

## **CE 4DS4 - LAB 00: Introduction Lab**

Professor: Dr. Mohamed Hassan

### **Section L04**

Khaled Hassan, hassak9, 400203796  
Nikola Petrevski, petrevsn, 400198379

Due: Friday, January 26, 2024

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

Submitted by: Nikola Petrevski (petrevsn, 400198379) & Khaled Hassan (hassak9, 400203796).

## Declaration of Contributions

We aimed to split the lab's workload as evenly as possible. We worked together to perform the experiments, solve the problems and document our results.

## Experiment 2 Part A

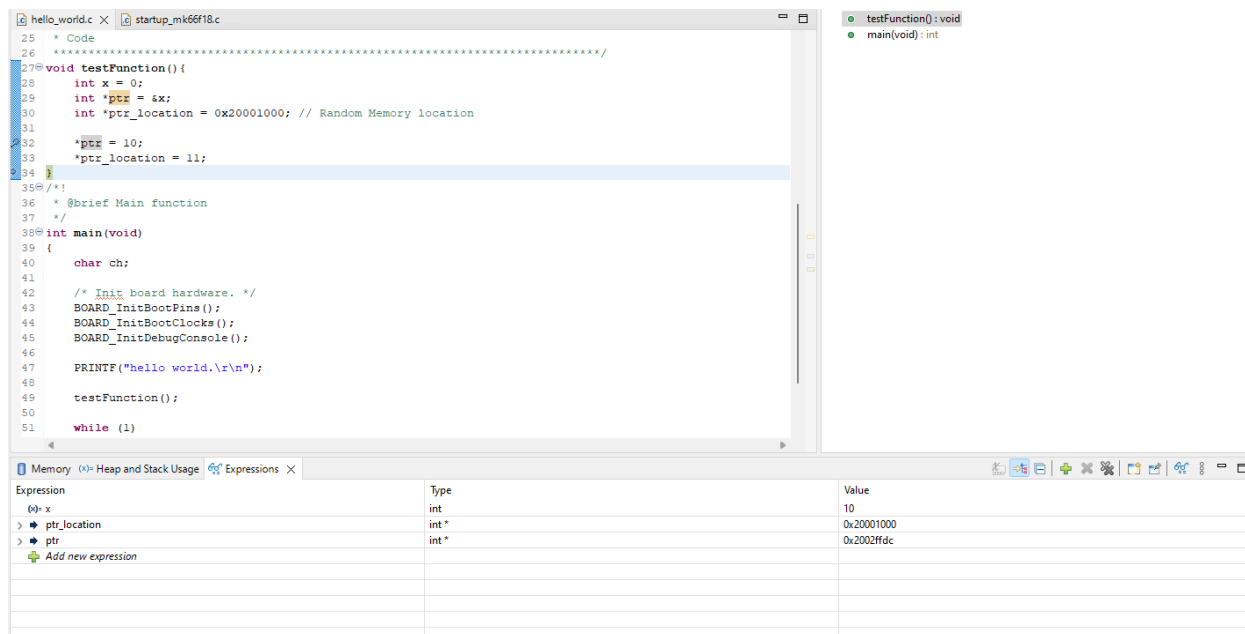


Figure 1: Watch panel showing the values of x, ptr\_location, and ptr

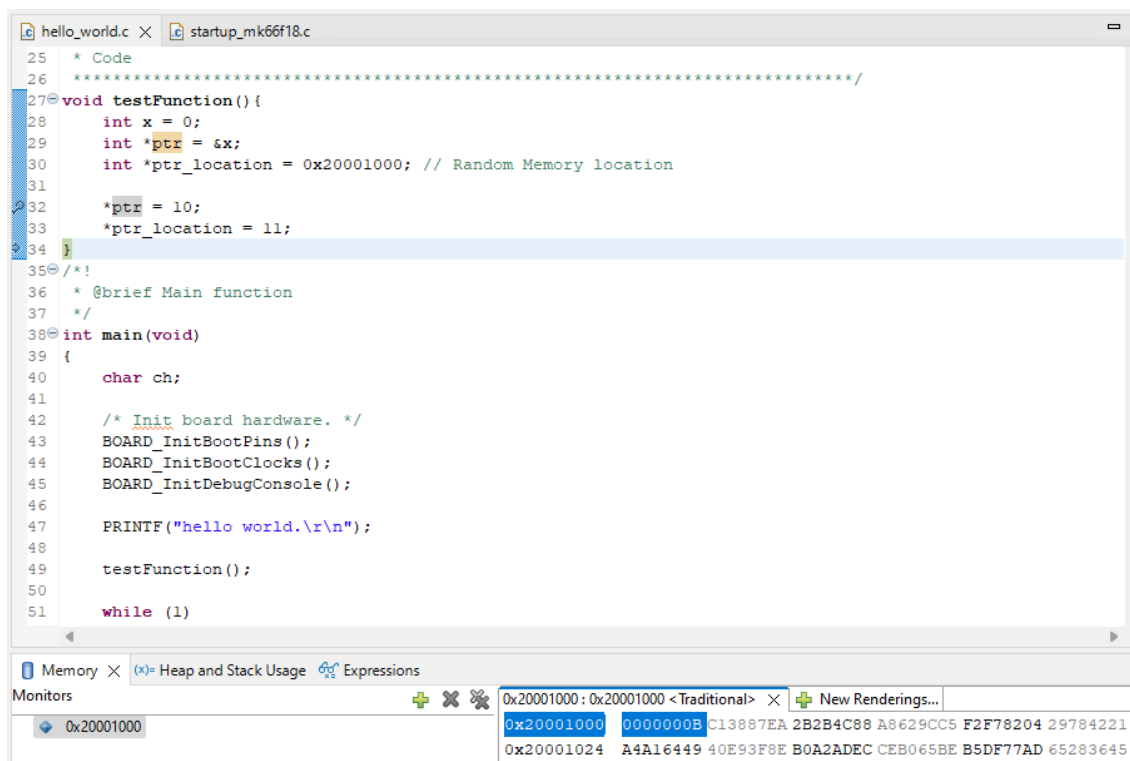
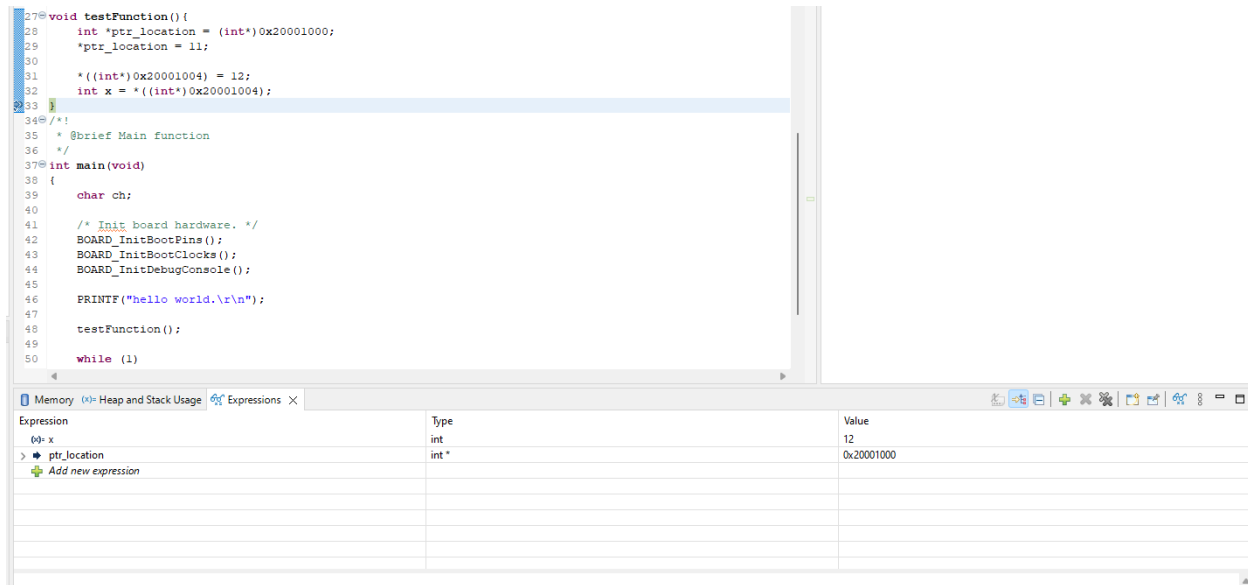


Figure 2: Memory location 0x20001000 (highlighted) showing a hex value of 0xB

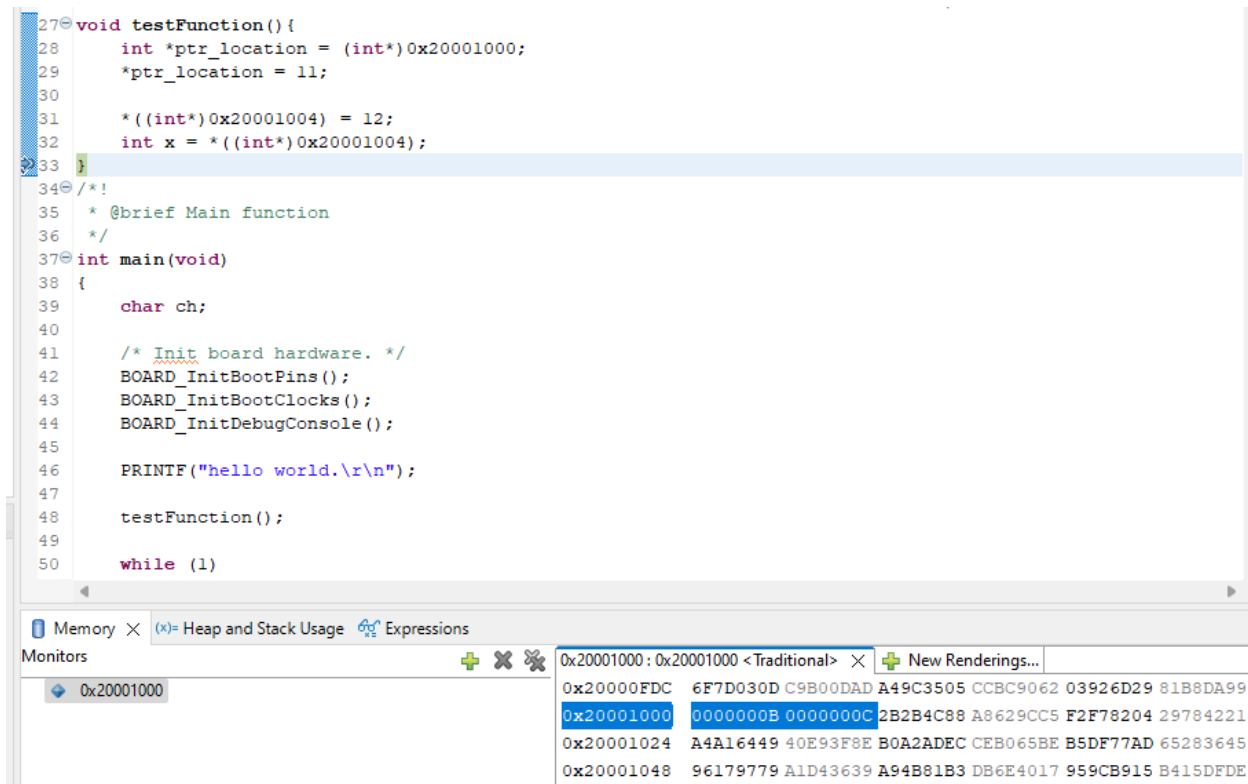
## Experiment 2 Part B



The screenshot shows an IDE with a C program. The code defines a `testFunction` that writes the value 12 to memory at address 0x20001004 and then reads it into a variable `x`. The `main` function calls `testFunction` and enters a `while (1)` loop. The Expressions panel at the bottom shows the current state of variables:

Expression	Type	Value
<code>x</code>	<code>int</code>	12
<code>ptr_location</code>	<code>int*</code>	0x20001000

Figure 3: Expression panel showing the values of x, and ptr\_location for step 2



The screenshot shows the same IDE with the Memory panel open. It displays the memory layout starting from address 0x20001000. The value at address 0x20001000 is shown as 0000000B, which is the hexadecimal representation of the decimal value 11.

Address	Value (Hex)
0x20001000	0000000B
0x20001004	A4A16449
0x20001008	96179779

Figure 4: Memory panel showing the value at address 0x20001000 for step 2

### Problem 1 - frdm\_k66f\_hello\_world

Name	Address	Size	Required Value
Loc1	0x20001000	1 Byte	0xAC
Loc2	0x20001001	4 Bytes	0xAABBCCDD
Loc3	0x20001005	2 Bytes	0xABCD
Loc4	0x20001007	4 Bytes	0xAABBCCDD

**Figure 5: Defined Memory Locations & Values**

The screenshot shows a code editor with two files: `hello_world.c` and `startup_mk66f18.c`. The `hello_world.c` file contains the following code:

```

16  * Definitions
17  *****/
18  #define INT_MEM_LOC(x) *((int*)x)
19  #define CHAR_MEM_LOC(x) *((char*)x)
20  #define SHORT_MEM_LOC(x) *((short*)x)
21  // #define ARBITRARY_LOCATION MEM_LOC(0x20001004)
22  #define Loc1 CHAR_MEM_LOC(0x20001000)
23  #define Loc2 INT_MEM_LOC(0x20001001)
24  #define Loc3 SHORT_MEM_LOC(0x20001005)
25  #define Loc4 INT_MEM_LOC(0x20001007)
26
27  *****/
28  * Prototypes
29  *****/
30  void testFunction();
31  *****/
32  * Code
33  *****/
34  void problem1() {
35      Loc1 = 0xAC;
36      Loc2 = 0xAABBCCDD;
37      Loc3 = 0xABCD;
38      Loc4 = 0xAABBCCDD;
39  }
40  /*
41  * @brief Main function
42  */

```

Below the code editor, there is a 'Memory' window showing the memory monitor. It displays the following data:

Address	Value	Radix	Column Size
0x20001000	AC DD CC BB	~i~	4 Bytes
0x20001004	AA CD AB DD	*i*	4 Bytes
0x20001008	CC BB AA 2B	i~*+	4 Bytes

**Figure 6: Code and Memory Locations defined in Problem 1**

As can be seen in figure 6 above, we used macros and `#define` to write the values defined in the lab document to their respective memory locations (Figure 5). The memory monitor above is configured according to the following settings:

- Endianness: Little,
- Cell Size: 1 Byte,
- Radix: Hexadecimal,
- Column Size: 4 Bytes.

### Problem 2 - frdmk66f\_hello\_world

With the absence of the `__attribute__((packed))` decorator, the compiler will align structure attributes in memory to a 4-byte alignment, effectively padding the space that the structure occupies in memory as necessary. For each of the following 4 structures, we determine that their size is as follows

- Struct 1: Contains a char x2 (1 Byte) and an int x1 (4 Bytes). The compiler will introduce a 3-byte gap between the locations of x2 and x1 such that struct1's total size becomes 8 Bytes. In a continuous memory space, the structure will appear as follows:

<b>x2 (char)</b>	<b>3-Byte Gap</b>			<b>x1 (int)</b>			

- Struct 2: contains a short x2 (2 Bytes) and an int x1 (4 Bytes). Similar to the above, the compiler will add 2-Byte padding to ensure alignment, and the total size is 8 Bytes. The structure will appear as:

<b>x2 (short)</b>	<b>2-Byte Gap</b>		<b>x1 (int)</b>				

- Struct 3: contains an int x1 (4 Bytes) and a short x2 (2 Bytes). The compiler still adds 2-Byte padding, such that the entire structure's size is 8 Bytes:

<b>x1 (int)</b>				<b>x2 (short)</b>		<b>2-Byte Gap</b>	

- Struct 4: contains an inner struct (size 12 Bytes), which itself contains a char x1 (1 Byte), a short x2 (2 Bytes) and an int x3 (4 Bytes). Due to padding within the inner struct, it's total size becomes 12 bytes. Finally, struct4 also contains an int x1 (4 Bytes) such that struct 4's total size is 16 Bytes:

<b>x1 (char)</b>	<b>3-Byte Gap</b>		<b>x2</b>	<b>2-Byte Gap</b>		<b>x3</b>				<b>x1</b>				

### Problem 3 - frdmk66f\_gpio\_led\_output

We repeated experiment 3, as outlined in the lab document, for the 3 LEDs named “LEDRGB\_BLUE”, “LEDRGB\_GREEN” and “LEDRGB\_RED”. From the schematics, we found that they corresponded to the following port/pin combinations: Blue (Port C Pin 8), Green (Port C Pin 9) and Red (Port D Pin 1). After initializing each LED’s appropriate port clock and setting its MUX, the pins were toggled in the main() function successfully to display the desired BLUE - GREEN - RED pattern.

### Problem 4 - frdmk66f\_gpio\_led\_output

As demonstrated in the lab, we wrote our own driver and utilized that instead of the provided one to re-do problem 3. To do this, we first studied the existing driver to understand its functionality. The GPIO\_Port structure was defined as:

```
29 typedef struct {
30     __IO uint32_t PDOR;           /**< Port Data Output Register, offset: 0x0 */
31     __IO uint32_t PSOR;           /**< Port Set Output Register, offset: 0x4 */
32     __IO uint32_t PCOR;           /**< Port Clear Output Register, offset: 0x8 */
33     __IO uint32_t PTOR;           /**< Port Toggle Output Register, offset: 0xC */
34     __IO uint32_t PDIR;           /**< Port Data Input Register, offset: 0x10 */
35     __IO uint32_t PDDR;           /**< Port Data Direction Register, offset: 0x14 */
36 } GPIO_Port;
```

Figure 7: GPIO\_Port data structure

Our driver included 3 main functions: init\_Pin, toggle\_Pin and read\_Pin. After the demo, we amended our function definitions to incorporate some of the feedback we got, namely about removing redundant pieces of code (for example, we had included the use of the GPIO\_FIT\_REG macro even though it wasn’t doing anything. Another point of feedback was to add a C and header file for the driver, not write the driver function definitions in gpio\_led\_output.c). As such, our 3 main driver functions appear as follows:

```
53 uint32_t read_Pin(GPIO_Port *base, uint32_t pin)
54 {
55     return (((uint32_t)(base->PDIR) >> pin) & 0x01UL);
56 }
57
58 void init_Pin(GPIO_Port *base, uint32_t pin, uint8_t pinDirection, uint8_t outputLogic)
59 {
60
61     GPIO_PinWrite(base, pin, outputLogic);
62     base->PDDR |= (1UL << pin);
63 }
64
65 void toggle_Pin(GPIO_Port *base, uint32_t pin){
66     toggle_Port(base, 1u << pin);
67 }
```

Figure 8: Driver Helper Functions

Finally, running the code in the gpio\_led\_output.c file yields the desired LED pattern behavior.

## Problem 5 - frdmk66f\_hello\_world\_2

To begin, we modified the `pwm_setup()` function as shown below in **Figure 9**

```
32 void pwm_setup()
33 {
34     ftm_config_t ftmInfo;
35     ftm_chnl_pwm_signal_param_t ftmParam[3];
36     // Blue C8
37     // Green C9
38     // Red D1
39
40     // D1, Red
41     ftmParam[0].chnlNumber = kFTM_Chnl_1;
42     ftmParam[0].level = kFTM_HighTrue;
43     ftmParam[0].dutyCyclePercent = 0;
44     ftmParam[0].firstEdgeDelayPercent = 0U;
45     ftmParam[0].enableComplementary = false;
46     ftmParam[0].enableDeadtime = false;
47
48     // C8, Blue
49     ftmParam[1].chnlNumber = kFTM_Chnl_4;
50     ftmParam[1].level = kFTM_HighTrue;
51     ftmParam[1].dutyCyclePercent = 0;
52     ftmParam[1].firstEdgeDelayPercent = 0U;
53     ftmParam[1].enableComplementary = false;
54     ftmParam[1].enableDeadtime = false;
55
56     // C9, Green
57     ftmParam[2].chnlNumber = kFTM_Chnl_5;
58     ftmParam[2].level = kFTM_HighTrue;
59     ftmParam[2].dutyCyclePercent = 0;
60     ftmParam[2].firstEdgeDelayPercent = 0U;
61     ftmParam[2].enableComplementary = false;
62     ftmParam[2].enableDeadtime = false;
63
64     FTM_GetDefaultConfig(&ftmInfo);
65
66     FTM_Init(FTM3, &ftmInfo);
67     FTM_SetupPwm(FTM3, &ftmParam, 3U, kFTM_EdgeAlignedPwm, 5000U, CLOCK_GetFreq(kCLOCK_BusClk));
68     FTM_StartTimer(FTM3, kFTM_SystemClock);
```

**Figure 9 : pwm\_setup() function for problem 5**

As can be seen above, we declared `ftmParam[3]` as an array of structures of type `ftm_chnl_pwm_signal_param_t` in order to be able to enable and make the necessary settings for each LED. For the `ftmParam[].chnlNumber` channel number, we referred to table 11.3.1 in the reference manual to set channels 1, 4, and 5 for ports D1 (red), C8 (green), and C9 (blue), respectively. We then passed the `ftmParam` array into the `FTM_SetupPwm()` function and modified the third argument to accept three inputs instead of one.

In the `BOARD_InitBootPins()` function, we enabled the clocks for both Ports C and D to account for all three LEDs. As well, we did three calls of the `PORT_SetPinMux()` function, one for each of the LEDs. For the third argument of the `PORT_SetPinMux()` function, we referred to table 11.3.1 in the reference manual to determine that MuxAlt4 was appropriate for the red LED at D1, while MuxAlt3 corresponded to the blue and green LEDs. These additions can be seen below in **Figure 10**.

```

CLOCK_EnableClock(kCLOCK_PortD);
CLOCK_EnableClock(kCLOCK_PortC);

/* PORTC9 (pin D7) is configured as PTC9 */
PORT_SetPinMux(PORTD, 1U, kPORT_MuxAlt4);
PORT_SetPinMux(PORTC, 8U, kPORT_MuxAlt3);
PORT_SetPinMux(PORTC, 9U, kPORT_MuxAlt3);

```

**Figure 10: Enabling the clocks and calling the PORT\_SetPinMux() function for all three LEDs in the BOARD\_InitBootPins() function**

```

72 int main(void)
73 {
74     char ch;
75     int* duty_cycle;
76     // int* RGB_values[3];
77
78     /* Init board hardware. */
79     BOARD_InitBootPins();
80     BOARD_InitBootClocks();
81     BOARD_InitDebugConsole();
82
83     pwm_setup();
84
85     printf("Please enter a 6-digit hex color scheme: ");
86
87     scanf("%x", &duty_cycle);
88
89     int val = duty_cycle;
90
91     int red = (((val >> 16) & 0xFF) * 100) / 255;
92     int green = (((val >> 8) & 0xFF) * 100) / 255;
93     int blue = ((val & 0xFF) * 100) / 255;
94
95     FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_1, kFTM_EdgeAlignedPwm, red);
96     FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_5, kFTM_EdgeAlignedPwm, green);
97     FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_4, kFTM_EdgeAlignedPwm, blue);
98     FTM_SetSoftwareTrigger(FTM3, true);

```

**Figure 11: main() function for problem 5**

The main() function for problem 5 is shown above in **figure 11**. To begin, we initialized the board hardware and set up the pwm for the three LEDs. We receive the 6-digit hex code as a single hex value on line 87 and then cast it to an integer in order to be able to perform the necessary integer division to scale the hex value for each LED to a maximum of 100% duty cycle. On lines 91-93, we extract the hex values relevant to each LED by bit shifting the corresponding values to the least significant position and then ANDing the result with 0xFF to eliminate any unneeded digits that may be left over in the more significant positions. After this, we perform a multiplication by 100 and a division by 255 in order to scale the hex values to the possible range of duty cycle values. Finally, we simply call the function to update the duty cycle for each color.