Lab Assignment 2

Jose Monroy Villalobos

San José State University
Department of Computer Engineering
CMPE 146-01, Real-Time Embedded System Co-Design, Fall 2024

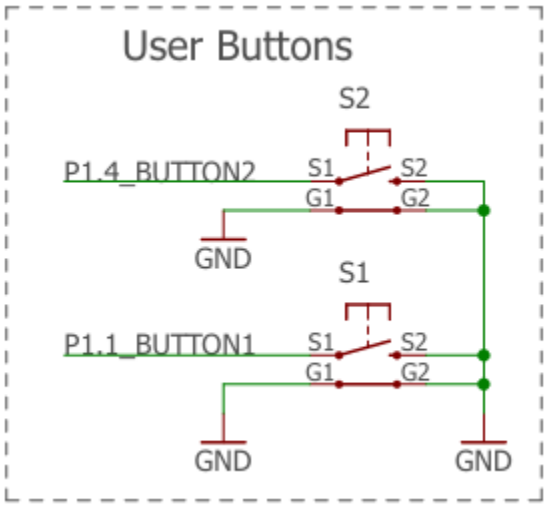
Exercise 1.1 DriverLib

Code: *Check appendix for full code*

```
/* Configuring Button 1,S1, as input*/
MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);

/* Configuring LEDS as output */
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);

/* Setting LEDS to low state */
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

/* Variable to keep track of previous state */

bool unpressed;

/* CONTROL LOGIC: When button is unpressed it reports back a 1, and when it is pressed it reports 0
 * I am using a bool to control and keep track if the previous state was unpressed
 * it starts as true then when pressed it goes to false and the light is toggled
 * it can only go back to true if the button is released so it solves issue I had that
 * if I held the button pressed it would toggle back and forth because my orignal logic
 * was that if the button was pressed than to toggle led issue with that logic was that
 * the button is polled so it checks and when held it detects each poll as a press is my
 * assumption.
 * */
unpressed =1;

while (1)
{
   printf("input: %i    Previous: %i\n", MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1), unpressed);
```

```
    if(MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1)==0){
        if(unpressed){
            unpressed = false;
            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN2);
        }
    }
    else
    {
        unpressed = true;
    }
}
 }
```

There should be a difference in power consumption of the LaunchPad between the two states of S1 (pressed or not pressed). Which state will consume more power? Justify your answer in reference to the LaunchPad's circuitry.



I think that the pressed state consumes more power because when it is pressed the circuit is closed and current flows. When it is unpressed the circuit is open and no current flows, which is I think the pressed state consumes more power.

Video Link:

https://youtube.com/shorts/0nYjyTgTjTI?feature=share

Exercise 1.2 Direct Port Access

The Port Registers Address is 0x4004C00

Port 1 Input offset is 000h

The pointer for Button 1 has the following behaviour:

When unpressed *buttonPTR = 0xCE = 11001110

When Pressed *buttonPTR = 0xCC = 11001100

This indicates that when the button 1 is pressed this bit changes, 1 for unpressed and 0 for pressed

Port 2 output offset is 003h

The pointer for the blue LED has the following behaviour:

When off *LedPTR = 0xC8 = 11001000

When On *LedPTR = 0xCC = 11001100

This indicates that when the Blue led is on this bit changes, 1 for on and 0 for off

Relevant Code:

```
  while (1)
  {
    /* Code used to see raw data and better understand the mechanic
     printf("\nbuttonPTR %x    ", buttonPTR);
     printf(" *buttonPTR %x      ", *buttonPTR);
     printf(" &buttonPTR %x\n", &buttonPTR);

     printf("LedPTR %x    ", LedPTR);
     printf(" *LedPTR %x      ", *LedPTR);
     printf(" &LedPTR %x\n\n", &LedPTR);
*/
     if((*buttonPTR & 0x02) == 0)
     {
       if(unpressed)
       {
```

```
            unpressed = false;
            *LedPTR = *LedPTR ^ 0x4;
          }
        }

        else
        {
          unpressed = true;
        }
      }
}
```

Follows similar logic to previous exercise. It checks the bit that represents button 1 state v ia and AND operation. If the bit that indicates button 1 is Pressed, it checks if the previous state was unpressed. If this is true it sets the flag to false and toggles the bit that controls the blue led using XOR operation. The else statement is to reset the unpressed bool back to true.

Video Link:

https://youtube.com/shorts/bQyOKDOPnbM?feature=share

Exercise 1.3 Bit-banding Access

alias_addr = (port_addr – peripheral_region_addr) * 32 + bit_position * 4 + alias_region_addr

For BLUE LED

= (0x40004C03- 0x40000000)*32 + 2 * 4 + 0x42000000

= (0x4C03)*32 + 2 * 4 + 0x42000000

= (0x4C03)*32 + 8 + 0x42000000

= (0x4C03)*0x20 + 0x42000008

= 0x96060 + 0x42000008

0x42096068

For Button 1

= (0x40004C00- 0x40000000)*0x1F + 1 * 4 + 0x42000000

= (0x4C00)*0x20 + 0x42000004

= 0x42098004

Debug Console Outputs:

Button 1 bit-band alias address: 42098004

Blue LED bit-band alias address: 42098068

Relevant code:

```
 bool unpressed;
 unpressed =1;



  while (1)
  {
//Check button1 state
    if(*buttonBitBand == 0)
    {
      if(unpressed)
      {
        unpressed = false;
//Toggle bit
          *ledBitBand = !(*ledBitBand);
      }
    }

    else
    {
      unpressed = true;
    }
  }
}
```

Explain the operations in the control loop. What are the advantages of the control method in this exercise, compared to the one in the previous exercise?

This method is much simpler in terms of the complexity required to set, toggle and read. There is no XOR or AND operations required all that is need to check the bit and to toggle the led bit. I would assume this one would be the better performing code.

Exercise 2 Time Measurement

Debug Console Output:

Duration of press: 164.983 ms

Duration of press: 107.596 ms

Duration of press: 3257.463 ms

Duration of press: 61.583 ms

Relevant code for timing:

```
/* Variable to keep track of timing */
uint32_t t0 = 0, t1 = 0;
uint32_t press_duration_cycles = 0;
float press_duration_ms = 0;

while (1)
{

  if(MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1)==0){

    if(unpressed){
      /* Get time of when pressed */
      t1 = MAP_Timer32_getValue(TIMER32_0_BASE);
      unpressed = false;
      pressed = true;
      MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN2);
    }
  }
  else
  {
    /* Get time of when unpressed */
    t0 = MAP_Timer32_getValue(TIMER32_0_BASE);

    /* checks if button was released */
    if(pressed){
      pressed = false;
```

```
            /* Calculate time in ms and display it*/
                press_duration_cycles = t1 - t0;
                press_duration_ms = (press_duration_cycles / (float)mclk_freq) * 1000;
                printf("Duration of press: %.3f ms\n", press_duration_ms);
            }

            unpressed = true;
        }
    }
```

Exercise 3 Frequency Measurement

Debug Console outputs:

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400063, Duration 466.688 ms, Frequency: 1.071 Hz

Counter Difference: 1400062, Duration 466.687 ms, Frequency: 1.071 Hz

Relevant code;

```
/* Variables to keep track of timing */
    uint32_t t0 = 0, t1 = 0;
    uint32_t Counter_Difference= 0;
    float difference_ms = 0.0f;
    float frequency_hz = 0.0f;

    while (1)
    {

        /* Record the timer value before the delay loop */
        t0 = MAP_Timer32_getValue(TIMER32_0_BASE);

        /* Delay Loop */
        for(ii = 0; ii < 100000; ii++)
        {

        }

        /* Toggle the LED */
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

        /* Record the timer value after the delay loop */
        t1 = MAP_Timer32_getValue(TIMER32_0_BASE);

        Counter_Difference = t0 - t1;


        /* Calculate duration in milliseconds */
        difference_ms = ((float)Counter_Difference / (float)mclk_freq) * 1000.0f;
```

```
        /* Since one loop iteration toggles the LED once, the full period is twice the duration */
        float period_ms = difference_ms * 2.0f;

        /* Calculate blinking frequency in Hz */
        frequency_hz = 1000.0f / period_ms;

        /* Print the results */
        printf("Counter Difference: %i, Duration %.3f ms, Frequency: %.3f Hz\n",Counter_Difference, difference_ms, frequency_hz);
    }
}
```

This method of measuring does carry some uncertainty because we are depending on the hardware and software to mark and compute the time which takes some time to assign the value. I would say the act of tracking and measuring makes each loop take a tiny bit longer and so the frequency cannot be measured accurately. The was a counter difference seen when looking at the console which was a .001ms difference between counts.

Appendix:

Exercise 1.1 DriverLib:

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
```

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

//![Simple GPIO Config]
int main(void)
 {
  volatile uint32_t ii;

  /* Halting the Watchdog */
  MAP_WDT_A_holdTimer();

  /* Configuring Button 1,S1, as input*/
  MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);

  /* Configuring LEDS as output */
  MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
  MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
  MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);

  /* Setting LEDS to low state */
  MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
  MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
  MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

  /* Variable to keep track of previous state */

  bool unpressed;

  /* When button is unpressed it reports back a 1, and when it is pressed it reports 0
   * I am using a bool to control and keep track if the previous state was unpressed
   * it starts as true then when pressed it goes to false and the light is toggled
   * it can only go back to true if the button is released so it solves issue I had that
   * if I held the button pressed it would toggle back and forth because my orignal logic
   * was that if the button was pressed than to toggle led issue with that logic was that
   * the button is polled so it checks and when held it detects each poll as a press is my
   * assumption.
   * */
  unpressed =1;
```

```
    while (1)
    {
        printf("input: %i    Previous: %i\n", MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1), unpressed);



        if(MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1)==0){
            if(unpressed){
                unpressed = false;
                MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN2);
            }
        }
        else
        {
            unpressed = true;
        }
}
  }
```

Exercise 1.2 Direct Port Access:

```
/* DriverLib Includes */
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
/* Port Registers address 0x40004C00
 * Port 1 Input   P1IN 000h
 * Port 2 Output  P2OUT 003h
 */
#define Button_address 0x40004C00
#define LED_address 0x40004C03
```

```
//![Simple GPIO Config]
int main(void)
  {
    volatile uint32_t ii;

    /* Halting the Watchdog */
    MAP_WDT_A_holdTimer();

    /* Pointers for LED and Button 1 */
    uint8_t *LedPTR = (uint8_t*)LED_address;
    uint8_t *buttonPTR = (uint8_t*)Button_address;

    /* Configuring Button 1,S1, as input*/
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);

    /* Configuring LEDS as output */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);

    /* Setting LEDS to low state */
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

    /* Variable to keep track of previous state */

    bool unpressed;

    /* When button is unpressed it reports back a 1, and when it is pressed it reports 0
     * I am using a bool to control and keep track if the previous state was unpressed
     * it starts as true then when pressed it goes to false and the light is toggled
     * it can only go back to true if the button is released so it solves issue I had that
     * if I held the button pressed it would toggle back and forth because my orignal logic
     * was that if the button was pressed than to toggle led issue with that logic was that
     * the button is polled so it checks and when held it detects each poll as a press is my
     * assumption.
```

```
    * */
   unpressed =1;



   while (1)
   {
     /* Code used to see raw data and better understand the mechanic
      printf("\nbuttonPTR %x    ", buttonPTR);
      printf(" *buttonPTR %x      ", *buttonPTR);
      printf(" &buttonPTR %x\n", &buttonPTR);

      printf("LedPTR %x    ", LedPTR);
      printf(" *LedPTR %x      ", *LedPTR);
      printf(" &LedPTR %x\n\n", &LedPTR);
*/
      if((*buttonPTR & 0x02) == 0)
      {
         if(unpressed)
         {
            unpressed = false;
            *LedPTR = *LedPTR ^ 0x4;
         }
      }

      else
      {
         unpressed = true;
      }
   }
}
```

Exercise 1.3 Bit-banding Access Code:

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
```

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

#define Button1_Port_Address        0x40004C00
#define BlueLed_Port_Address        0x40004C03
#define Peripheral_Start_Address    0x40000000
#define Alias_Start_Address         0x42000000

#define Button1_Bit_Position        1
#define Blue_Led_Bit_Position       2

#define BUTTON_BIT_BAND_ALIAS ((Button1_Port_Address - Peripheral_Start_Address) * 32 + Button1_Bit_Position * 4 + Alias_Start_Address)
#define LED_BIT_BAND_ALIAS    ((BlueLed_Port_Address - Peripheral_Start_Address) * 32 + Blue_Led_Bit_Position * 4 + Alias_Start_Address)

//![Simple GPIO Config]
int main(void)
 {
   volatile uint32_t ii;

   /* Halting the Watchdog */
   MAP_WDT_A_holdTimer();

   /* Pointers for LED and Button 1 */
   uint8_t *buttonBitBand = (volatile uint8_t *)BUTTON_BIT_BAND_ALIAS;
   uint8_t *ledBitBand    = (volatile uint8_t *)LED_BIT_BAND_ALIAS;

   /* Configuring Button 1,S1, as input*/
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);

   /* Configuring LEDS as output */
   MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
   MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
   MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);

   /* Setting LEDS to low state */
   MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
   MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
```

```
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

    /* Print the computed alias addresses */
       printf("Button 1 bit-band alias address: %x\n", BUTTON_BIT_BAND_ALIAS);
       printf("Blue LED bit-band alias address: %x\n", LED_BIT_BAND_ALIAS);

    /* Variable to keep track of previous state */

    bool unpressed;

    /* When button is unpressed it reports back a 1, and when it is pressed it reports 0
     * I am using a bool to control and keep track if the previous state was unpressed
     * it starts as true then when pressed it goes to false and the light is toggled
     * it can only go back to true if the button is released so it solves issue I had that
     * if I held the button pressed it would toggle back and forth because my orignal logic
     * was that if the button was pressed than to toggle led issue with that logic was that
     * the button is polled so it checks and when held it detects each poll as a press is my
     * assumption.
     * */
    unpressed =1;



    while (1)
    {
      /*Testing deactivated
       printf("\nbuttonPTR %x    ", buttonPTR);
       printf(" *buttonPTR %x      ", *buttonPTR);
       printf(" &buttonPTR %x\n", &buttonPTR);

       printf("LedPTR %x    ", LedPTR);
       printf(" *LedPTR %x      ", *LedPTR);
       printf(" &LedPTR %x\n\n", &LedPTR);
*/
       if(*buttonBitBand == 0)
       {
          if(unpressed)
          {
             unpressed = false;
```

```
            *ledBitBand = !(*ledBitBand);
        }
    }

    else
    {
        unpressed = true;
    }
  }
}
//![Simple GPIO Config]
```

Exercise 2 Time Measurement code:

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>

//![Simple GPIO Config]
int main(void)
 {
  MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1, TIMER32_32BIT,
  TIMER32_FREE_RUN_MODE);
  MAP_Timer32_startTimer(TIMER32_0_BASE, 0);
  printf("%u\n", MAP_CS_getMCLK());

  uint32_t mclk_freq = MAP_CS_getMCLK();
  volatile uint32_t ii;

  /* Halting the Watchdog */
  MAP_WDT_A_holdTimer();

  /* Configuring Button 1,S1, as input*/
  MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
```

```
/* Configuring LEDS as output */
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);

/* Setting LEDS to low state */
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

/* Variable to keep track of previous state */

bool unpressed;
bool pressed;

unpressed =1;
pressed =0;

/* Variable to keep track of timing */
uint32_t t0 = 0, t1 = 0;
uint32_t press_duration_cycles = 0;
float press_duration_ms = 0;

while (1)
{

  if(MAP_GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1)==0){

    if(unpressed){
      /* Get time of when pressed */
      t1 = MAP_Timer32_getValue(TIMER32_0_BASE);
      unpressed = false;
      pressed = true;
      MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN2);
    }
  }
 else
  {
```

```
        /* Get time of when unpressed */
        t0 = MAP_Timer32_getValue(TIMER32_0_BASE);

        /* checks if button was released */
        if(pressed){
            pressed = false;

            /* Calculate time in ms and display it*/
            press_duration_cycles = t1 - t0;
            press_duration_ms = (press_duration_cycles / (float)mclk_freq) * 1000;
            printf("Duration of press: %.3f ms\n", press_duration_ms);
        }

        unpressed = true;
    }
}
 }
```

Exercise 3 Frequency Measurement

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>

//![Simple GPIO Config]
int main(void)
{
    MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1, TIMER32_32BIT,
    TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_0_BASE, 0);
    printf("%u\n", MAP_CS_getMCLK());

    uint32_t mclk_freq = MAP_CS_getMCLK();
    volatile uint32_t ii;
```

```
/* Halting the Watchdog */
MAP_WDT_A_holdTimer();

/* Configuring P1.0 as output */
MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);



/* Variables to keep track of timing */
uint32_t t0 = 0, t1 = 0;
uint32_t Counter_Difference= 0;
float difference_ms = 0.0f;
float frequency_hz = 0.0f;

while (1)
{

   /* Record the timer value before the delay loop */
   t0 = MAP_Timer32_getValue(TIMER32_0_BASE);

   /* Delay Loop */
   for(ii = 0; ii < 100000; ii++)
   {


   }

   /* Toggle the LED */
   MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

   /* Record the timer value after the delay loop */
   t1 = MAP_Timer32_getValue(TIMER32_0_BASE);

   Counter_Difference = t0 - t1;



   /* Calculate duration in milliseconds */
   difference_ms = ((float)Counter_Difference / (float)mclk_freq) * 1000.0f;

   /* Since one loop iteration toggles the LED once, the full period is twice the duration */
```

```
        float period_ms = difference_ms * 2.0f;

        /* Calculate blinking frequency in Hz */
        frequency_hz = 1000.0f / period_ms;

        /* Print the results */
        printf("Counter Difference: %i, Duration %.3f ms, Frequency: %.3f Hz\n",Counter_Difference, difference_ms, frequency_hz);
    }
}
```