

## CG2271: Real-Time Operating Systems

### Lab2: Programming your Ports

#### Objective

In this lab, we will learn how to create our own project using the Keil IDE. We will also develop our own device-driver code for the GPIO Peripheral block.

#### Equipment Needed:

FRDM Board + USB Cable  
Laptop with Keil IDE Software

#### Introduction

In Lab 1, we imported an existing project, compiled it and download to our board. As a result, we saw the RGB LED changing colours as the program executed. In this lab, we will first be creating our own project to see how simple it is. After that, we will be programming the ports that control the RGB LED so that we can see how it actually works.

### A. Creating your own Project

Steps:

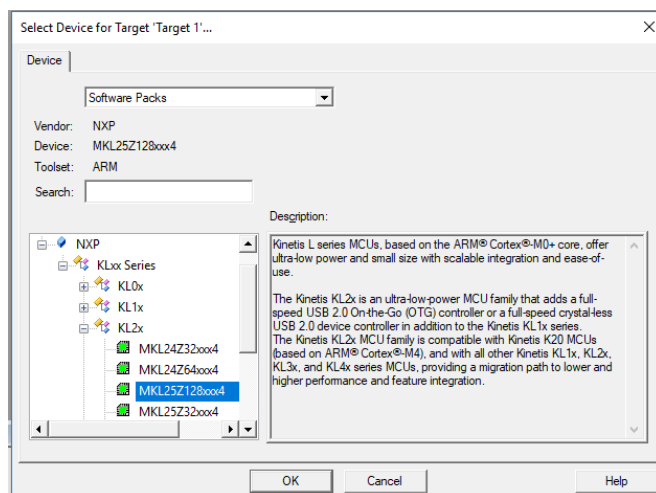
1. Start  $\mu$ Vision.

Create a new Directory and New Project:

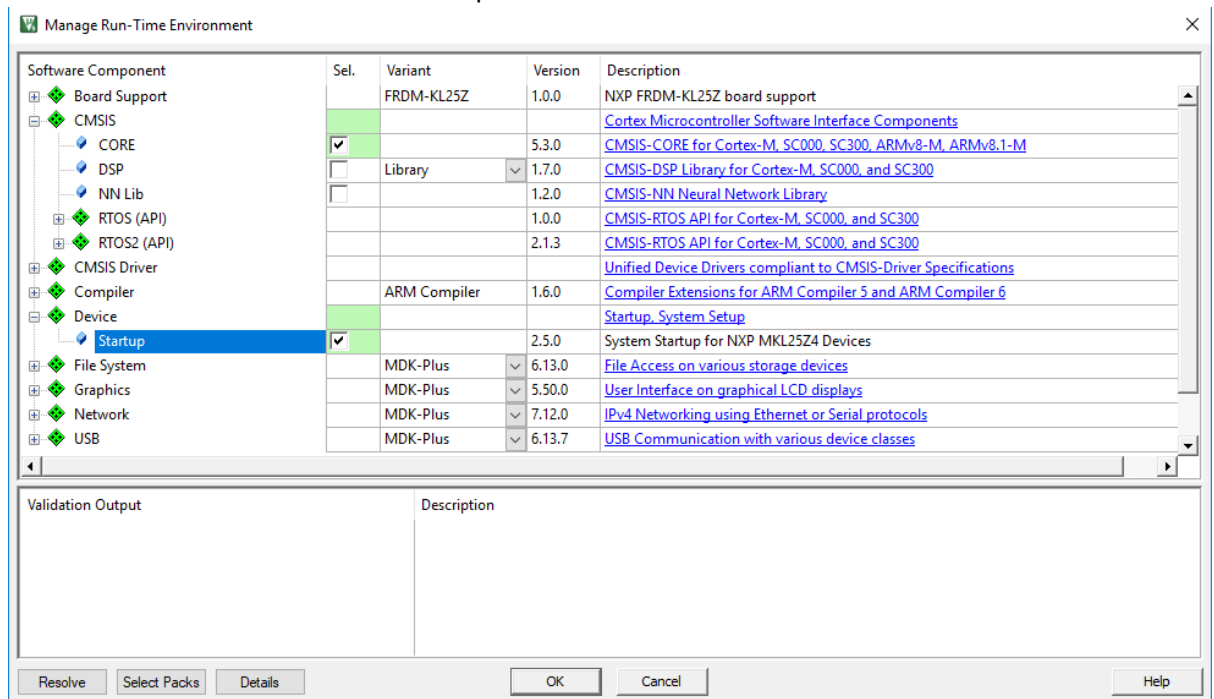
2. Click on Project -> New  $\mu$ Vision Project
3. Navigate to your desired folder and name the file as 'MyBlinky'
4. This creates the MyBlinky.uvproj
5. As soon as you click Save, a new window opens

Select the Device you are using:

6. Expand NXP then KLxx Series, then KL2x and then select MKL25Z128xxx4 as shown:

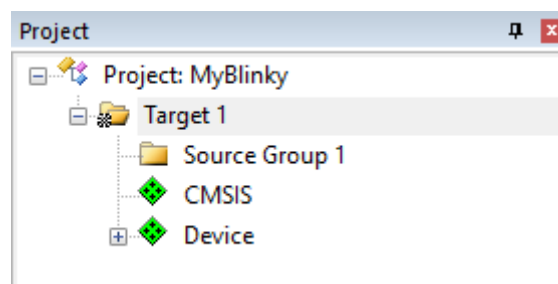


7. Click OK and the Manage Run Time window shown below opens.  
Select CMSIS->Core and Device->Startup



8. Click OK.

You have now created a blank  $\mu$ Vision Project. Now you need to add you own source files.

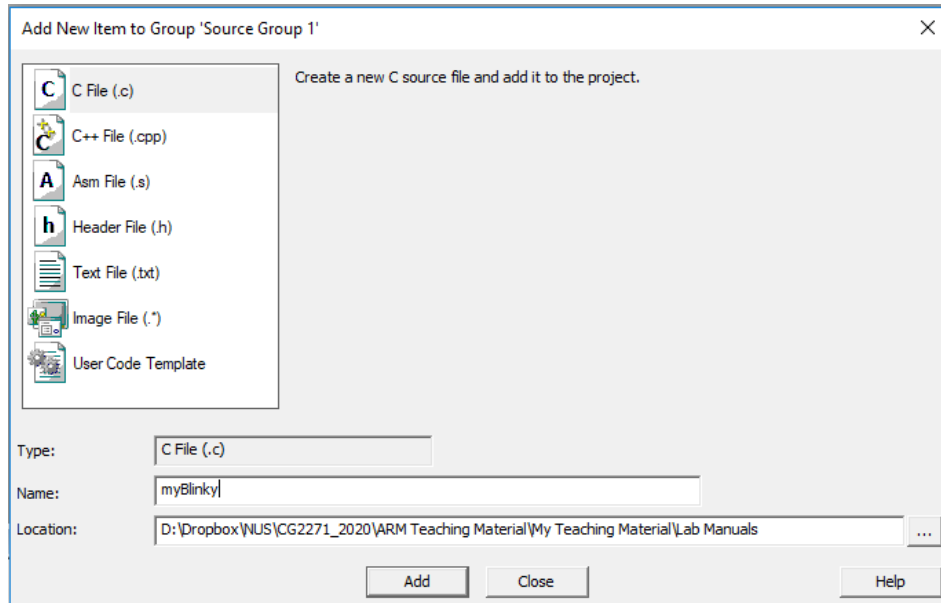


Create a blank C source file:

9. Right click on Source Group 1 in the Project window and select

Add New Item to Group 'Source Files'...

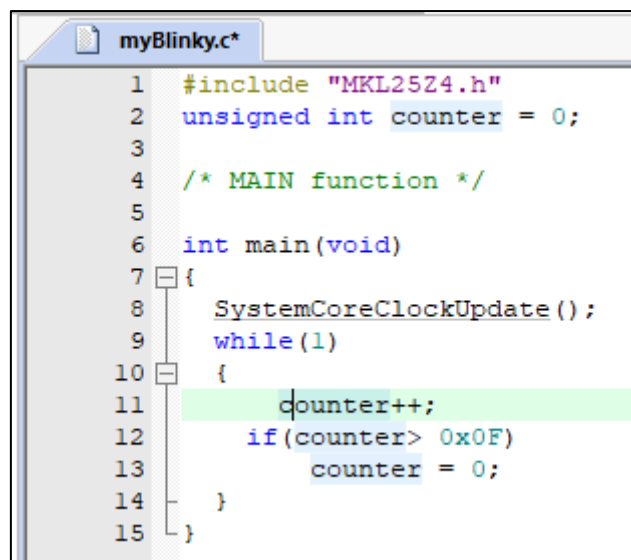
## 10. This window opens up:



11. Highlight the upper left icon: C file (.c):
12. In the Name: field, enter myBlinky.
13. Click on Add to close this window.
14. Click on File/Save All
15. Expand Source Group 1 in the Project window and myBlinky.c will now display.
16. It will also open in the Source window.


Add some code to myBlinky.c

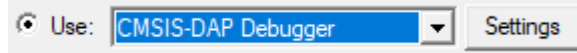
17. In the blank myBlinky.c, right click and select Insert '# include file'.
- Select # include <MKL25Z4.h>
18. Add the rest of the code as shown in the box below:



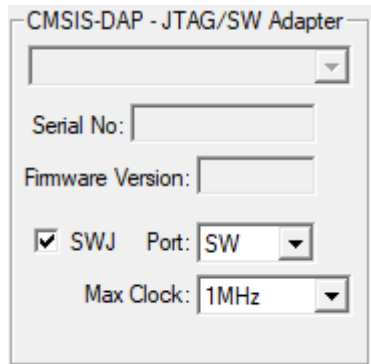
19. Click on File/Save All
20. Build the files. There will be no errors or warnings if all was entered correctly.

Configure the Target CMSIS-DAP Debug Adapter:

21. Select the Target Options icon. . You can also use Alt-F7 to launch the options.
22. Select Use MicroLIB to optimize for smaller code size.
23. Click on the Debug tab. Select CMSIS-DAP Debugger in the Use: box:

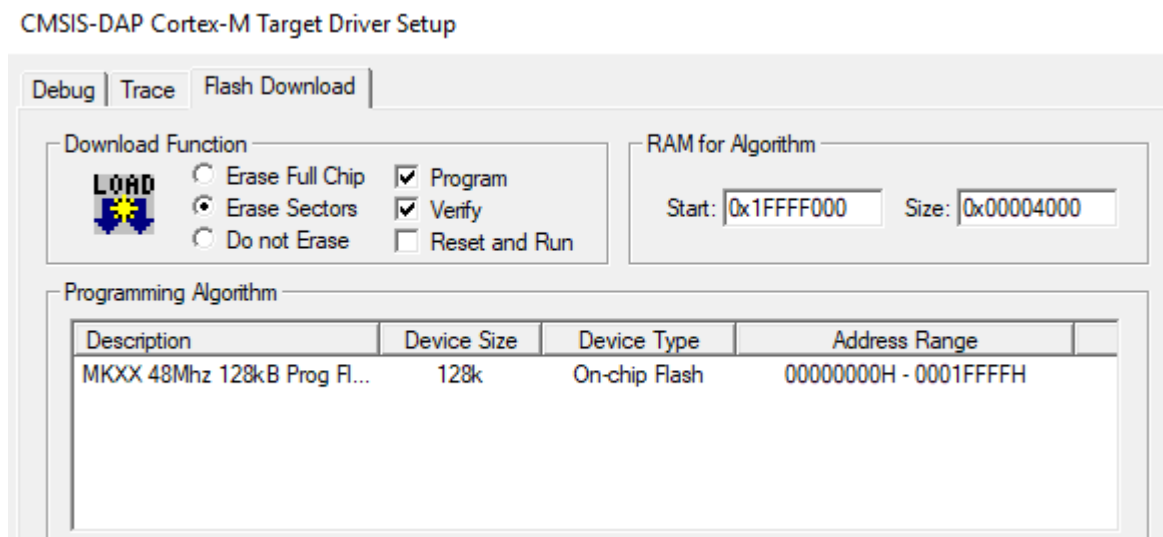


24. Select Settings: icon beside Use: CMSIS-DAP.
25. Select SWJ and SW as shown here:



Confirm Flash Programming Algorithm is Configured:

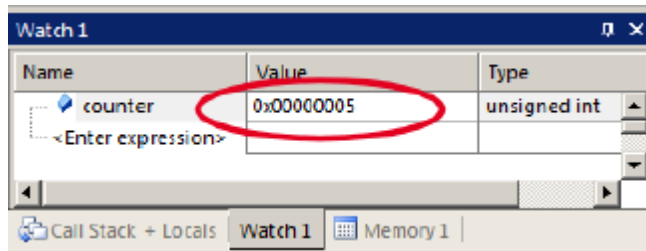
26. Click on OK once to go back to the Target Configuration window.
27. Click on the Utilities tab. Select Settings and confirm the correct Flash algorithm is present:  
Shown is the correct one for the Freedom KL25Z board:



28. Click on OK twice to return to the main menu.
29. Click on File/Save All
30. Build the files. There will be no errors or warnings if all was entered correctly. If there are, please fix them !

Downloading and Running your Program.

31. Enter Debug mode by clicking on the Debug icon .
32. Click on the RUN icon.
33. Right click on <counter> in Blinky.c and select Add counter to ... and select Watch 1.
34. The <counter> variable will be updating here:



Note: The Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

35. You can also set a breakpoint in myBlinky.c and the program should stop at this point if it is running properly. Experiment with setting breakpoints in different parts of the code and try stepping through the code.

## B. Programming the Ports

In the Lecture, we saw how we can program the Ports to control the necessary pins. Let's review the RGB LED connections from Lab 1.

### RGB LED

Three PWM-capable signals are connected to a red, green, blue LED, D3. The signal connections are shown in Table 1 below. Figure 2 shows the schematic.

RGB LED	KL25Z128
Red Cathode	PTB18
Green Cathode	PTB19
Blue Cathode	PTD1 <sup>1</sup>

Table 1: Mapping of RGB LED

(Note 1: PTD1 is also connected to the I/O header on J2 pin 10 (also known as D13).)

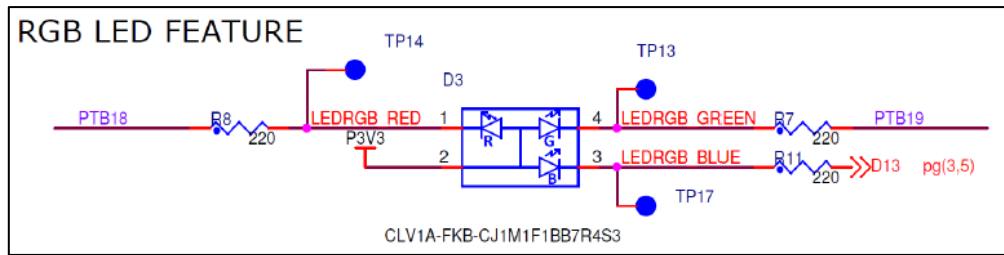


Figure 2: RGB LED Schematic

It can be seen that we need to control PTB18, PTB19 and PTD1 in order to change the output of the RGB LED.

The code to complete the GPIO Initialization is given here:

```
#define RED_LED      18    // PortB Pin 18
#define GREEN_LED    19    // PortB Pin 19
#define BLUE_LED     1     // PortD Pin 1
#define MASK(x)      (1 << (x))

void InitGPIO(void)
{
    // Enable Clock to PORTB and PORTD
    SIM->SCGC5 |= ((SIM_SCGC5_PORTB_MASK) | (SIM_SCGC5_PORTD_MASK));

    // Configure MUX settings to make all 3 pins GPIO

    PORTB->PCR[RED_LED] &= ~PORT_PCR_MUX_MASK;
    PORTB->PCR[RED_LED] |= PORT_PCR_MUX(1);

    PORTB->PCR[GREEN_LED] &= ~PORT_PCR_MUX_MASK;
    PORTB->PCR[GREEN_LED] |= PORT_PCR_MUX(1);

    PORTD->PCR[BLUE_LED] &= ~PORT_PCR_MUX_MASK;
    PORTD->PCR[BLUE_LED] |= PORT_PCR_MUX(1);

    // Set Data Direction Registers for PortB and PortD
    PTB->PDDR |= (MASK(RED_LED) | MASK(GREEN_LED));
    PTD->PDDR |= MASK(BLUE_LED);
}
```

You are free to use your own code if you wish to do so.

Complete the entire code, to achieve the objective of lighting up the RGB LED one colour at a time, e.g. RED->GREEN->BLUE->RED->GREEN->....

**IMPORTANT HINT:** Develop reusable code by producing good device driver libraries that can be used in future projects. This skillset is important in ensuring that you can develop code that will be used in the future.

**IMPORTANT HINT:** Have a good look at the schematic to understand how the LED is configured. Once you have achieved the objective, demonstrate your code to the Lab TA to get the marks for this lab. You have until the following week to demo the functionality to your TA.

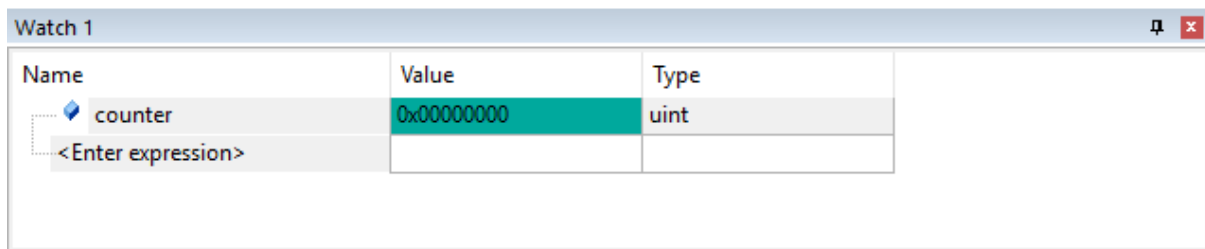
#### Demo Requirements:

- LED Flashing in Running Sequence, One Colour at a time.
- Development of High-Level Library Functions for the LED Control.  
e.g.  
led\_control(color\_t color)
  - ➔ color\_t is a enum holding possible color types
  - ➔ This function takes only the color value and controls the appropriate LED.

### C. What's My Clock?

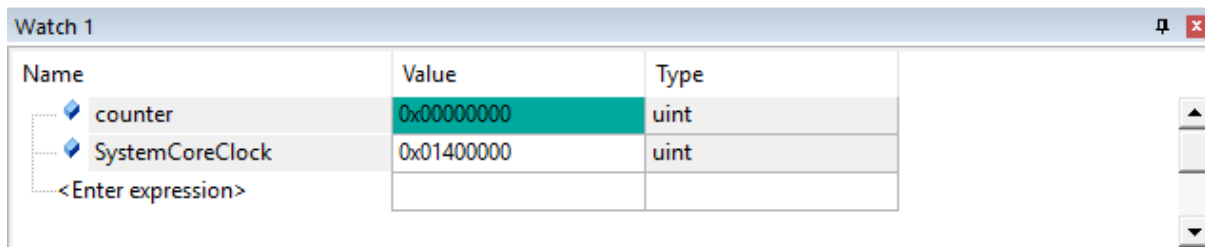
Now that your program is running as expected, at what frequency are you actually clocking the Cortex-M0+ core?

Let's observe the core clock frequency currently used by the code. Select the <Enter expression> space in the Watch 1 window.



Enter the following variable name: SystemCoreClock

You should see the result as such:



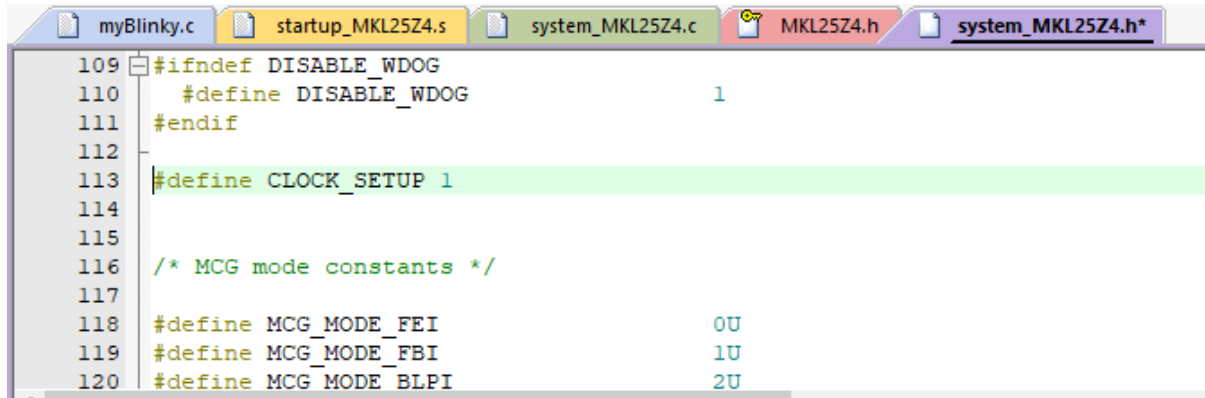
Right-Click on <SystemCoreClock> and select the option that says "Hexadecimal Display" that is current has a tick beside it. The value will now display as decimal 20971520. This is the default configuration of the system with:

Core clock = 20.97152MHz

Bus clock = 20.97152MHz

We will change it to operate at its highest speed of 48MHz. To do this, we need to open the file <system\_MKL25Z4.h> you should see this in the Project space on your left of the IDE. Add the line:

```
#define CLOCK_SETUP 1
```



```

109 #ifndef DISABLE_WDOG
110     #define DISABLE_WDOG        1
111 #endif
112
113 #define CLOCK_SETUP 1
114
115
116 /* MCG mode constants */
117
118 #define MCG_MODE_FEI            0U
119 #define MCG_MODE_FBI            1U
120 #define MCG_MODE_BLPI           2U
  
```

Now recompile and Debug your code. You should see that the SystemCoreClock is now 48MHz.

Watch 1		
Name	Value	Type
counter	0x00000001	uint
SystemCoreClock	48000000	uint
<Enter expression>		

Well Done!

## CONCLUSION

Great Job in getting started with bare-metal coding for the Cortex-M0+ processor. This skillset will give your LinkedIn Profile a great boost! Remember to get it updated soon! 😊