

**int** **\_times**(**struct** tms \*buf)

{

**return** -1;

}

**int** **\_stat**(**char** \*file, **struct** stat \*st)

{

st->st\_mode = S\_IFCHR;

**return** 0;

}

**int** **\_link**(**char** \*old, **char** \*new)

{

errno = EMLINK;

**return** -1;

}

**int** **\_fork**(**void**)

{

errno = EAGAIN;

**return** -1;

}

**int** **\_execve**(**char** \*name, **char** \*\*argv, **char** \*\*env)

{

errno = ENOMEM;

**return** -1;

}

**Screenshot:**

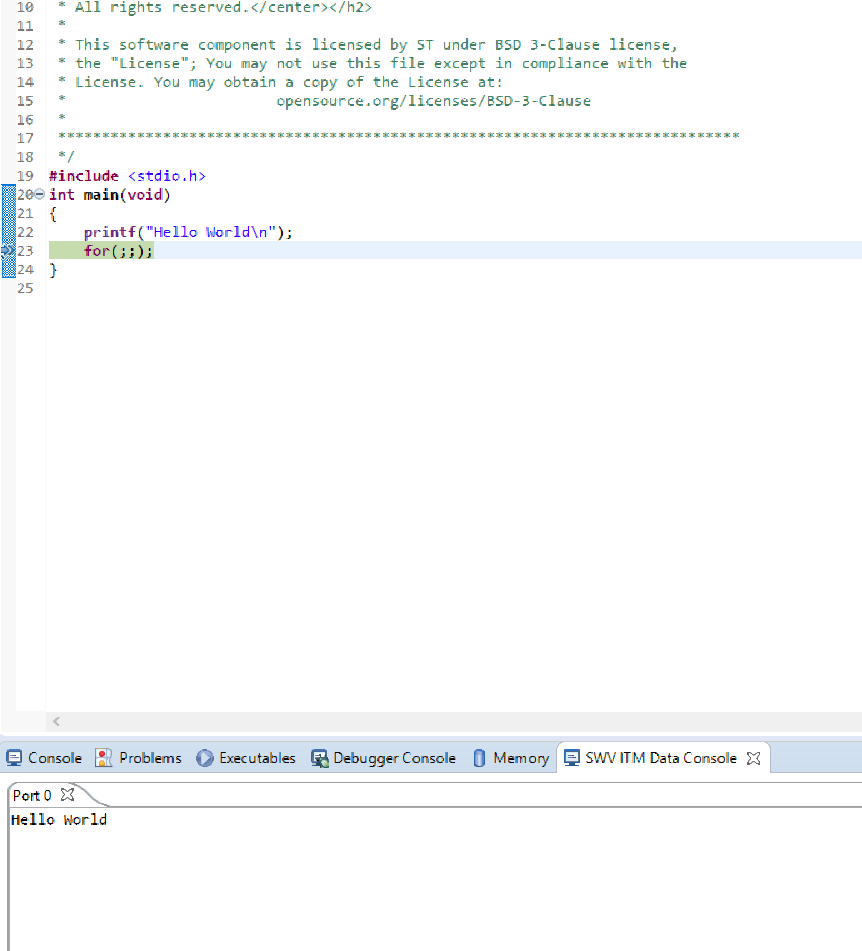


Figure 5 - Serial Wire Viewer

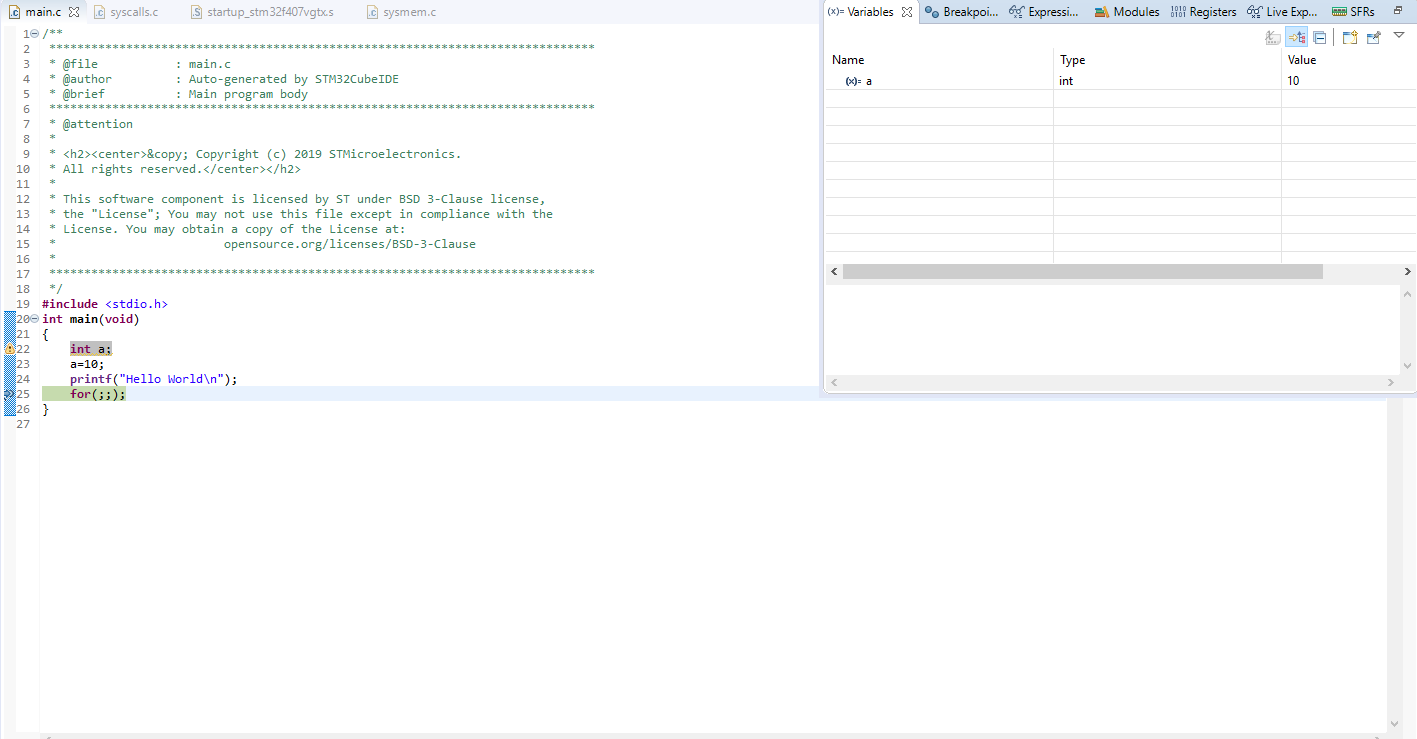


Figure 6 - Variable window

# Activity 5 – Completing MCU specific header files

**Requirement:**