EEL 4742C: Embedded Systems
Name: Hamzah Ullah
Lab 3: Using the Timer

Introduction:

In the lab, we will be introduced to different timer modes, specifically up mode and continuous mode. Continuous mode is when Timer_A is running at a certain frequency with no stopping point or reference other than the frequency, and the clearing of the flag. Up mode allows us to manipulate how many cycles the will occur till the flag is cleared, as opposed to just manipulating the timer frequency. With these two methods, we will enable the clock and use it to manipulate the LEDs.

Part 1: The Continuous Mode

This part of the lab introduces us to continuous mode using ACLK. At default, the ACLK is configured at 32Khz but the lab has given us a function to configure the clock to 32Khz to be used in future labs. It does this by rerouting the corresponding pins to the ACLK which we are using for this lab, and does not become stable until the fault flags have been cleared.

To configure the frequency and clock type, we must manipulate the TA0CTL to the corresponding values. The values we will using are, ACLK, DIV by 1, Continuous, and TACLR which results TAR to zero and resets the clock divider. To do this we set the TA0CTL to the following :

TA0CTL = TASSEL_1 | ID_0 | MC_2| TACLR;

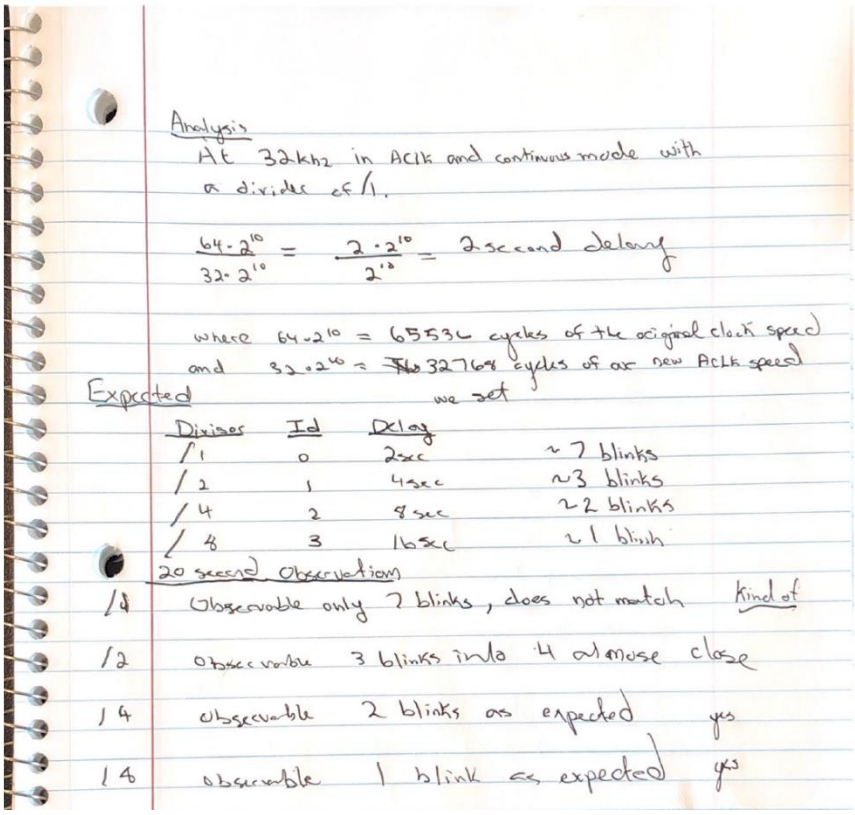
The second part of the program is to configure the while loop. The while loop waits until the Flag has been raised before triggering the desired response of triggering the LED. Below is the code that shows the full continuous mode, as well as the analysis and response of each divisor:

Continuous Mode Code with ID 0 for /1 with expected delay of 2 seconds

```
1 // Flashing the LED with Timer_A, continuous mode, via polling
2 #include <msp430fr6989.h>
3 #define redLED BIT0 //Red LED at P1.0
4 #define greenLED BIT7 //Green LED at P9.7
5
6 void main(void)
7 {
8     WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer
9     PM5CTL0 &= ~LOCKLPM5;           //Enable the GPIO Pins
10
11    P1DIR |= redLED;    //Direct pin as Output
12    P9DIR |= greenLED;  //Direct pin as Output
13    P1OUT &= ~redLED;   //Turn LED off
14    P9OUT &= ~greenLED; //Turn LED off
15
16 //  Configure ACLK to the 32Khz Crystal(function call).
17    config_ACLK_to_32KHz_crystal();
18    //Configure Timer_A
19    //Use ACLK, divide by 1, continuous mode, clear TAR
20    TA0CTL = TASSEL_1 | ID_0 | MC_2 |TACLR;
21
22    //Ensure flag is cleared at the start
23    TA0CTL &= ~TAIFG;
24
25    //Infinite loop
26    for(;;)
27    {
28        //Empty while loop; waits here until TAIFG raised
29        while((TA0CTL & TAIFG)== 0){}
30        P1OUT ^= redLED;
31        TA0CTL &= ~TAIFG; //set flag
32
33    }
34 }
```

```c
//Configure ACLK to the 32Khz crystal (function call)
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

        //Reroute pins to LFXIN/LFXOUT functionality
        PJSEL1 &= ~BIT4;
        PJSEL0 |= BIT4;

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

        CSCTL0_H = 0; //lock CS registers
        return;
    }
```

Analysis and Observations



Analysis

At 32kHz in ACLK and continuous mode with a divider of /1.

$\frac{64 \cdot 2^{10}}{32 \cdot 2^{10}} = \frac{2 \cdot 2^{10}}{2^{10}} =$ 2 second delay

where $64 \cdot 2^{10}$ = 65536 cycles of the original clock speed

and $32 \cdot 2^{10}$ = ~~The~~ 32768 cycles of our new ACLK speed we set

Expected

Divider	Id	Delay	
/1	0	2sec	~7 blinks
/2	1	4sec	~3 blinks
/4	2	8 sec	~2 blinks
/8	3	16sec	~1 blink

20 second Observation

/1 Observable only 7 blinks, does not match kind of

/2 Observable 3 blinks into 4 almost close

/4 observable 2 blinks as expected yes

/8 observable 1 blink as expected yes

Part 2: The Up Mode

The up mode gives us more control over the frequency of the flag to be raised, allowing us to set specific intervals of when we want the trigger the event. In terms of coding, it is extremely similar with an extra line of code to configure the length of the period. To do this, we must understand the frequency notation. 32KHz is more clearly explained as 32Khz/s (kilohertz per second) meaning that every second the frequency is running at the labeled speed. In order to achieve 1 second, we have to determine the period of 32Khz/s. To do this, we convert it using computer engineering understanding of 32Khz. This means it is not defined as 32000 cycles, but rather 32768 cycles. To achieve this number, we use kilo defined as 2^{10} = 1024 cycles, and determine the value of 2^x = 32, which calculates out to 2^5. Multiplying those values out we get:

$2^5 * 2^{10} = 32768$

This is the amount of cycles per second, and the value we will use for TA0CCR0. We must subtract by one to take into account that the value 1 starts at 0 in the programming language, therefore:

TA0CCR0 = 32768 – 1; //1 sec period for a 32khz clock.

The TA0CTL is also slightly different by simply selecting the up mode setting, below is the line of code:

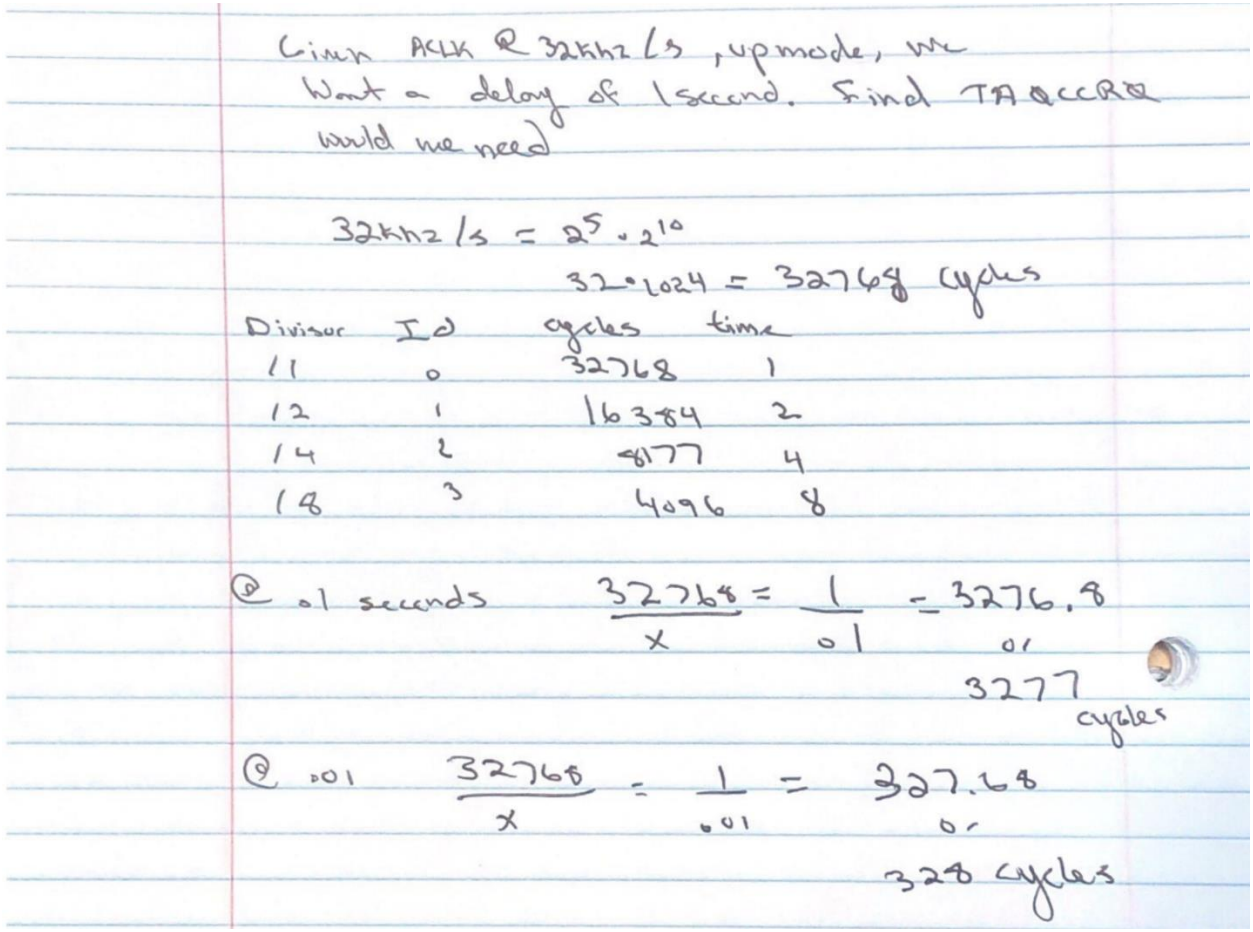
TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;

Below is the full code: **Up mode, 1 sec delay , 2 second light ACLK with divisor of 1.**

```
#include <msp430fr6989.h>
#define redLED BIT0
#define greenLED BIT7

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= redLED; //Direct pin as output
    P9DIR |= greenLED; //Direct pin as output
    P1OUT &= ~redLED; //Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure ACLK to the 32khz crystal(function call).
    config_ACLK_to_32KHz_crystal();
    //Set Timer Period
    TA0CCR0 = (32768-1);
    //Use ACLK, divide by 1, continuous mode, clear TAR
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;

    //Ensure flag is cleared at the start
    TA0CTL &= ~TAIFG;

    //Infinite loop
    for(;;)
    {
        // empty while loop; waits here until TAIFG raised
        while((TA0CTL & TAIFG) == 0){}
        P1OUT ^= redLED;
        TA0CTL &= ~TAIFG; //set flag

    }
}
```

```
36      }
37  }
38  //Configure ACLK to the 32Khz crystal (function call)
39  void config_ACLK_to_32KHz_crystal()
40      {
41          //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz
42
43          //Reroute pins to LFXIN/LFXOUT functionality
44          PJSEL1 &= ~BIT4;
45          PJSEL0 |= BIT4;
46
47          //Wait until the oscillator fault flags remain cleared
48          CSCTL0 = CSKEY;
49          do
50          {
51              CSCTL5 &= ~LFXTOFFG;    //local fault flag
52              SFRIFG1 &= ~OFIFG;      //Global fault flag
53          }
54          while((CSCTL5 & LFXTOFFG) != 0);
55
56          CSCTL0_H = 0; //lock CS registers
57          return;
58      }
```

Analysis and observations:



Part 3: Your Own Design

For continuous mode, I could of just blinked both lights on, which I did, but I made a counter so that at even numbers the divisor would change to 32khz/8 for a significantly slower trigger time for the lights, and the default /1 at the odd numbers, and it would oscillate between those numbers.

Continuous mode design, both red and green LED turn on but changes the divisor between even and odd value of count using modulo:

```
// Flashing the LED with Timer_A, continuous mode, via polling
#include <msp430fr6989.h>
#define redLED BIT0 //Red LED at P1.0
#define greenLED BIT7 //Green LED at P9.7

void main(void)
{
    int count= 0;
    WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;           //Enable the GPIO Pins

    P1DIR |= redLED;    //Direct pin as Output
    P9DIR |= greenLED;  //Direct pin as Output
    P1OUT &= ~redLED;   //Turn LED off
    P9OUT &= ~greenLED; //Turn LED off


//  Configure ACLK to the 32Khz Crystal(function call).
    config_ACLK_to_32KHz_crystal();

    //Use ACLK, divide by 1, up mode, clear TAR
    TA0CTL = TASSEL_1 | ID_0 | MC_2 |TACLR;


    //Ensure flag is cleared at the start
    TA0CTL &= ~TAIFG;

    //Infinite loop
    for(;;)
    {
        //Empty while loop; waits here until TAIFG raised
        while((TA0CTL & TAIFG)== 0){}
        //checks for even number, to change divisor to /8 else remain divisor /1
        if( count % 2 != 0)
            TA0CTL = TASSEL_1 | ID_3 | MC_2 |TACLR;
        else
            TA0CTL = TASSEL_1 | ID_0 | MC_2 |TACLR;
        //LED output
        P9OUT ^= greenLED;
        P1OUT ^= redLED;
```

```
        TA0CTL &= ~TAIFG; //clear flag
        count++; //increments count to cycle between even and odd
    }
}
//Configure ACLK to the 32Khz crystal (function call)
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

        //Reroute pins to LFXIN/LFXOUT functionality
        PJSEL1 &= ~BIT4;
        PJSEL0 |= BIT4;

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

        CSCTL0_H = 0; //lock CS registers
        return;
    }
```

For up mode, I added a line of code that would gradually decrement the number of cycles by 1000 every time the flag was raised prior to blinking the lights. This would cause the lights to blink faster and faster before resetting back to the maximum period.

BELOW IS UP MODE CODE

Up Mode gradually flashes faster as the period reaches 0 and resets to MAX

```
#include <msp430fr6989.h>
#define redLED BIT0
#define greenLED BIT7

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= redLED; //Direct pin as output
    P9DIR |= greenLED; //Direct pin as output
    P1OUT &= ~redLED; //Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure ACLK to the 32khz crystal(function call).
    config_ACLK_to_32KHz_crystal();
    //Set Timer Period
    TA0CCR0 = (32768-1);
    //Use ACLK, divide by 1, continuous mode, clear TAR
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;

    //Ensure flag is cleared at the start
    TA0CTL &= ~TAIFG;

    //Infinite loop
    for(;;)
    {
        // empty while loop; waits here until TAIFG raised
        while((TA0CTL & TAIFG) == 0){}

        TA0CCR0 -=1000;
        P1OUT ^= redLED;
        P9OUT ^= greenLED;
        TA0CTL &= ~TAIFG; //set flag
         //set flag
    }
}
```

QUESTIONS:

1. **So far, we have seen two ways of generating delays: using a delay loop and using Timer_A. Which approach provides more control and accuracy over the delays? Explain.**
   Although at first glance, the delay loop seems like a very simple way to create triggers for the code, it is not very accurate. Timers allows us to calculate exactly how much time we need between different functions. Especially up mode, up mode allows us to set the exact number of cycles relative to the frequency of the clock speed. Therefore, the Timer is for more accurate and gives us more control over its functions, tan simply adjusting the delay.

2. **Explain the polling technique and how it's used in this lab.**
   The polling technique is when we allow the MCU to cycle through at a certain frequency. Once the MCU has reached the end of the polling limit determined by the frequency at which it reaches it, the flag is raised which allows us to create a trigger to certain actions to occur. In this lab, the up mode allows us to create artificial triggers by adjusting the limit of the polling, while continuous polls at the end of the clock cycle based off the frequency.

3. **Is the polling technique a suitable choice when we care about saving battery power? Explain.**
   Polling is not suitable when it comes to battery power, because the MCU is constantly running till it reaches the flag where it would need to be reset. A much better approach would be to create interrupts in the code which allows us to pause the system at a certain point, initiate our triggered response, and restart when desired.

4. **If we write 0 to TAR using line code, does the TAIFG go to 1?**
   No the TAIFG is not set if TAR is set to zero.

5. **In this lab, we used TAIFG to time the duration. TAIFG is known as Timer_A Interrupt flag, which is an interrupt flag. Were we using interrupts in this lab? Explain.**
   In a sense we were not using a true interrupt where the system completely stops to deal with the issue, but just a flag. The flag just pauses the system till the flag is cleared, than resumes operation. A true interrupt would stop he timer via a vector. Therefore we were not using interrupts in the lab.

6. **From what we have seen in this lab, which mode gives us more control over the timing duration: the up mode or the continuous mode?**
   The up mode because it allows us to change the period at which the TAIFG will restart at zero by manipulating the TA0CCR0 length which is the period of the timer based off of frequency.