Lab Assignment 5

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 Delay Function implementation:

Delay_ms Function Code:

```
void delay_ms(uint32_t count)
{
    uint32_t start = MAP_Timer32_getValue(TIMER32_0_BASE); // 16-bit value
    uint64_t duration_cycles = (uint64_t)count * mclk_freq_div_1000;
    uint64_t elapsed_cycles = 0;
    uint32_t current;


    while (elapsed_cycles < duration_cycles)
    {
         current = MAP_Timer32_getValue(TIMER32_0_BASE); // 16-bit value

        if (start >= current)
        {
            elapsed_cycles += start - current;
        }
        else
        {
            // Handle 16-bit wrap-around
            elapsed_cycles += 0xFFFF - current + start + 1;
        }

        start = current;
    }
}
```

The way this function works is that the number of cycles that must elapse for the delay to happen is calculated and then the while loop continuously measures the cycles elapsed and adds it to a variable to keep track of how many cycles have elapsed. I optimized the code and found that making the processor frequency and the divide by 1000 one variable that was computed outside the function had a huge impact on lowering the error.

Debug Console Outputs:

```
[CORTEX_M4_0] MCLK Frequency: 3000000 Hz
Target: 5000 ms, Difference: 15000170 cycles, Measured time: 5000056 us, Error: 56 us
Target: 2000 ms, Difference: 6000140 cycles, Measured time: 2000046 us, Error: 46 us
Target: 1000 ms, Difference: 3000130 cycles, Measured time: 1000043 us, Error: 43 us
Target: 50 ms, Difference: 150140 cycles, Measured time: 50046 us, Error: 46 us
Target: 20 ms, Difference: 60115 cycles, Measured time: 20038 us, Error: 38 us
```

```
Target: 10 ms, Difference: 30150 cycles, Measured time: 10050 us, Error: 50 us
Target: 5 ms, Difference: 15135 cycles, Measured time: 5045 us, Error: 45 us
Target: 2 ms, Difference: 6165 cycles, Measured time: 2055 us, Error: 55 us
Target: 1 ms, Difference: 3110 cycles, Measured time: 1036 us, Error: 36 us
Target: 0 ms, Difference: 122 cycles, Measured time: 40 us, Error: 40 us
```

Exercise 1.2 Timing control:

Debug Console Outputs:

```
[CORTEX_M4_0] MCLK Frequency: 3000000 Hz
delay_in_ms: 1250 ms
```

The logic to getting the light to turn on and off at .4hz was to have a while loop that would toggle led and then delay for half of the time necessary for .4hz frequency and the loop and toggle the light and then do the other half of the time. The time was calculated outside the while loop and passed to the delay_ms function. Here is the math:

Frequency = .4 Hz   Time (s) = $\frac{1}{Frequency\ (hz)}$

Time (ms) = $\frac{1000}{Frequency\ (hz)}$ = 2500 ms

So the delay must be half of 2500 ms which results in 1250 ms being the delay required as shown in the debug output.

The measurement method I used was to start the stop watch when I saw the led turn on and then the next time it turned on I started counting 1 to 10. If my code is correct this

should take 25 seconds, and this is what was observed. Here is a screen shot from excel that has the necessary values:

Measurements	Time (s) for 10 blinks	Frequency (Hz)			
1	24.96	0.400641026			
2	25.24	0.396196513			
3	24.83	0.402738623	Average Frequency =		0.399859
			Error =	0.03532	

Equations used:

Average Frequency= (Freq1 + Freq2 + Freq3)/ 3

Error = |(Average Frequency- .4)/.4| * 100

The error was .03%, which is well below the 1%.

Exercise 2 GPIO oscillator:

Debug Console Outputs:

```
[CORTEX_M4_0] MCLK Frequency: 3000000 Hz
Sampled Data:
1011011010010110100101101001011010010110110100101101001011010010110110101101001011010
010110100101101
Sampling Rate: 105969.62 Hz
```

Code Snippets:

```
uint32_t t0 = MAP_Timer32_getValue(TIMER32_1_BASE);
    for (ii = 0; ii < Sample_Size; ii++)
    {
        sampleData[ii] = CAPTIO0CTL & 0x200;
    }

    uint32_t t1 = MAP_Timer32_getValue(TIMER32_1_BASE);
```

The time is taken before the for loop in to and after in t1. Within the for loop the array is filled with data and then exits when done.

```
//Handle timer wrap-around
uint32_t difference;
if (t0 >= t1)
{
    difference = t0 - t1;
}
else
{
    difference = t0 + (0xFFFFFFFF - t1) + 1;
}
//Convert timer counts to time per sample in seconds
double time_per_sample = (double)difference / (mclk_freq * Sample_Size);

//Calculate sampling rate in Hz
double sampling_rate = 1.0 / time_per_sample;
```

I include wrap around edge case code to be through.

The equation for time is usually the difference between times divided by the MCLK_Freq but since the time is for the amount it took for the sample_size I divide by sample_size to get the time it took for each item approximately. Frequency is the inverse of time so I do 1 divided by the time to get the sampling rate frequency.

Exercise 3 Frequency measurement:

It would not make sense to paste the long console output here so instead I have picked significant portions of the output for the untouched and touched run of the program. I assume the graphs are to see the whole picture.

Untouched Debug Console Outputs:

```
[CORTEX_M4_0] MCLK Frequency: 3000000 Hz
Frequency: 5852000.00 Hz
Frequency: 5851000.00 Hz
```

```
Frequency: 5850666.67 Hz
Frequency: 5851000.00 Hz
Frequency: 5851333.33 Hz
Frequency: 5851333.33 Hz
Frequency: 5851666.67 Hz
Frequency: 5851333.33 Hz
```

<-Cut here for Scroll ability

Touched Debug Console Outputs:

```
Frequency: 5375666.67 Hz
Frequency: 5376000.00 Hz
Frequency: 5375666.67 Hz
Frequency: 5376333.33 Hz
Frequency: 5375333.33 Hz
Frequency: 5375333.33 Hz
Frequency: 5375000.00 Hz
Frequency: 5374666.67 Hz
Frequency: 5358666.67 Hz
Frequency: 1329333.33 Hz
Frequency: 672666.67 Hz
Frequency: 447333.33 Hz
Frequency: 369333.33 Hz
Frequency: 350666.67 Hz
```

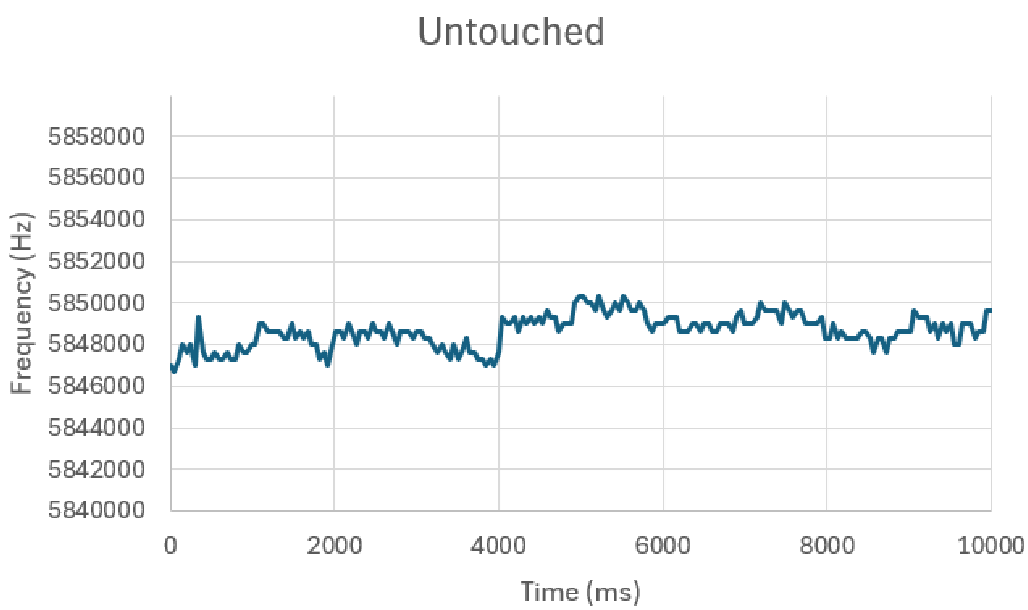
Code Snippet:

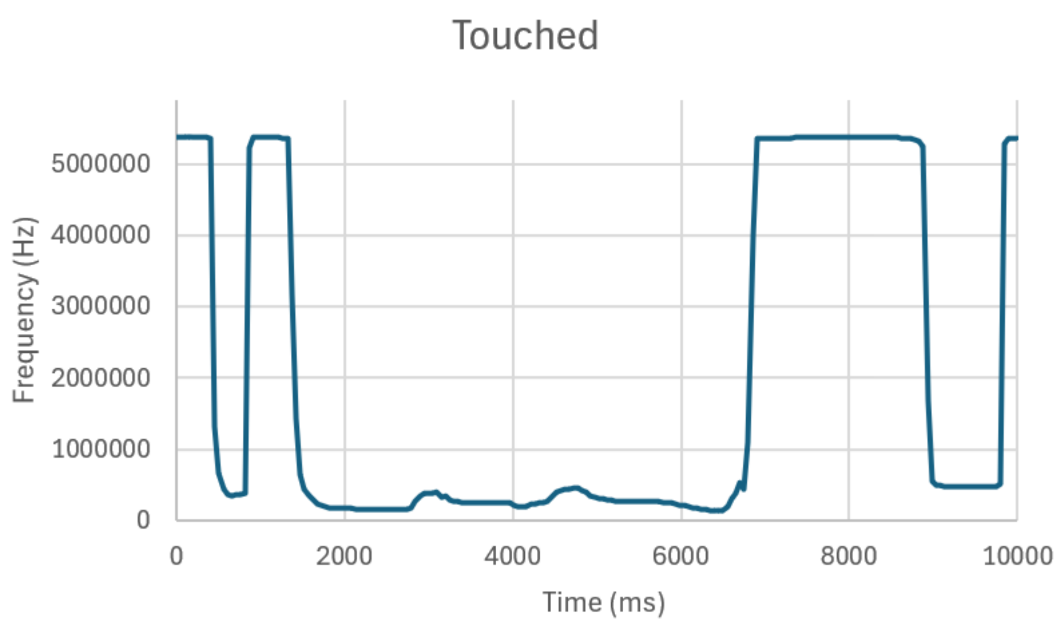
```
    uint32_t startTime = MAP_Timer32_getValue(TIMER32_1_BASE);
    uint32_t currentTime = startTime;
    uint64_t elapsedTime = 0;
    uint64_t duration = (uint64_t) mclk_freq * 10; // 10 seconds in timer counts

    while (elapsedTime < duration)
    {
        //Clear Timer_A counter
        MAP_Timer_A_clearTimer(TIMER_A2_BASE);

        //Delay for 3 ms
        delay_ms(3);

        //Read Timer_A counter value
        uint16_t counterValue = MAP_Timer_A_getCounterValue(TIMER_A2_BASE);

        //Calculate frequency
        double frequency = (double) counterValue / 0.003; //3 ms = 0.003 s

        //Print frequency
        printf("Frequency: %.2f Hz\n", frequency);
```

```
        //Update elapsed time
        uint32_t newTime = MAP_Timer32_getValue(TIMER32_1_BASE);
        if (currentTime >= newTime)
        {
            elapsedTime += currentTime - newTime;
        }
        else
        {
            elapsedTime += currentTime + (0xFFFFFFFF - newTime) + 1;
        }
        currentTime = newTime;
    }
```

Code Snippet Explanation:

I used the logic in my delay_ms function to measure the oscillation frequency. I calculated the duration of 10 seconds and have a while loop that runs while it is under this value. It gets the value and finds the difference and adds it to a tracker variable to keep track of the duration. The actual frequency measurement part involves clearing timer A using the delay_ms function and then getting value and using that to calculate the frequency.





For the touched graph run, I did one short touch then did a long hold and then a short touch. It is evident where that occurred as there is one short drop in frequency followed by a long one and then a short one. For the untouched run the oscillation frequency remains fairly level with a slight rise as time proceeds.

Exercise 4 LED control:

For this exercise I made two new functions **calibrate**(),**measure_freq**(). Calibrate gets the frequency and puts it in a global frequency, measure_freq() gets frequency and returns it as a double. I then have another value called threshold which is the calibrated frequency multiplied by 0.98. The purpose of this value is to detect when the pin has been touched, the measure frequency would drop below this value and would toggle the LED. I have a forever loop with some control logic that is if follows:

If the measured frequency drops below the threshold, the light is toggled and a bool called unpressed is set to false. If the measured frequency reaches above the threshold the bool is set to true. It has two states, pressed and unpressed.

Code Snippet:

```
//Calibrate
calibrate();
//Print Calibrated Frequency
printf("Calibrated Frequency: %.2f Hz\n", Calibrated_Freq);
double current_frequency = 0;
//threshold value to sense if pin was touched
double threshold = Calibrated_Freq * 0.98;
printf("Threshold Frequency: %.2f Hz\n", threshold);

//LED 1 Set up
MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

bool unpressed = true;
while (1)
{

    current_frequency = measure_freq();
    if (current_frequency < threshold && unpressed)
    {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
        unpressed = false;
    }
    else if (current_frequency > threshold)
    {
        unpressed = true;
    }
```

```
    }
}
```

Debug Console Output:

```
[CORTEX_M4_0] MCLK Frequency: 3000000 Hz
Calibrated Frequency: 5844666.67 Hz
Threshold Frequency: 5727773.33 Hz
```

Video Link:

https://youtube.com/shorts/S28tshiP83I?feature=share

Appendix:

Exercise 1.1 Delay Function implementation:

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

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

void delay_ms(uint32_t count);

//global variables so that function can access value
uint32_t mclk_freq;
uint64_t mclk_freq_div_1000;


int main(void)
{
    /* Stop Watchdog */
    MAP_WDT_A_holdTimer();

    //Initialize Timer32_0 in 16-bit mode
    MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1,
                           TIMER32_16BIT, TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_0_BASE, 0);
```

```
    //Initialize Timer32_1 in 32-bit mode
    MAP_Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1,
                           TIMER32_32BIT, TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_1_BASE, 0);

    //get frequency value
    mclk_freq = MAP_CS_getMCLK();
    //do math outside of delay function to streamline
    mclk_freq_div_1000 = ((uint64_t)mclk_freq) / 1000ULL;
    printf("MCLK Frequency: %u Hz\n", mclk_freq);

    uint32_t t0 = 0, t1 = 0, difference = 0;
    uint64_t time_us = 0;
    int32_t arr[] = {5000, 2000, 1000, 50, 20, 10, 5, 2, 1, 0};
    int count = 0;

    while (count < 10)
    {
        t0 = MAP_Timer32_getValue(TIMER32_1_BASE);

        delay_ms(arr[count]);

        t1 = MAP_Timer32_getValue(TIMER32_1_BASE);

        // Handle wrap-around for 32-bit timer
        if (t0 >= t1)
        {
            difference = t0 - t1;
        }
        else
        {
            difference = t0 + (0xFFFFFFFF - t1) + 1;
        }

        // Convert cycles to microseconds
        time_us = ((uint64_t)difference * 1000000ULL) / mclk_freq;

        // Calculate error in microseconds
        int64_t error_us = time_us - ((int64_t)arr[count] * 1000ULL);

        // Print the results
        printf("Target: %d ms, Difference: %u cycles, Measured time: %llu us, Error:
%lld us\n",
               arr[count], difference, time_us, error_us);

        count++;
    }

    while (1)
    {

    }
}
```

```
void delay_ms(uint32_t count)
{
    uint32_t start = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value
    uint64_t duration_cycles = (uint64_t)count * mclk_freq_div_1000;
    uint64_t elapsed_cycles = 0;
    uint32_t current;


    while (elapsed_cycles < duration_cycles)
    {
         current = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value

        if (start >= current)
        {
            elapsed_cycles += start - current;
        }
        else
        {
            //Handle 16-bit wrap-around
            elapsed_cycles += 0xFFFF - current + start + 1;
        }

        start = current;
    }
}
```

Exercise 1.2 Timing control:

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

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

void delay_ms(uint32_t count);

//global variables so that function can access value
uint32_t mclk_freq;
uint64_t mclk_freq_div_1000;


int main(void)
{
    /* Stop Watchdog */
    MAP_WDT_A_holdTimer();

    //Initialize Timer32_0 in 16-bit mode
    MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1,
                           TIMER32_16BIT, TIMER32_FREE_RUN_MODE);
```

```c
    MAP_Timer32_startTimer(TIMER32_0_BASE, 0);

    //Initialize Timer32_1 in 32-bit mode
    MAP_Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1,
                           TIMER32_32BIT, TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_1_BASE, 0);

    //get frequency value
    mclk_freq = MAP_CS_getMCLK();
    //do math outside of delay function to streamline
    mclk_freq_div_1000 = ((uint64_t)mclk_freq) / 1000ULL;
    printf("MCLK Frequency: %u Hz\n", mclk_freq);

    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    float freq_hz = .4;
    uint32_t delay_in_ms = 0;

    //delay would be half the of the time since 1 cycle is turn on and off
    //delayinms = (1/f) * (1000/2)
    delay_in_ms = (1U/freq_hz)*500U;
    printf("delay_in_ms: %u ms\n", delay_in_ms);

    while (1)
    {

        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
        delay_ms(delay_in_ms);

    }
}

void delay_ms(uint32_t count)
{
    uint32_t start = MAP_Timer32_getValue(TIMER32_0_BASE); // 16-bit value
    uint64_t duration_cycles = (uint64_t)count * mclk_freq_div_1000;
    uint64_t elapsed_cycles = 0;
    uint32_t current;


    while (elapsed_cycles < duration_cycles)
    {
         current = MAP_Timer32_getValue(TIMER32_0_BASE); // 16-bit value

        if (start >= current)
        {
            elapsed_cycles += start - current;
        }
        else
        {
            // Handle 16-bit wrap-around
            elapsed_cycles += 0xFFFF - current + start + 1;
        }
```

```
        start = current;
    }
}
```

Exercise 2 GPIO oscillator:

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

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#define Sample_Size 100

int main(void)
{
    /* Stop Watchdog  */
    MAP_WDT_A_holdTimer();

    CAPTIO0CTL = 0;
    // Clear control register
    CAPTIO0CTL |= 0x0100;   // Enable CAPTIO
    CAPTIO0CTL |= 4 << 4;   // Select Port 4. Place port number in Bits 7-4
    CAPTIO0CTL |= 1 << 1;   // Select Pin 1. Place pin number in Bits 3-1


    //Get Frequency Value
    uint32_t mclk_freq = MAP_CS_getMCLK();
    printf("MCLK Frequency: %u Hz\n", mclk_freq);

    bool sampleData[Sample_Size];
    uint32_t ii;
    //Initialize Timer32_1 in 32-bit mode
    MAP_Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1,
    TIMER32_32BIT,
                           TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_1_BASE, 0);

    uint32_t t0 = MAP_Timer32_getValue(TIMER32_1_BASE);
    for (ii = 0; ii < Sample_Size; ii++)
    {
        sampleData[ii] = CAPTIO0CTL & 0x200;
    }

    uint32_t t1 = MAP_Timer32_getValue(TIMER32_1_BASE);

    //Handle timer wrap-around
    uint32_t difference;
```

```
    if (t0 >= t1)
    {
        difference = t0 - t1;
    }
    else
    {
        difference = t0 + (0xFFFFFFFF - t1) + 1;
    }
    //Convert timer counts to time per sample in seconds
    double time_per_sample = (double)difference / (mclk_freq * Sample_Size);

    //Calculate sampling rate in Hz
    double sampling_rate = 1.0 / time_per_sample;

      //Print Data
      printf("Sampled Data: ");

      for ( ii = 0; ii < Sample_Size; ii++)
      {
          printf("%d", sampleData[ii]);
      }
      printf("\n");

      //Print Sampling Rate
      printf("Sampling Rate: %.2f Hz\n", sampling_rate);


}
```

Exercise 3 Frequency measurement:

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

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


void delay_ms(uint32_t count);

//global variables so that function delay_ms can access values
uint32_t mclk_freq;
uint64_t mclk_freq_div_1000;

int main(void)
{
    /* Stop Watchdog  */
    MAP_WDT_A_holdTimer();

    CAPTIO0CTL = 0;
```

```
// Clear control register
CAPTIO0CTL |= 0x0100;   // Enable CAPTIO
CAPTIO0CTL |= 4 << 4;   // Select Port 4. Place port number in Bits 7-4
CAPTIO0CTL |= 1 << 1;   // Select Pin 1. Place pin number in Bits 3-1

//get frequency value
mclk_freq = MAP_CS_getMCLK();
//do math outside of delay function to streamline
mclk_freq_div_1000 = ((uint64_t) mclk_freq) / 1000ULL;
printf("MCLK Frequency: %u Hz\n", mclk_freq);

//Initialize Timer32_0 in 16-bit mode
MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1,
TIMER32_16BIT,
                       TIMER32_FREE_RUN_MODE);
MAP_Timer32_startTimer(TIMER32_0_BASE, 0);

//Initialize Timer32_1 in 32-bit mode
MAP_Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1,
TIMER32_32BIT,
                       TIMER32_FREE_RUN_MODE);
MAP_Timer32_startTimer(TIMER32_1_BASE, 0);

Timer_A_ContinuousModeConfig timer_continuous_obj;
timer_continuous_obj.clockSource =
        TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK;
timer_continuous_obj.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
timer_continuous_obj.timerInterruptEnable_TAIE =
TIMER_A_TAIE_INTERRUPT_DISABLE;
timer_continuous_obj.timerClear = TIMER_A_DO_CLEAR;
MAP_Timer_A_configureContinuousMode(TIMER_A2_BASE, &timer_continuous_obj);
MAP_Timer_A_startCounter(TIMER_A2_BASE, TIMER_A_CONTINUOUS_MODE);


uint32_t startTime = MAP_Timer32_getValue(TIMER32_1_BASE);
uint32_t currentTime = startTime;
uint64_t elapsedTime = 0;
uint64_t duration = (uint64_t) mclk_freq * 10; // 10 seconds in timer counts

while (elapsedTime < duration)
{
    //Clear Timer_A counter
    MAP_Timer_A_clearTimer(TIMER_A2_BASE);

    //Delay for 3 ms
    delay_ms(3);

    //Read Timer_A counter value
    uint16_t counterValue = MAP_Timer_A_getCounterValue(TIMER_A2_BASE);

    //Calculate frequency
    double frequency = (double) counterValue / 0.003; //3 ms = 0.003 s

    //Print frequency
    printf("Frequency: %.2f Hz\n", frequency);
```

```
        //Update elapsed time
        uint32_t newTime = MAP_Timer32_getValue(TIMER32_1_BASE);
        if (currentTime >= newTime)
        {
            elapsedTime += currentTime - newTime;
        }
        else
        {
            elapsedTime += currentTime + (0xFFFFFFFF - newTime) + 1;
        }
        currentTime = newTime;
    }

    while (1)
    {
    }
}
void delay_ms(uint32_t count)
{
    uint32_t start = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value
    uint64_t duration_cycles = (uint64_t) count * mclk_freq_div_1000;
    uint64_t elapsed_cycles = 0;
    uint32_t current;

    while (elapsed_cycles < duration_cycles)
    {
        current = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value

        if (start >= current)
        {
            elapsed_cycles += start - current;
        }
        else
        {
            //Handle 16-bit wrap-around
            elapsed_cycles += 0xFFFF - current + start + 1;
        }

        start = current;
    }
}
```

Exercise 4 LED control:

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

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

void delay_ms(uint32_t count);
void calibrate();
```

```
double measure_freq();
//global variables so that function delay_ms can access values
uint32_t mclk_freq;
uint64_t mclk_freq_div_1000;

double Calibrated_Freq;

int main(void)
{
    /* Stop Watchdog  */
    MAP_WDT_A_holdTimer();

    CAPTIO0CTL = 0;
    // Clear control register
    CAPTIO0CTL |= 0x0100;   // Enable CAPTIO
    CAPTIO0CTL |= 4 << 4;   // Select Port 4. Place port number in Bits 7-4
    CAPTIO0CTL |= 1 << 1;   // Select Pin 1. Place pin number in Bits 3-1

    //Initialize Timer32_0 in 16-bit mode
    MAP_Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1,
    TIMER32_16BIT,
                           TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_0_BASE, 0);

    //Initialize Timer32_1 in 32-bit mode
    MAP_Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1,
    TIMER32_32BIT,
                           TIMER32_FREE_RUN_MODE);
    MAP_Timer32_startTimer(TIMER32_1_BASE, 0);

    //Timer A set up
    Timer_A_ContinuousModeConfig timer_continuous_obj;
    timer_continuous_obj.clockSource =
    TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK;
    timer_continuous_obj.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    timer_continuous_obj.timerInterruptEnable_TAIE =
    TIMER_A_TAIE_INTERRUPT_DISABLE;
    timer_continuous_obj.timerClear = TIMER_A_DO_CLEAR;
    MAP_Timer_A_configureContinuousMode(TIMER_A2_BASE, &timer_continuous_obj);
    MAP_Timer_A_startCounter(TIMER_A2_BASE, TIMER_A_CONTINUOUS_MODE);

    //get frequency value
    mclk_freq = MAP_CS_getMCLK();
    //do math outside of delay function to streamline
    mclk_freq_div_1000 = ((uint64_t) mclk_freq) / 1000ULL;
    printf("MCLK Frequency: %u Hz\n", mclk_freq);

    //Calibrate
    calibrate();
    //Print Calibrated Frequency
    printf("Calibrated Frequency: %.2f Hz\n", Calibrated_Freq);
    double current_frequency = 0;
    double threshold = Calibrated_Freq * 0.98;
    printf("Threshold Frequency: %.2f Hz\n", threshold);
```

```
    //LED 1 Set up
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    bool unpressed = true;
    while (1)
    {

        current_frequency = measure_freq();
        if (current_frequency < threshold && unpressed)
        {
            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
            unpressed = false;
        }
        else if (current_frequency > threshold)
        {
            unpressed = true;
        }

    }
}
void delay_ms(uint32_t count)
{
    uint32_t start = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value
    uint64_t duration_cycles = (uint64_t) count * mclk_freq_div_1000;
    uint64_t elapsed_cycles = 0;
    uint32_t current;

    while (elapsed_cycles < duration_cycles)
    {
        current = MAP_Timer32_getValue(TIMER32_0_BASE); //16-bit value

        if (start >= current)
        {
            elapsed_cycles += start - current;
        }
        else
        {
            //Handle 16-bit wrap-around
            elapsed_cycles += 0xFFFF - current + start + 1;
        }

        start = current;
    }
}

void calibrate()
{
    //Clear Timer_A counter
    MAP_Timer_A_clearTimer(TIMER_A2_BASE);

    //Delay for 3 ms
    delay_ms(3);

    //Read Timer_A counter value
```

```
    uint16_t counterValue = MAP_Timer_A_getCounterValue(TIMER_A2_BASE);

    //Calculate frequency
    Calibrated_Freq = (double) counterValue / 0.003; //3 ms = 0.003 s

}

double measure_freq()
{
    MAP_Timer_A_clearTimer(TIMER_A2_BASE);

    //Delay for 3 ms
    delay_ms(3);

    //Read Timer_A counter value
    uint16_t counterValue = MAP_Timer_A_getCounterValue(TIMER_A2_BASE);

    //Calculate frequency
    double frequency = (double) counterValue / 0.003; //3 ms = 0.003 s
    return frequency;

}
```