Exercise Manual
For

SC3103
Embedded Programming

Practical Exercise #1
RTOS Based Programming:
Task Creation in FreeRTOS

Venue: Hardware Laboratory 2
(Location: N4-01b-05)

COLLEGE OF COMPUTING AND DATA SCIENCE
NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

In this lab, students will gain hands-on experience to develop a basic multi-task application on an embedded system running FreeRTOS operating System. In addition, students will also learn how to understand an embedded board (based on the ARM Cortex-M4 microcontroller) from its schematic diagrams and datasheets provided by vendors.

The exercises involve developing a basic application from scratch by using the given libraries from vendors and open-source resources, setting up of a cross complier in an IDE and understanding the process to transfer a firmware to the embedded board.

Equipment and accessories required

i) An embedded development board with a STM32F407VET6 microcontroller
ii) FlyMcu - An In-System Programming (ISP) software to upload machine codes to flash memory)
iii) Keil's µVision IDE
iv) ST-Link Programmers & Debuggers
References: Keil Microcontroller Development Kit (MDK-Lite)
http://www.keil.com/arm/mdk.asp
http://www.keil.com/uvision/

---

1. Introduction

1.1 CMSIS Library

The Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for microcontrollers that are based on Arm Cortex processor. CMSIS defines generic tool interfaces and enables consistent device support. The CMSIS software interfaces simplify software reuse, reduce the learning curve for microcontroller developers, and improve time to market for new devices.

1.2 STM32F407VET6

The STM32F407VET6 is a 100-pin high-performance microcontroller chip based on ARM Cortex-M4 32-bit RISC architecture design.

STM32	F	4	07	V	E	T	6
Family	Type	Core	Line	# of Pins	Flash Size	Package	Temperature Range

It is fabricated in a low-profile quad flat package (LQFP) operating at a 168MHz frequency with 512Kbyte Flash memory and 196Kbyte SRAM. It supports up to 82 GPIOs, ten general-purpose timers, three I2C, three SPI, four USART, three 12-bit ADC, two 12-bit buffered DAC etc. (Refer to the datasheet for more details)

<u>Preliminary:</u>

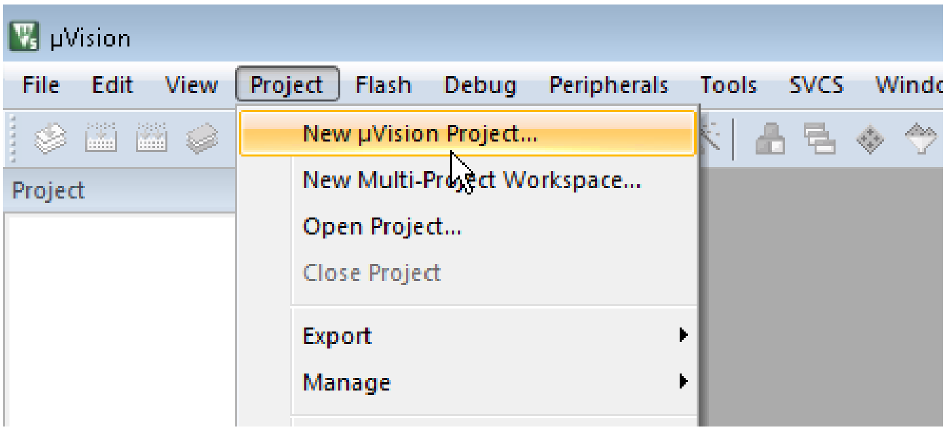
I. Study the data sheet and identify the Flash memory and SRAM address location
Flash memory:
start address: ____________ size: ___________
SRAM:
start address: ____________ size: ___________

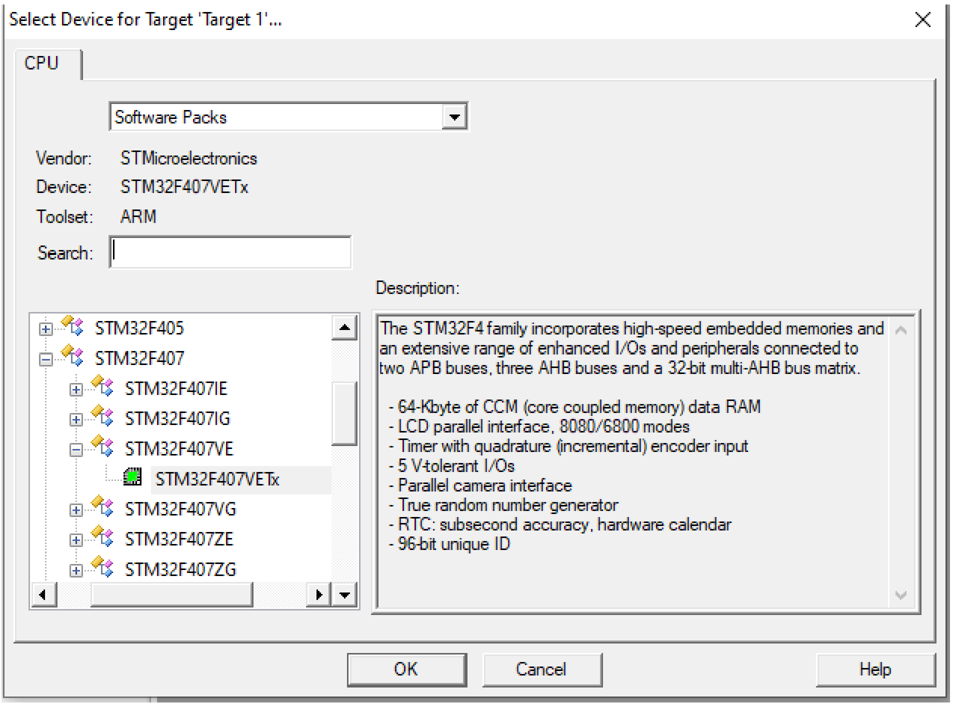
2. Practice Exercises

➢ In these exercises, you will learn how to create a project and configure its various settings for the microcontroller using the µVision 5 IDE. (The debugger is not supported by the µVision 5 and will hence not be covered.) You will then compile the assembly and C programs into an Intel hexadecimal object file format or Intellec Hex (.hex file), which contains the machine code to run in the microcontroller.

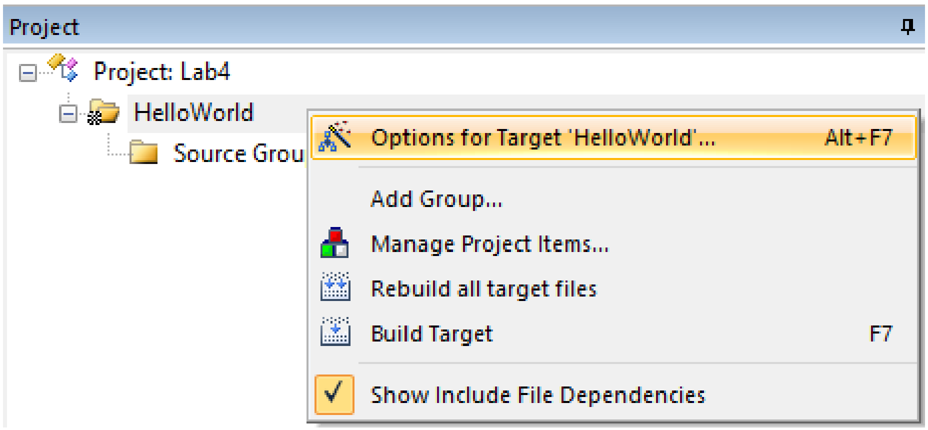
2.1 Practice A – Getting Started: Hello µVision 5

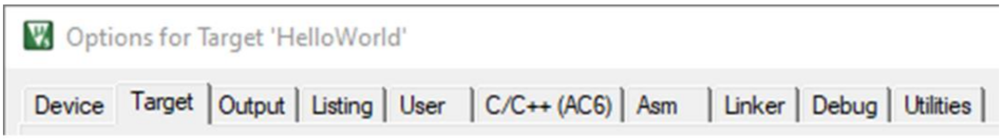
➢ First, create a new project (.uvproj) in µVision 5 by selecting STM32F407VE as shown in the following diagrams





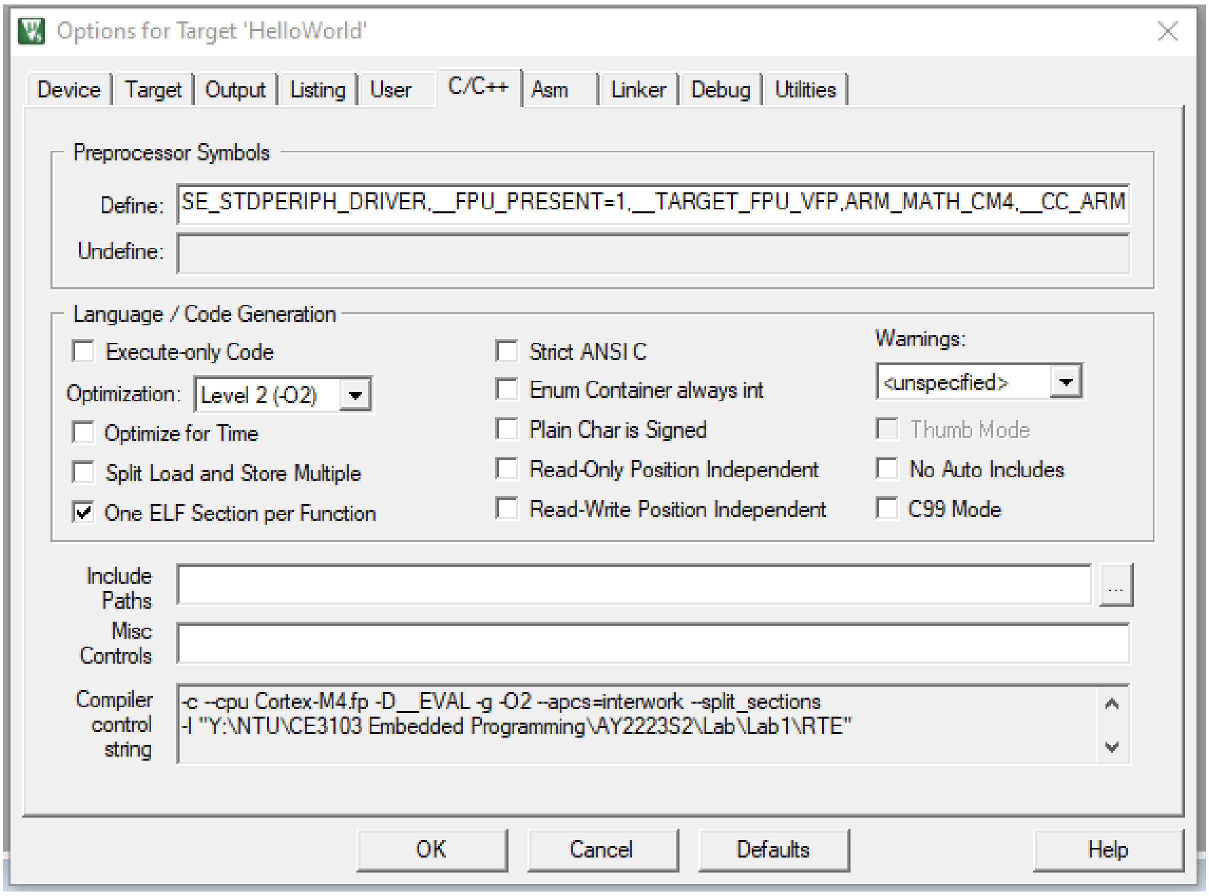
➢ Next we need to <u>configure the project's options</u> to match the embedded platform we will be using.





The following are a number of MUST select options to be configured :

a) Target:
   a. ARM Compiler: "Use default compiler"
   b. Xtal(Mhz): 168
b) Output: select "**Create HEX File**"
c) C/C++ compiler : See below (note the compiler control flag used)



d) Linker: select “**Use Memory Layout from Target Dialog**”

e) Preprocessor Symbols:
STM32F40_41xxx,USE_STDPERIPH_DRIVER,__FPU_PRESENT=1,__TARGET_FPU_VFP,ARM_MATH_CM4,__CC_ARM

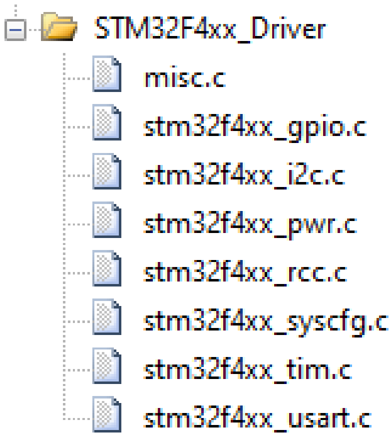
➢ Adding of skeleton files:

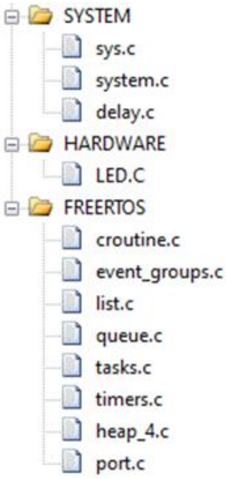
a) Select the following ARM CMSIS library files for the project

- startup_stm32f40_41xxx.s: provide all initial configuration. (You can find out how the microcontroller knows where your main function is.)
    - system_stm32f4xx.c: provide system initialization especially the system clock setup. (You will find out the reason for selecting STM32F40_41xxx in Preprocessor Symbols in **the compiler setup)**

<u>Questions:</u> What is the System Clock? ___________

b) Peripheral Driver Files: You may not need all of them - figure out which one you actually need.



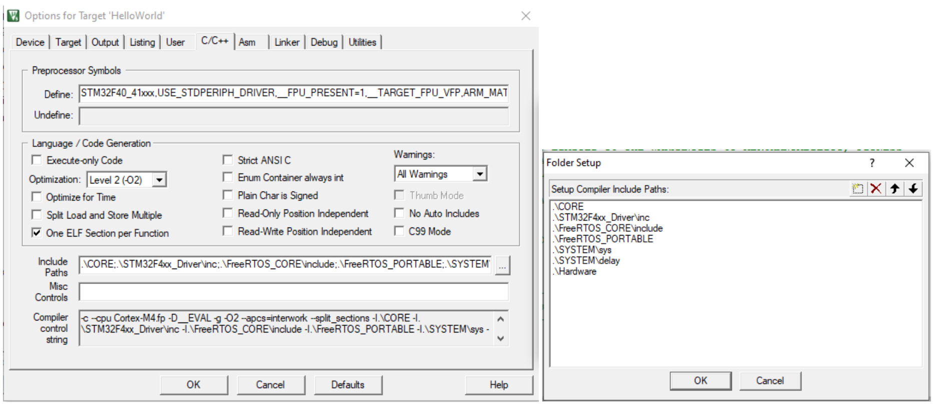


<u>Peripheral Driver</u> **<u>FreeRTOS</u>**

c) FreeRTOS library. You will not need all the FreeRTOS modules at this stage, but we just add them first. The portable implementation of ARM CM4F and heap_4 for memory management are in use in this helloWorld project.

d) Some system initialization, delay and led configuration in sys.c, system.c, delay.c and led.c are also added.

e) The corresponding header files (.h files) are required in the compilation. Thus, we also need to indicate the folder where you store these header files.



2.2 Practice B – Hello STM32F103 – LED Blink

Study the LED.c file. Note the **`LED_Init()`** function which show how the configuration of the LED is done, and the **`led_task`** function that provides a task to blink the LED3.

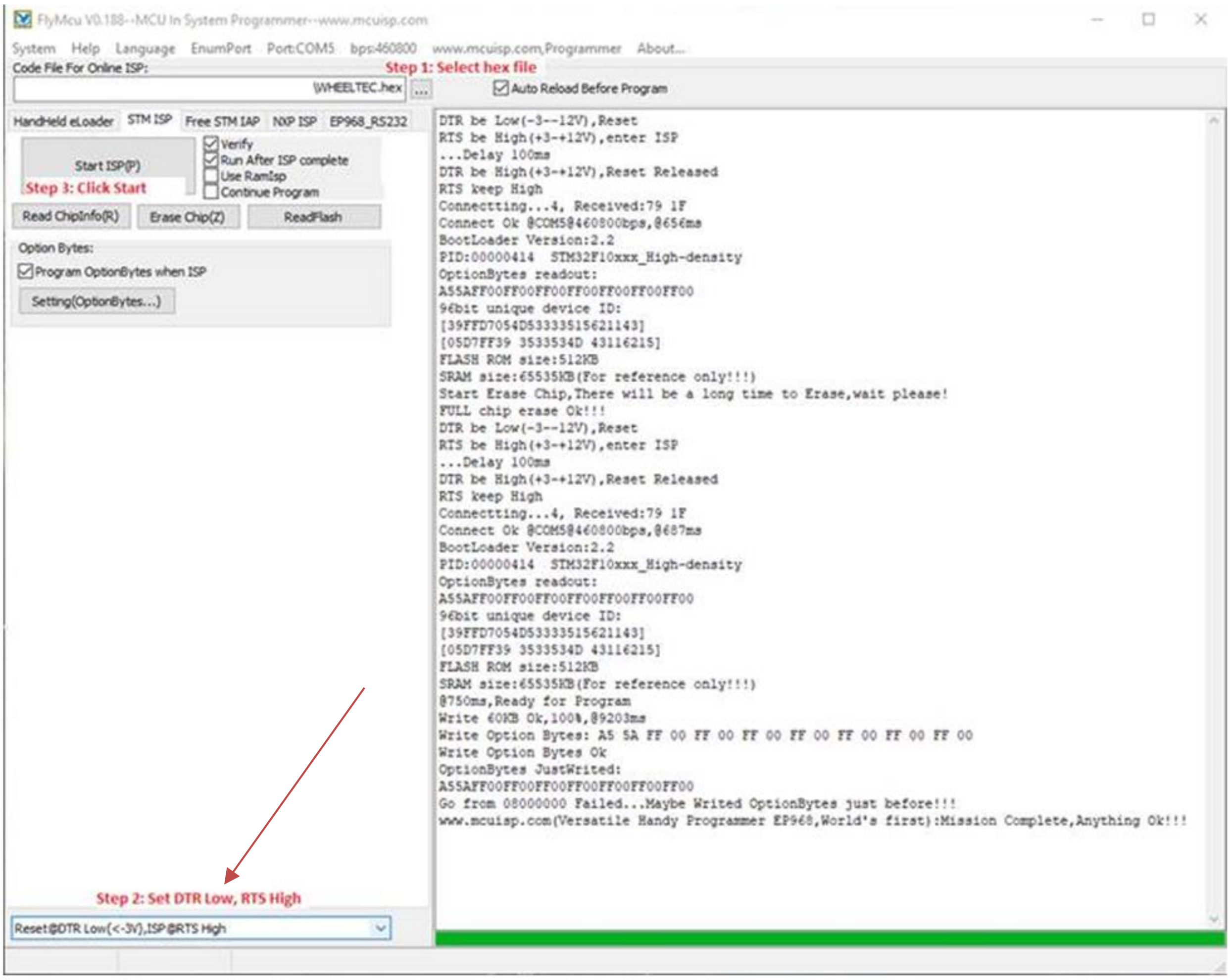
- Use the following code to create a task in the main() to blink the LED as follow:

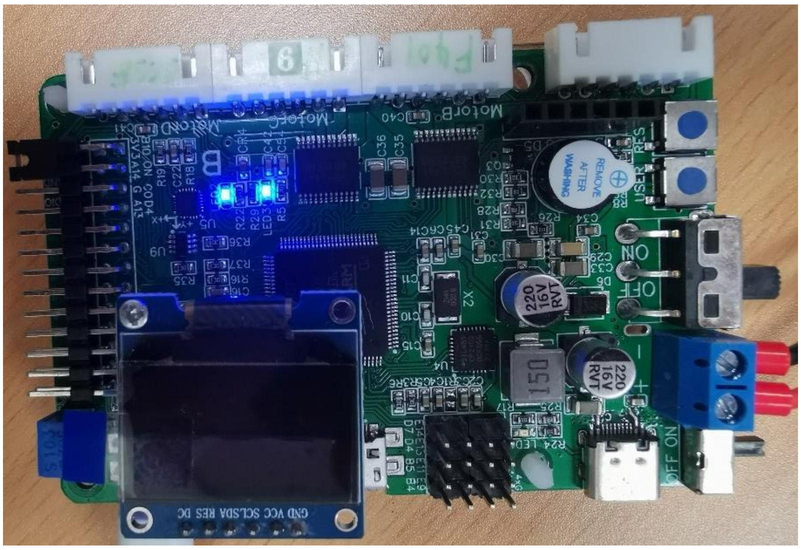
```
#include "system.h"

#define START_TASK_PRIO      4
#define START_STK_SIZE       256
TaskHandle_t StartTask_Handler;
void start_task(void *pvParameters);

int main(void)
{
   systemInit();


    xTaskCreate((TaskFunction_t )start_task,
                (const char*    )"start_task",
                (uint16_t       )START_STK_SIZE,
                (void*          )NULL,
                (UBaseType_t    )START_TASK_PRIO,
                (TaskHandle_t*  )&StartTask_Handler);

    vTaskStartScheduler();

}

void start_task(void *pvParameters)
{

    taskENTER_CRITICAL();

    xTaskCreate(led_task, "led_task", LED_STK_SIZE, NULL, LED_TASK_PRIO, NULL);

    vTaskDelete(StartTask_Handler);

    taskEXIT_CRITICAL();

}
```

- Build the program. If compile successfully, you will find a hex file in your object folder.

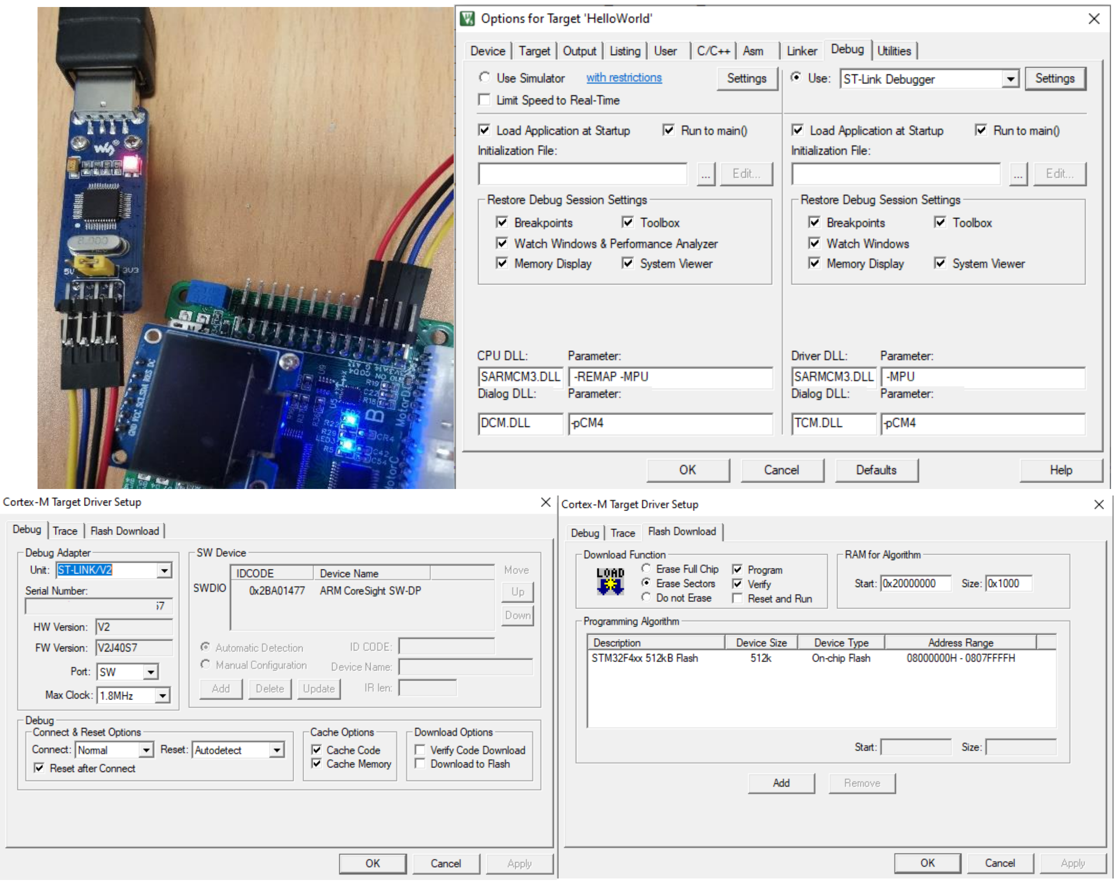
- You can now use the FlyMcu software to upload this hex file the board. Make sure that the option setting in step 2 is:

“**`Reset@DTR Low (<-3V), ISP @RTS High`**”





➢ Alternately, you can download the axf file to the board via ST-Link Debugger. First, connect the ST-link to your board as the figure below. Next, you need select ST-Link Debugger under the option->Debug. Under the Settings beside the dropdown list, please make sure the SW port is selected for debugging. Other settings in figure below are default settings. If yours are not, please change accordingly.


Options for Target 'HelloWorld'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Simulator
with restrictions
Settings
Limit Speed to Real-Time
Use: ST-Link Debugger
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Memory Display
System Viewer
Watch Windows
CPU DLL:
Parameter:
SARMCM3.DLL
-REMAP -MPU
Dialog DLL:
DCM.DLL
pCM4
Driver DLL:
-MPU
TCM.DLL
OK
Cancel
Defaults
Help
Cortex-M Target Driver Setup
Debug
Trace
Flash Download
Debug Adapter
Unit: ST-LINK/V2
Serial Number:
HW Version: V2
FW Version: V2J40S7
Port: SW
Max Clock: 1.8MHz
SW Device
IDCODE
Device Name
SWDIO
0x2BA01477
ARM CoreSight SW-DP
Move
Up
Down
Automatic Detection
ID CODE:
Manual Configuration
Device Name:
Add
Delete
Update
IR len:
Connect & Reset Options
Connect: Normal
Reset: Autodetect
Reset after Connect
Cache Options
Cache Code
Cache Memory
Download Options
Verify Code Download
Download to Flash
OK
Cancel
Apply
Download Function
Erase Full Chip
Erase Sectors
Do not Erase
Program
Verify
Reset and Run
RAM for Algorithm
Start: 0x20000000
Size: 0x1000
Programming Algorithm
Description
Device Size
Device Type
Address Range
STM32F4xx 512kB Flash
512k
On-chip Flash
08000000H - 0807FFFFH
Start:
Size:
Add
Remove

3. Exercises

Based on what you have learnt so far, you are now going to develop a multiple task program to control the various peripherals found on the embedded board

3.1 Exercise A – Buzzer Task

- Modify the code to make the delay in `led_task()` to be of 5 seconds.

- Create another new task **buz_task()** to turn on and off the on-board buzzer at every 7 seconds intervals. A separate `.c` file and `.h` file for buzzer should be created. (You should refer to the given schematic diagram to find out the GPIO port of the buzzer.)

3.2 Exercise B - OLED Display Task

- Study the source code of the given `show.c` and `oled.c` files.

- Add another task to your program to display some text string (e.g., your name) on the OLED display screen.

3.3 Exercise C – Design A Digital Timer on the OLED Display

- Add another task or reuse the previous OLED task to display a digital timer on the OLED display screen.

- The timer will update every one second.