Week 3 Lab

CC2511

This week, you will write your first microcontroller program. Your objective is to control a LED on the development board. You'll also learn how to use the debugger to step through your program line-by-line and inspect memory.

Task Description

By turning on or off different colours in an RGB LED, there are 7 possible colours that can be made:

Red	**Green**	**Blue**	**Resultant colour**
✓	-	-	Red
-	✓	-	Green
-	-	✓	Blue
✓	✓	-	Yellow
✓	-	✓	Purple
-	✓	✓	Cyan
✓	✓	✓	White

Your task is to demonstrate all seven colours by writing a program that iterates through each combination. **Follow the steps below to learn how to configure the microcontroller to drive the LEDs.**

Terminology

- A **pin** is a connection between the microcontroller and the rest of the circuit.
- A **port** is a collection of pins. On Kinetis microprocessors, ports are labelled with letters, e.g. Port A, Port B, Port C.
- The pins within a port are numbered, e.g. Port A Pin 0, Port B Pin 1. These are abbreviated as "PTA0" and "PTB1" respectively.

Identifying the microprocessor pins

The FRDM boards have a tricolour LED connected to digital output pins on the microprocessor. This device has three separate LEDs – red, green, and blue – in a common housing. To use this LED, follow these steps:

1. Open up the schematic for the FRDM board. You can download this from LearnJCU.

Question 1—What is the electrical configuration of the tricolour LED? Is it a common cathode or a common anode?

Question 2—To power on the LED, do you set the output pins to logic HIGH or logic LOW?

2. Locate the RGB LED on the schematic.

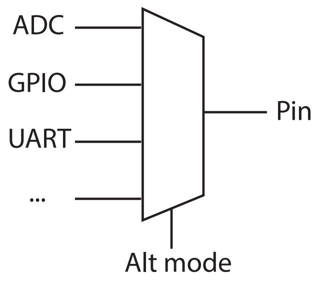
3. By following the schematic, **determine which microprocessor pins are connected to which colour**. Complete the following table[1]:

LED Colour	Microprocessor pin
Red	PTA1
Green	PTA2
Blue	PTD5

[1] You do not need to write the full name of the pin. Here, you can use the GPIO name which is of the format "PT*xx*", e.g. PTA1, PTB2, etc.

Signal Multiplexing

The microprocessor has many different blocks of functionality, such as different communication interfaces, voltage sensing, and digital input/output. Each physical pin is routed through a multiplexer that specifies which of these signals is the one to be connected through to the pin. Our device describes these as "alternatives".



In the **Reference Manual Chapter 10** there is a table listing the signal multiplexing and pin assignments. The "alternative mode" for each pin defines which signal is connected to each pin. You can find the Reference Manual on LearnJCU.

We will drive the LEDs by using the microprocessor's general purpose input/output (GPIO) function. In GPIO mode, each pin can be configured to generate either a low voltage (0 V) or a high voltage (3.3 V). To use a pin as general purpose input/output (GPIO), the relevant "ALT" function must be selected[2]. **You must identify which ALT mode is needed to use the pin for GPIO.** The GPIO function will appear in the table under a name such as "PT*xn*", for example "PTA0" and "PTB1".

[2] Examples of ALT modes are ALT0, ALT1, ALT2, ...

Complete the following table for each of the pins that you identified above:

Microprocessor pin	ALT mode to enable GPIO
PTA1	ALT1
PTA2	ALT1
PTD5	ALT1

Port Control

Pins are configured using the Port Control module. This module is specified in the **Reference Manual Chapter 11**, and in particular, the section "**Memory map and register definition**".

Every pin has a dedicated **pin control register** named in the format `PORTx_PCRn`. For each of the three pins that you identified above, you must set the corresponding pin control register in order to enable the required "ALT" function.

Carefully study the description of the `PORTx_PCRn` fields to determine the values needed to enable the required ALT mode. Complete the following table:

Port Control Register	**Value**
PORTA_PCR1	1<<8
PORTA_PCR2	1<<9
PORTD_PCR5	(1<<10)|(1<<8)

Data Direction

GPIO pins can be configured as either digital input or digital output. In digital input mode, the microprocessor senses whether the voltage on the pin is high or low. In digital output mode, the microprocessor acts as a voltage source and generates either a low voltage or a high voltage. This setting is referred to as the data direction. Examine the chapter on **General-Purpose Input/Output (GPIO)** to determine how to configure each pin as digital output.

Each port has a **data direction register**. Each bit within that register defines the data direction for the corresponding pin within the port. Fill in the table below with the values necessary to configure the respective pins as outputs.

Data Direction Register	**Value**
GPIOA_PDDR	0b1
GPIOA_PDDR	0b10
GPIOD_PDDR	0b10000

Controlling each LED

A GPIO pin can output either logic low (aka logic 0, which is 0 volts), or logic high (aka logic 1, which is 3.3 volts on this system). There are multiple ways to interact with the GPIO pins.

1. You can set the value directly by writing into the **port data output register**. There is a single register for the entire port, and each bit corresponds to a pin. The least significant bit corresponds to pin 0. For example, the binary value 1000 0111 would set a logic high on pins 7, 2, 1, and 0. All other pins would be zero to logic low.

   This mode enables you to change the value of all pins in the port in a single statement. However, if you are using multiple pins for different purposes, you need to be careful to change only the bits of interest. This can be done using bitwise logic statements such as AND or OR, e.g.

   ```
   GPIOA_PDOR = GPIOA_PDOR | (1 << 4);
   ```

   would OR the existing value with 1 shifted left 4 bits (i.e. 10000 binary) and thereby turn pin 4 on.

EXTENSION TOPIC—The code here (writing into GPIOA_PDOR) has an issue. Can you see it?
The CPU must read the value of the register, then compute the OR, then write it back. If the register should change in the meantime (perhaps because this code was interrupted by another task), the *old* value of the other bits will be restored. This is called a *race condition* and can cause intermittent bugs. The port clear and port set registers are preferred in this case.

2. An alternative interface allows direct control over each pin without interfering with others in the same port. This can be achieved using the **port clear output register** and the **port set output register**. Each binary 1 written into the clear register will clear the corresponding pin (to zero); whereas each binary 1 written into the set register will set the corresponding pin to one. For example, the binary value 1000 0000 written into the clear register would cause pin 7 to be changed to zero whereas all other pins would be unchanged. A convenient way to generate this binary value would be `(1 << 7)`.

Your task is to control all three of the LEDs. Refer to the GPIO chapter to determine which registers need to be set to which value.

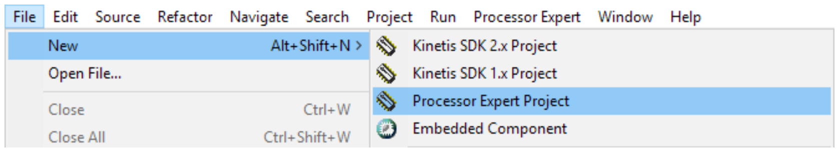
Colour	Register setting for LED On	Register setting for LED Off
Red	GPIOA_PDOR = GPIOA_PDOR l (1)	
Green	GPIOA_PDOR = GPIOA_PDOR l (1)	
Blue	GPIOD_PDOR	

Writing the software

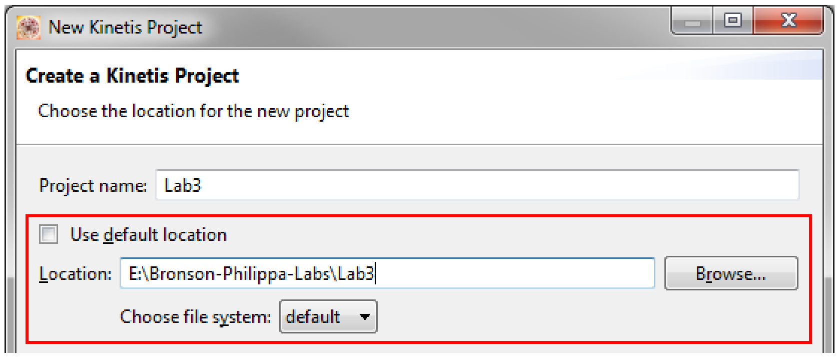
Now that you have identified which registers need to be set to which values, you can write the software to achieve this.

Creating a Kinetis Design Studio project

1. Start up Kinetis Design Studio.
2. Select File -> New -> **Processor Expert Project**.



3. Name your project "**Lab3**". **Ensure that the location is inside the Git repository that you cloned in the first week.**


New Kinetis Project

Create a Kinetis Project

Choose the location for the new project

Project name: Lab3

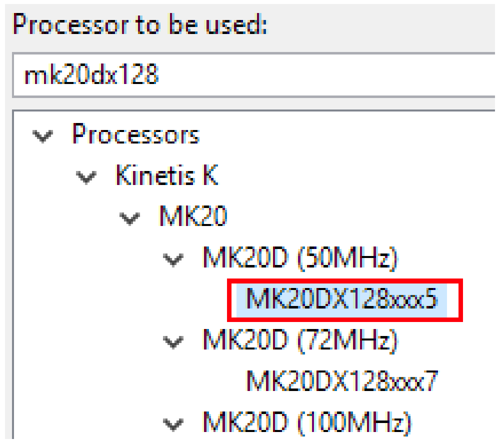
Use default location: unchecked

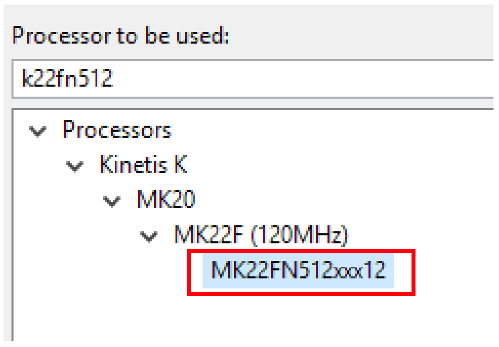
Location: E:\Bronson-Philippa-Labs\Lab3

Choose file system: default

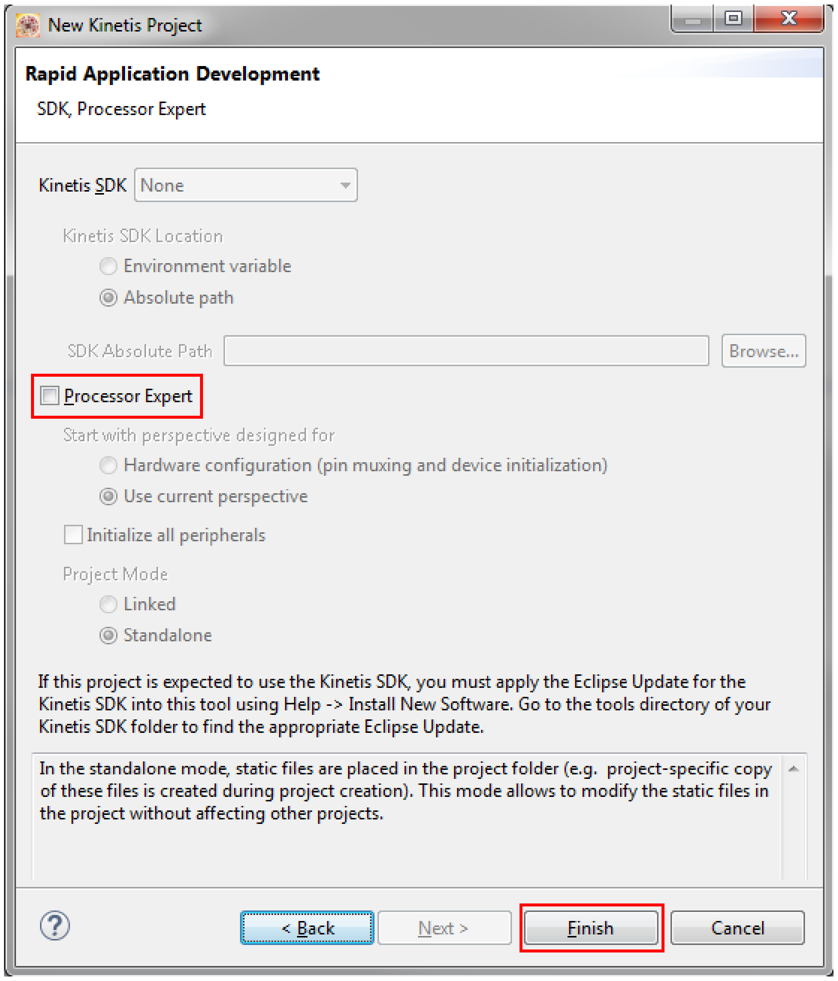
4. Choose the appropriate processor to match your development board.

K20D50M board	**K22F board**
MK20DX128xxx5 (50MHz)	MK22FN512xxx12 (120 MHz)


Processor to be used: mk20dx128


Processor to be used: k22fn512

5. Ensure that the "Processor Expert" is unticked and click **Finish**.



Kinetis SDK: None

Kinetis SDK Location
- ○ Environment variable
- ● Absolute path

SDK Absolute Path

☐ Processor Expert

Start with perspective designed for
- ○ Hardware configuration (pin muxing and device initialization)
- ● Use current perspective

☐ Initialize all peripherals

Project Mode
- ○ Linked
- ● Standalone

If this project is expected to use the Kinetis SDK, you must apply the Eclipse Update for the Kinetis SDK into this tool using Help -> Install New Software. Go to the tools directory of your Kinetis SDK folder to find the appropriate Eclipse Update.

In the standalone mode, static files are placed in the project folder (e.g. project-specific copy of these files is created during project creation). This mode allows to modify the static files in the project without affecting other projects.

Writing the code

Clocking the Port Control module

Various components of the microprocessor are not clocked by default. To use the Port Control module, it first must be connected to a clock source so that its digital electronics may operate.

You can clock the Port Control module using the following C code, placed inside your main function above any other code:

```
// Enable the clock for the port control module
SIM_SCGC5 |= 0x3e00;
```

QUESTION 3—Why wouldn't the entire microprocessor be clocked by default? What advantage is gained by allowing various modules to have their digital clock removed? (Hint: Refer to the datasheet page 17.)

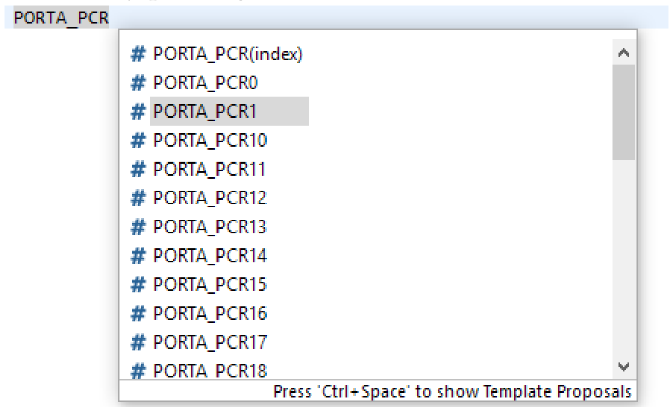
Setting up standard boolean types

A boolean type ("bool") and values true and false are not guaranteed to be part of the C language. You can use the following code at the top of your main.c file (with the other includes) if you want to use boolean values:

```
#include <stdbool.h>
```

Interacting with the registers

Kinetis Design Studio has provided you with a header file that provides access to all the CPU registers. You can browse the available names by pressing **control+space** in the editor.



```
PORTA_PCR
# PORTA_PCR(index)
# PORTA_PCR0
# PORTA_PCR1
# PORTA_PCR10
# PORTA_PCR11
# PORTA_PCR12
# PORTA_PCR13
# PORTA_PCR14
# PORTA_PCR15
# PORTA_PCR16
# PORTA_PCR17
# PORTA_PCR18
Press 'Ctrl+Space' to show Template Proposals
```

The register names are the same as those that appear in the reference manual.

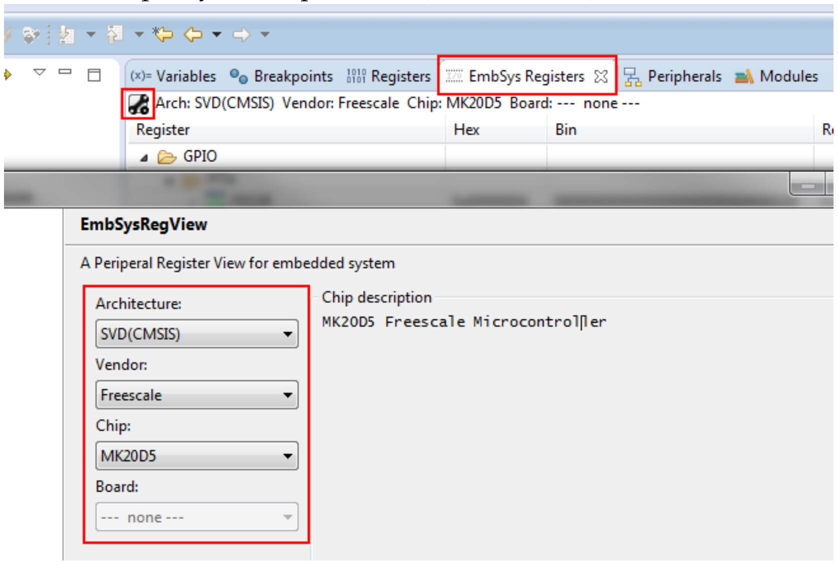
Your task

1. Write code to cycle through the LED colour combinations. You might find it helpful to write a function that sets a given LED combination to reduce repetition in your code.

2. Use the debugger to step through your program line by line so that you can see each LED turn on as you set the relevant registers.

Aside: Using the EmbSys Register inspector

Sometimes it is helpful to view and modify peripheral registers in the debugger. You can inspect registers using a KDS plugin called EmbSys. To set up EmbSys, click the spanner icon to the left of the window and specify the chip (MK20D5 or MK22FN51212).



Arch: SVD(CMSIS) Vendor: Freescale Chip: MK20D5 Board: --- none ---

EmbSysRegView

A Periperal Register View for embedded system

Architecture: SVD(CMSIS)

Vendor: Freescale

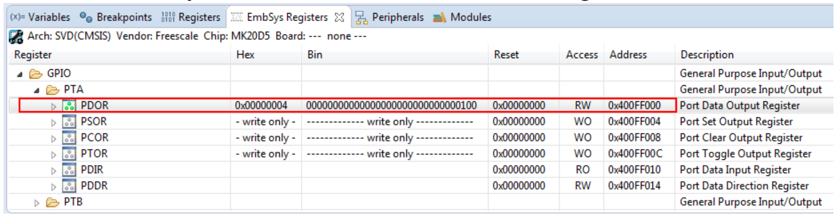
Chip: MK20D5

Board: --- none ---

Chip description

MK20D5 Freescale Microcontroller

Double click on a register to add it to the list of watched registers, which are shown with a highlighted green icon. Watched registers are updated every time the debugger advances to a new line. In the screenshot below, you can see that PTA2 is set to logic 1.



Arch: SVD(CMSIS) Vendor: Freescale Chip: MK20D5 Board: --- none ---

Register	Hex	Bin	Reset	Access	Address	Description
GPIO						General Purpose Input/Output
PTA						General Purpose Input/Output
PDOR	0x00000004	00000000000000000000000000000100	0x00000000	RW	0x400FF000	Port Data Output Register
PSOR	- write only -	------------- write only -------------	0x00000000	WO	0x400FF004	Port Set Output Register
PCOR	- write only -	------------- write only -------------	0x00000000	WO	0x400FF008	Port Clear Output Register
PTOR	- write only -	------------- write only -------------	0x00000000	WO	0x400FF00C	Port Toggle Output Register
PDIR			0x00000000	RO	0x400FF010	Port Data Input Register
PDDR			0x00000000	RW	0x400FF014	Port Data Direction Register
PTB						General Purpose Input/Output

Committing your code to GitHub

You must upload your lab work to GitHub. Open up a **Git Bash** prompt, and use:

1. **git status** to see the modified files.
2. **git add <files>** to stage the modified files.
3. **git commit** to commit the previously staged files.
4. **git push origin master** to upload your work to GitHub.

HINT—Use of version control software such as Git is an essential part of professional software development. It is a mandatory requirement of this subject. If you are unsure of how to use Git, please ask your prac tutor or lecturer.

On a personal laptop, you might prefer to use a graphical user interface to commit your code to GitHub instead of the bash prompt. This is fine; it doesn't matter which tools you use.

Assessment

To finish this lab, you must:

- Demonstrate a working board that cycles through each of the seven possible colours.
- Show your prac tutor your GitHub webpage with your Lab 3 code visible to demonstrate that you have uploaded it.