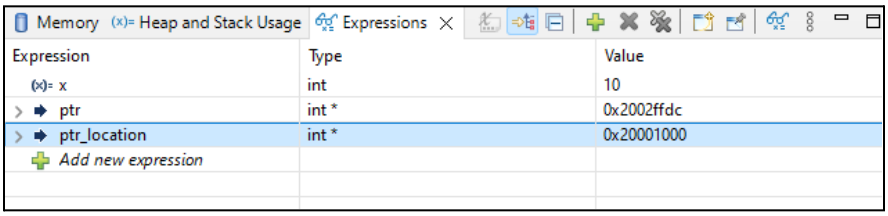
4DS4 Lab0 Report:
Introduction Lab

Johnnathan Gershkovich - 400408809
Joshua Umansky - 400234265
Shanzeb Immad - 400382928

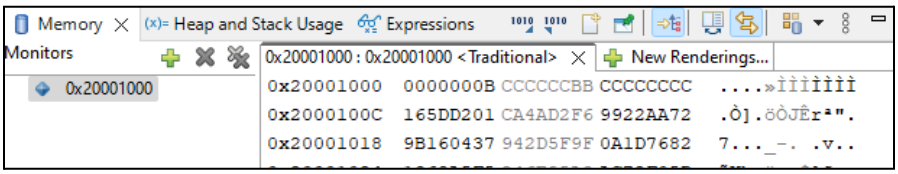
Name	Contribution
Johnnathan Gershkovich	Wrote Problem 3 and 4 writeup Aided in calculation for problem 2 Assisted writing code for Problems 1,3,4,5
Joshua Umansky	Wrote Experiment #2, Problem 5 discussion Assisted writing code for Problems 1,3,4,5
Shanzeb Immad	Write up for problem 1 and aided in calculation for problem 2 Assisted writing code for Problems 1,3,4

Experiment #2

2.A.11:

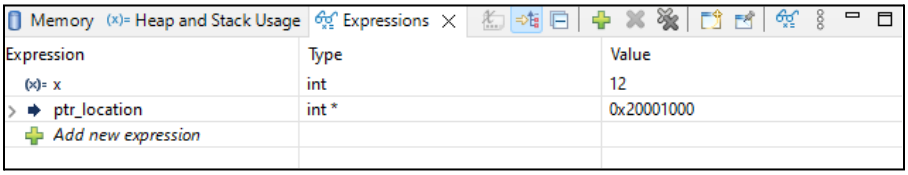


Three Variables inspected post Experiment 2A

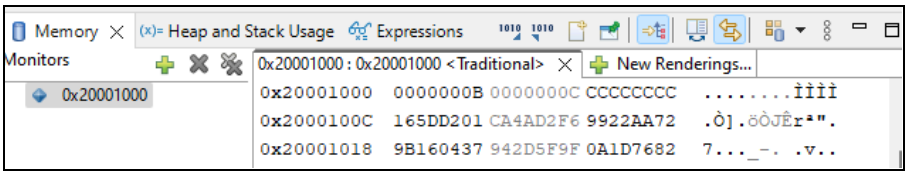


Memory heap inspected at location 0x20001000

2.B.2:



Two variables inspected post Experiment 2B



Memory heap inspected at location 0x20001000

Problem 1:

The main objective of this problem was to write down values of different data structures into specific locations. Similar to experiment 2 part B, #define were used to assign names to the addresses given. Depending on the size given by the table, the data structure was determined utilizing the table provided in *Lec2_SW.*

```
#define Loc1 *((char*)0x20001000)
#define Loc2 *((int*)0x20001001)
#define Loc3 *((short int*)0x20001005)
#define Loc4 *((int*)0x20001007)
```

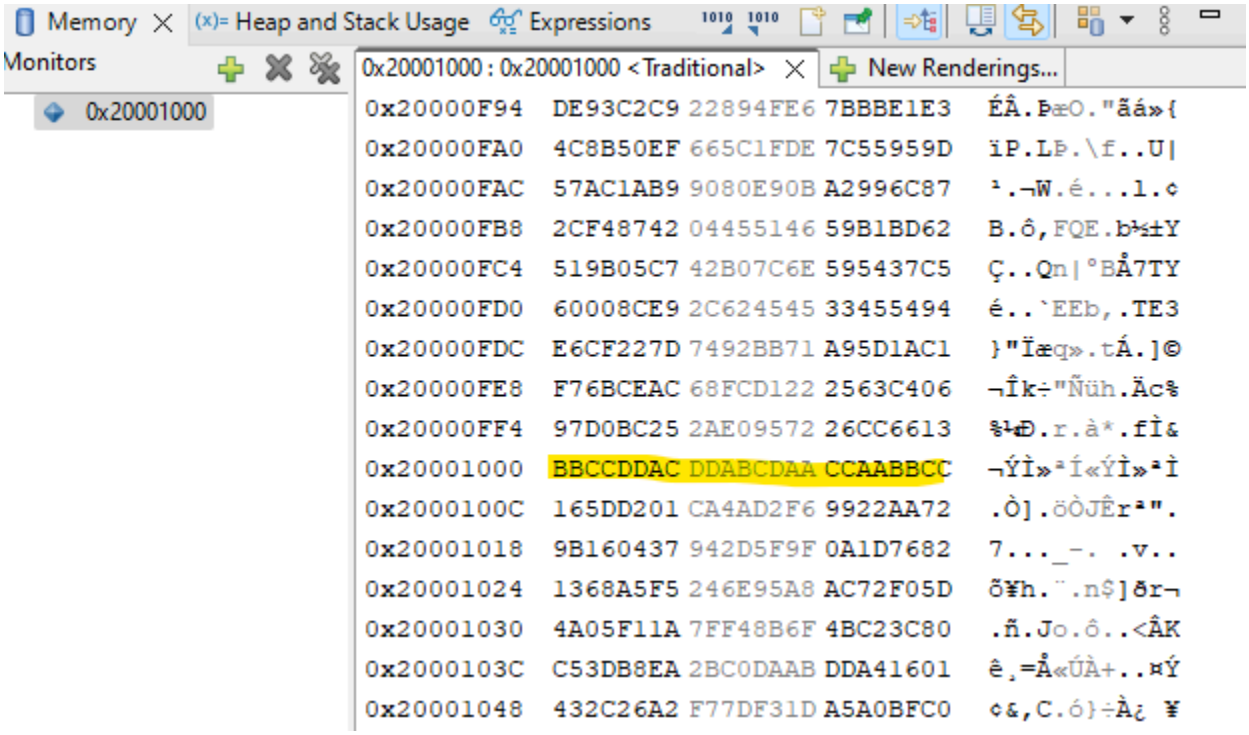
Data type	Storage	Range
char	8 bits	[-128,+127]
unsigned char	8 bits	[0,_____]
short int	16 bits	[-32768,+32767]
unsigned short int	16 bits	[0,_____]
int	16 or 32 bits	[-2^{15}, 2^{15}-1] or _____
long int	32 or 64 bits	[-2^{31}, 2^{31}-1] or _____
long long int	64 bits	[-2^{63}, 2^{63}-1]

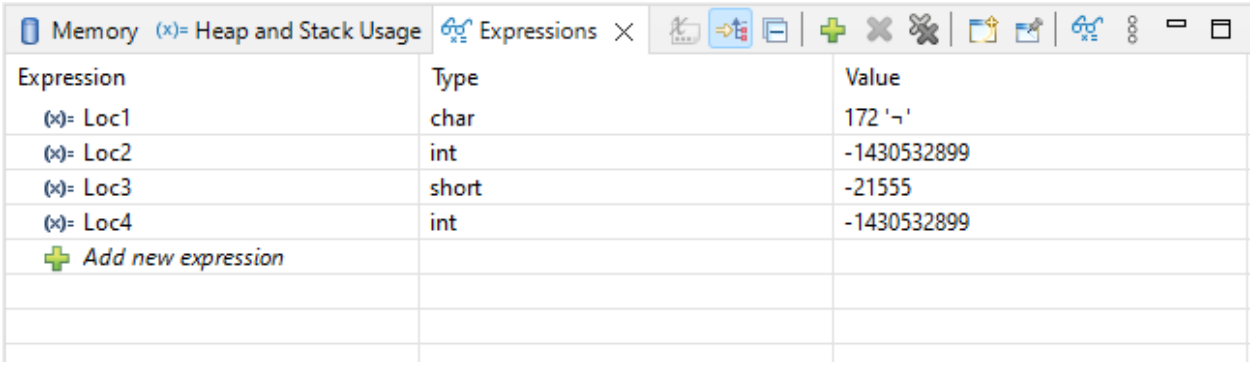
Now that the data structures and location have been defined, all that was needed to do was to assign a value at said location following the given table.

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

```
54      void Problem1(){
55              Loc1 = 0xAC;
56              Loc2 = 0xAABBCCDD;
57              Loc3 = 0xABCD;
58              Loc4 = 0xAABBCCDD;
59
60      }
```

Once this code is debugged and built, breakpoints were added and the memory location of Loc1 is viewed in the memory panel and all the LocX were added in the expression panels. Both figures below display what each resulted in. The cell size is at its default and the endianness is set to Little Endian, this can be seen with Loc1 as 0x20001001 holds 0xDD and 0x20001002 holds the next CC.





Since the memory locations are being determined and no structs are used, the data displays as “packed” where the only garbage data is the CC at byte 11.

Problem 2:

Struct1: 8 bytes, 3 padding bytes
- char: XXXX XXAA (1 byte, 3 bytes padding)
- int: BBBB BBBB (4 bytes)

Struct 2: 8 bytes, 2 padding bytes
- Short: XXXX AAAA (2 bytes, 2 bytes padding)
- Intt: BBBB BBBB (4 bytes)

Struct 3: 8 bytes, 2 padding bytes
- Int: AAAA AAAA (4 bytes)
- Short: XXXX BBBB (2 bytes, 2 bytes padding)

Struct 4: 16 bytes, 5 padding bytes
- Inner_struct: 12 bytes, 5 padding bytes
  - Char: XXXX XXAA (1 byte, 3 bytes padding)
  - Short: XXXX BBBB (2 bytes, 2 bytes padding)
  - Int: CCCC CCCC (4 bytes)
- Int: DDDD DDDD (4 bytes)

Problem #3

Referring to the MCU datasheet, the red LED is connected to pin 1 on GPIO port D and the blue and green LEDs are connected to pins 8 and 9 on GPIO port C. To control these LEDs initialization code was added to the pin_mux.c file for both ports to enable their clocks, as well as code to set each of these pins as an output.

```
/* Port C Clock Gate Control: Clock enabled */
CLOCK_EnableClock(kCLOCK_PortC);
/* Port D Clock Gate Control: Clock enabled */
CLOCK_EnableClock(kCLOCK_PortD);

//Red
PORT_SetPinMux(PORTD, 1U, kPORT_MuxAsGpio);
//Blue
PORT_SetPinMux(PORTC, 8U, kPORT_MuxAsGpio);
//green
PORT_SetPinMux(PORTC, 9U, kPORT_MuxAsGpio);
```

These ports and pins are also defined in the gpio_led_output.c file inorder to enable modification of the GPIO output values to these LEDs

```
#define BOARD_LED_GPIOC     GPIOC
#define BOARD_LED_GPIOD     GPIOD
#define BOARD_LED_GPIO_PIN_RED 1
#define BOARD_LED_GPIO_PIN_BLUE 8
#define BOARD_LED_GPIO_PIN_GREEN 9
```

In the main function in the same gpio_led_output.c file, these definitions are used to initialize the GPIO pins to allow for outputting to the LEDs.

```
/* Init output LED GPIO. */
GPIO_PinInit(BOARD_LED_GPIOC, BOARD_LED_GPIO_PIN_BLUE, &led_config);
GPIO_PinInit(BOARD_LED_GPIOC, BOARD_LED_GPIO_PIN_GREEN, &led_config);
GPIO_PinInit(BOARD_LED_GPIOD, BOARD_LED_GPIO_PIN_RED, &led_config);
```

Finally, in an unending loop, each LED is toggled on, and then off, sequentially in the order, Blue, Green and then Red. A small delay is inserted between each action.

```
while (1)
{
```

```
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOC, 1u << BOARD_LED_GPIO_PIN_BLUE);
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOC, 1u << BOARD_LED_GPIO_PIN_BLUE);
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOC, 1u << BOARD_LED_GPIO_PIN_GREEN);
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOC, 1u << BOARD_LED_GPIO_PIN_GREEN);
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOD, 1u << BOARD_LED_GPIO_PIN_RED);
    delay();
    GPIO_PortToggle(BOARD_LED_GPIOD, 1u << BOARD_LED_GPIO_PIN_RED);
}
```

Problem #4

In order to implement a custom driver to output to the RGB LEDs connected to GPIO ports a struct was made with 4 bytes set aside each of the 6 GPIO registers. This along with the three basic function prototypes were defined in the *led_driver.h* file.

```
typedef struct{
    int GPIOx_PDOR;
    int GPIOx_PSOR;
    int GPIOx_PCOR;
    int GPIOx_PTOR;
    int GPIOx_PDIR;
    int GPIOx_PDDR;
}GPIO_Struct;

//Helper functions protoypes
void Init(GPIO_Struct*, char);
void Clear(GPIO_Struct*);
void Toggle(GPIO_Struct*, uint32_t mask);
```

The corresponding function code was written in the file *led_driver.c*. The *Init* function initializes all of the registers except for PDDR to all 0s as this is the reset value defined in the documentation. The PDDR register is set to all 0s except for the 1 bit if the port is *d* or the 8 and 9 bits if the port is *c*. This sets the corresponding LED pins directions to output for their corresponding ports. The *Clear* function sets the PCOR register to all 1s which will reset all of the pins to their default state. The *Toggle* function sets the given bits which are 1 in the bit mask to 1 in the toggle register PTOR.

```
void Init(GPIO_Struct* gpio, char c){
    gpio->GPIOx_PDOR = 0x00000000;
    gpio->GPIOx_PCOR = 0x00000000;
    gpio->GPIOx_PDIR = 0x00000000;
    gpio->GPIOx_PSOR = 0x00000000;
    gpio->GPIOx_PTOR = 0x00000000;
    if(c == 'c'){
        gpio->GPIOx_PDDR = 0x00000300;//0000 0011 0000 0000
    }
    else if(c == 'd'){
        gpio->GPIOx_PDDR = 0x00000002;
    }
}

void Clear(GPIO_Struct* gpio){
    //update PCOR
    gpio->GPIOx_PCOR = 0x11111111;
}

void Toggle(GPIO_Struct* gpio, uint32_t mask){
    //update PTOR
    gpio->GPIOx_PTOR = mask;
}
```

Lastly, in the *gpio_led_output.c* file, two gpio structs are created, at memory locations of the two gpio ports, C and D, as provided in the K66 reference manual. The *Init* function is then called for both structs, passing in the struct and its corresponding port letter in order to initialize the correct pin out directions for both ports. The same process to toggle on and off the LEDs is used as in problem #3, with the given GPIO toggle function being replaced with the newly implemented toggle function that takes in the port, and the pin(s) on the port, in the form of a bit mask, to toggle

```
GPIO_Struct *gpioc = (GPIO_Struct*)0x400FF080;
GPIO_Struct *gpiod = (GPIO_Struct*)0x400FF0C0;
char d ='d';
char c = 'c';
Init(gpioc, c);
Init(gpiod, d);

while (1)
{
    delay();
    Toggle(gpioc, 1u << 8);
    delay();
    Toggle(gpioc, 1u << 8);
    delay();
    Toggle(gpioc, 1u << 9);
    delay();
    Toggle(gpioc, 1u << 9);
    delay();
    Toggle(gpiod, 1u << 1);
    delay();
    Toggle(gpiod, 1u << 1);
    delay();
}
```

Problem #5

After editing the BOARD_BootClockRun function to allow for accurate timing which was required for PWM usage, as specified by Experiment 4, we began to work towards PWM control for the three LEDs that would allow for variable brightness settings. We changed the pin_mux definitions for the three LED pins;

```
// BLUE
    PORT_SetPinMux(PORTC, 8U, kPORT_MuxAlt3);
// GREEN
PORT_SetPinMux(PORTC, 9U, kPORT_MuxAlt3);
// RED
PORT_SetPinMux(PORTD, 1U, kPORT_MuxAlt4);
```

By using a specific mux alternate function, the LEDs can be driven through PWM duty cycles instead of direct GPIO output, as each pin at which an LED is connected to has an alternative function of PWM output. Using this alternate PWM function, varied brightness can be outputted using different duty cycles instead of binary GPIO outputs.

```
void pwm_setup()
{
	ftm_config_t ftmInfo;
	ftm_chnl_pwm_signal_param_t ftmParam[3];

	ftmParam[0].chnlNumber = kFTM_Chnl_1;
	ftmParam[1].chnlNumber = kFTM_Chnl_4;
	ftmParam[2].chnlNumber = kFTM_Chnl_5;
	for (int i = 0; i < 3; i++) {
		ftmParam[i].level = kFTM_HighTrue;
		ftmParam[i].dutyCyclePercent = 0;
		ftmParam[i].firstEdgeDelayPercent = 0U;
		ftmParam[i].enableComplementary = false;
		ftmParam[i].enableDeadtime = false;
	}

	FTM_GetDefaultConfig(&ftmInfo);

	FTM_Init(FTM3, &ftmInfo);
	FTM_SetupPwm(FTM3, &ftmParam, 3U, kFTM_EdgeAlignedPwm, 5000U, CLOCK_GetFreq(kCLOCK_BusClk));
	FTM_StartTimer(FTM3, kFTM_SystemClock);
}
```

In the pwm_setup function, three channels were set up, and passed to prebuilt functions (FTM_SetupPwm), one for each individual LED, with the chnlNumber different for each of the struct's created. Continuing in the hello_world.c file, the main function is changed in order to facilitate user input, and then subsequently enabling the LEDs accordingly. Using the "*x*" input flag for scanf and limiting each input to two 2 hex characters, a 6 hex input is automatically parsed into three 2 hex variables. We chose to manipulate the input directly into a type-cast int variable for each independent colour value, which is then used as the duty cycle for each individual channel created earlier.

```
pwm_setup();

printf("Enter html color value: \n");

int red, green, blue;
scanf("%2x%2x%2x", &red, &green, &blue);

int red_val = (red*100)/255;
int green_val = (green*100)/255;
int blue_val = (blue*100)/255;

//PRINTF("hello world.\r\n");
int duty_cycle=0;
FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_1, kFTM_EdgeAlignedPwm, red_val);
FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_4, kFTM_EdgeAlignedPwm, blue_val);
FTM_UpdatePwmDutycycle(FTM3, kFTM_Chnl_5, kFTM_EdgeAlignedPwm, green_val);
FTM_SetSoftwareTrigger(FTM3, true);
```