Kyle Rosenthal

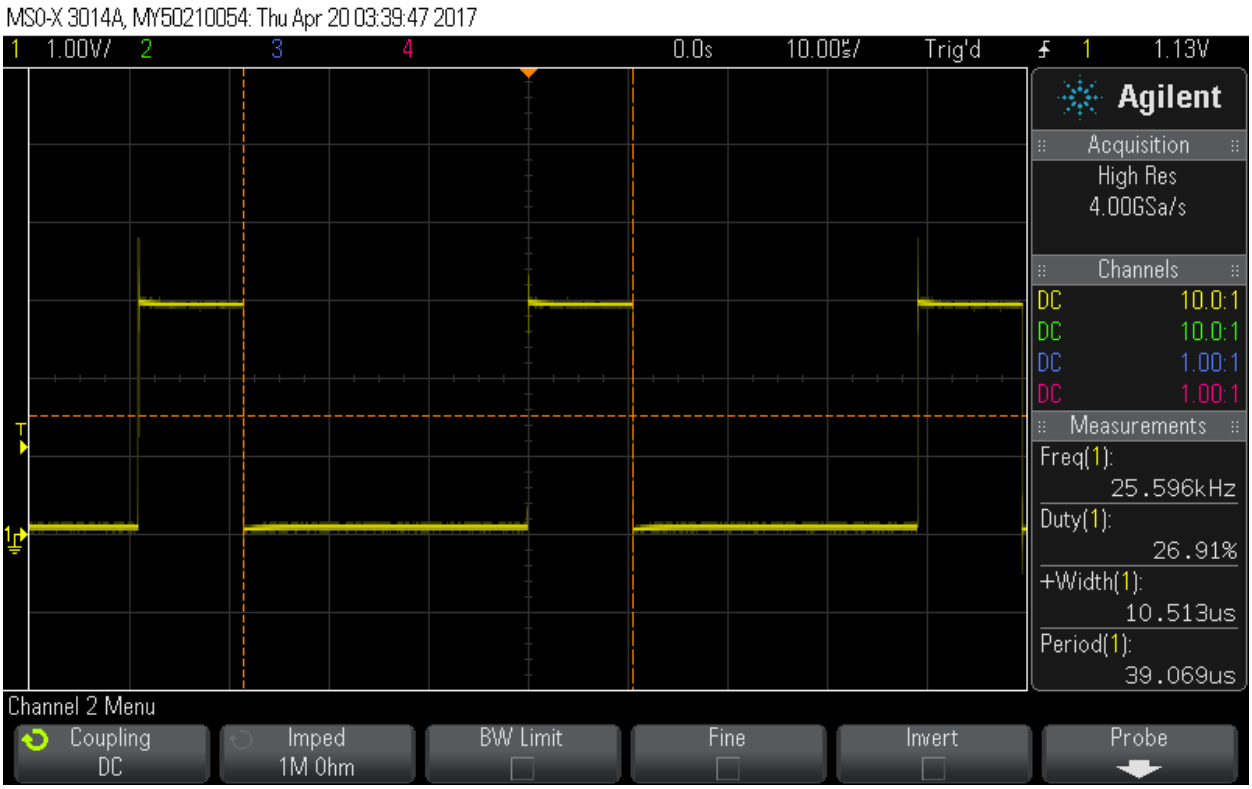
Assignment 4

Step 1

Observed blinking LED.

Step 2

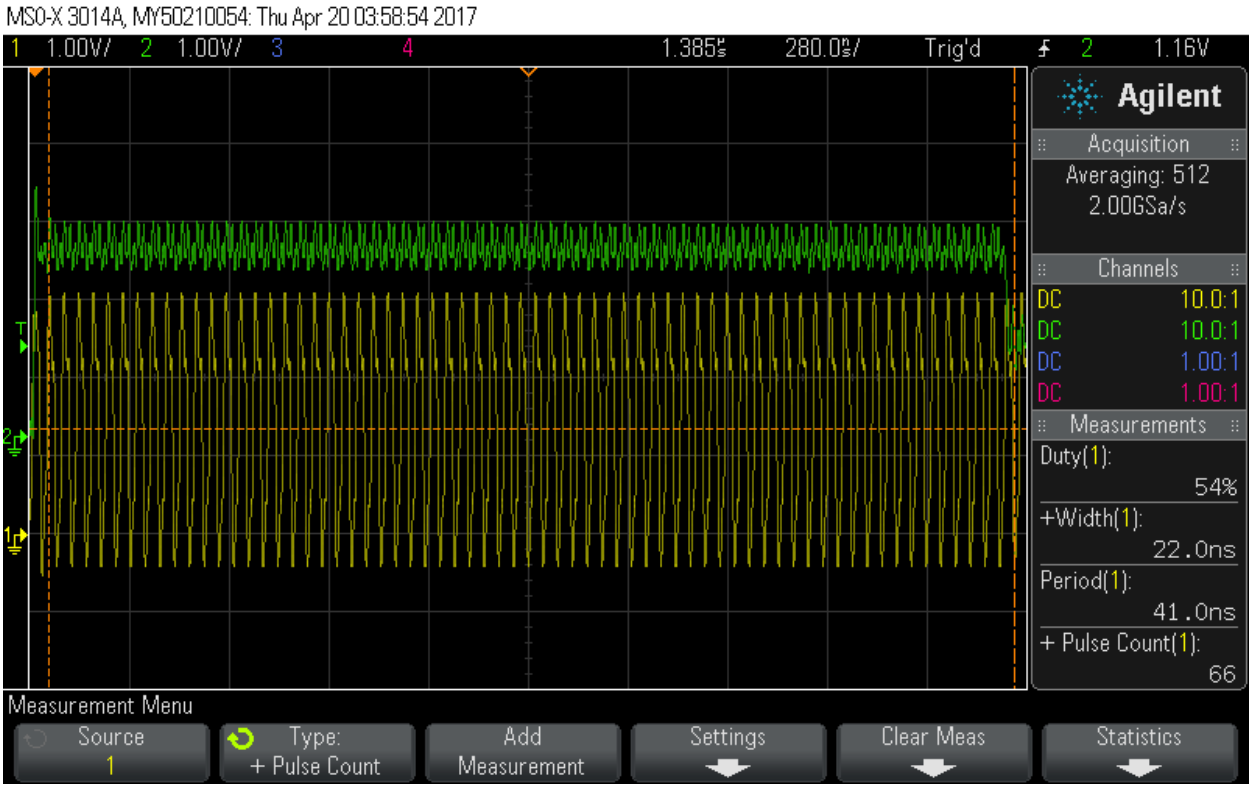
Period = 1 / 25000; CCR[0] = (1 / 25k) / (1 / 24M) / 4 then tweaked a bit to make it closer to perfect



Step 3

Double CCRO and toggle bit every time to raise duty cycle to 50%.

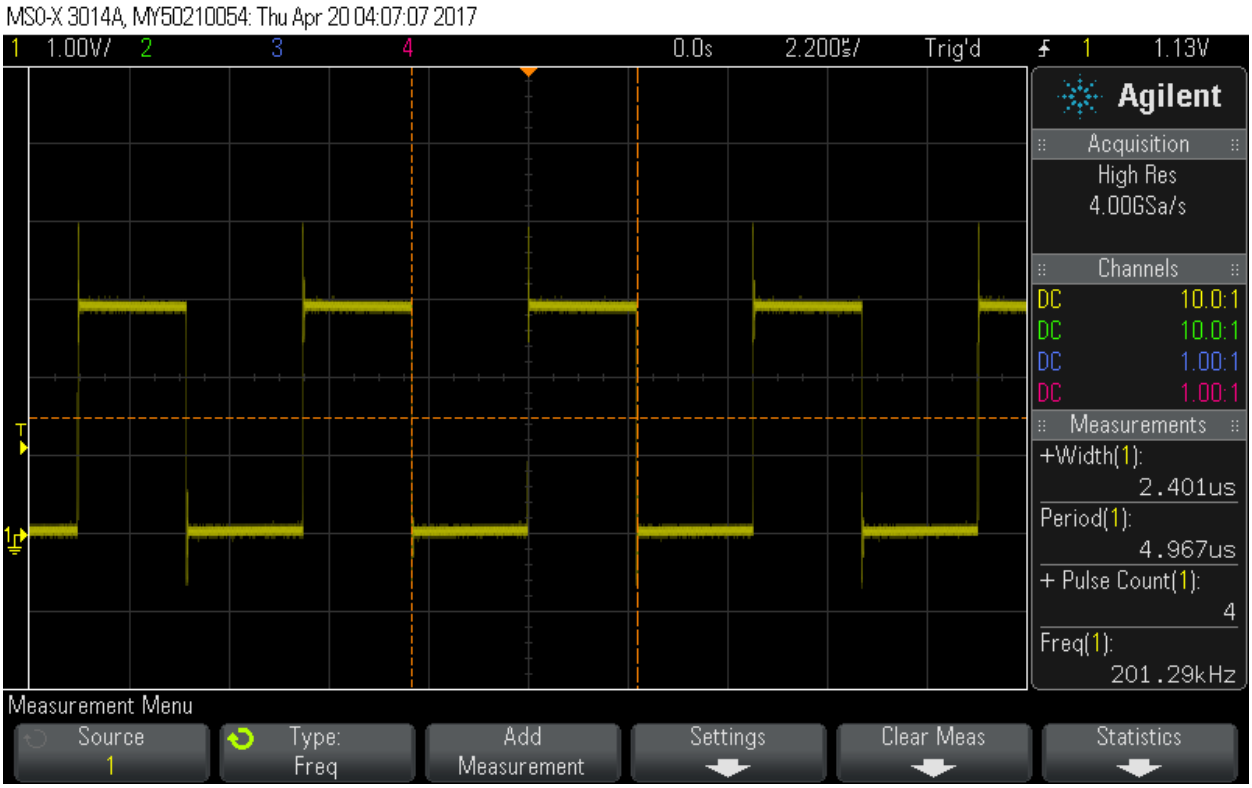
Step 4



66 cycles for the interrupt.

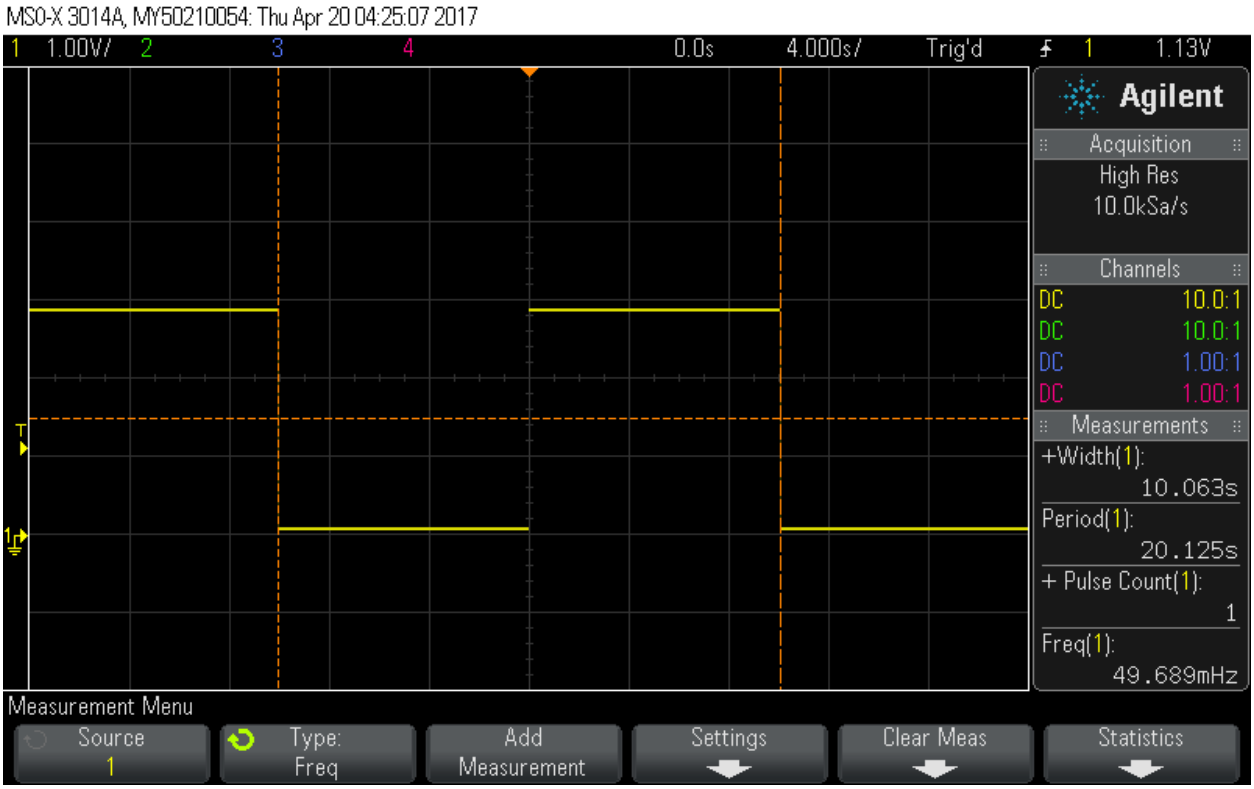
Step 5

Lowest value was 60, which is in line with the cycle count from step 4.



Step 6

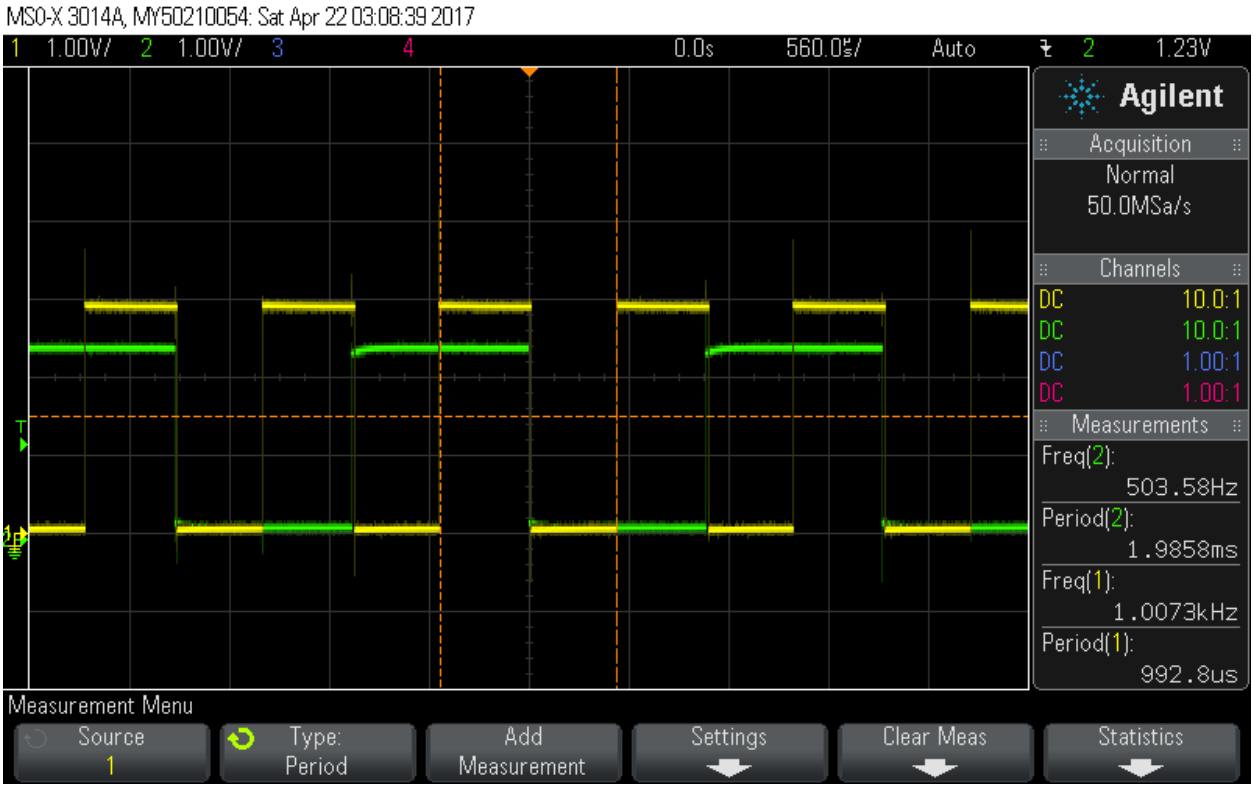
1.5Hz with CCRO of 0xFFFF gets a 86ms period, to get that to 20 seconds we need to only toggle the bit every 465 times (20 secs/ 86 ms).



Step 7

Created two interrupts with one toggling both output bits, and one only toggling one of them.

Step 8



The waveforms effectively form a 2-bit counter.

Youtube Link

20 second clock

https://www.youtube.com/watch?v=UT6FJ-BboNk

Reflex

https://www.youtube.com/watch?v=JcMbWuh2dOM

Clock Calculation

Code

25KHz main.c

```
#include "msp.h"

#define FREQ_24_MHz     24000000

int main(void) {
    WDT_A->CTL = WDT_A_CTL_PW |             // Stop WDT
            WDT_A_CTL_HOLD;


    CS->KEY = CS_KEY_VAL; // unlock CS registers
```

```
    CS->CTL0 = 0; // clear register CTL0
    CS->CTL0 = CS_CTL0_DCORSEL_4; //set 24 MHz
    CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3; // select clock
sources
    CS->KEY = 0; // lock the CS registers

    // Configure GPIO
    P1->DIR |= BIT0;
    P1->OUT |= BIT0;

    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE; // TACCR0 interrupt enabled
    TIMER_A0->CCR[0] = 55;
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK, continuous mode
            TIMER_A_CTL_MC__CONTINUOUS |
            TIMER_A_CTL_ID__1;

    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;    // Enable sleep on exit from ISR

    // Enable global interrupt
    __enable_irq();

    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);

    //MCLK out on pin 4.3
    P4->SEL0 |= BIT3;
    P4->DIR |= BIT3;

    P6->DIR |= BIT0;

    while (1)
    {
        __sleep();

        __no_operation();                   // For debugger
    }
}

// Timer A0 interrupt service routine
// Step 2 CCRO = 172 * 2, 64 *2
// STEP 4 CCRO = 256
// step 5 ccro = 60
void TA0_0_IRQHandler(void) {
    //P6->OUT |= BIT0;
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
    if (P1->OUT & BIT0) {
        TIMER_A0->CCR[0] += 256;
        P1->OUT &= ~BIT0;
    } else {
        TIMER_A0->CCR[0] += 256;
        P1->OUT |= BIT0;
    }
    //P6->OUT &= ~BIT0;
}
```

20-sec period main.c

```
#include "msp.h"

#define FREQ_24_MHz     24000000

int main(void) {
    WDT_A->CTL = WDT_A_CTL_PW |             // Stop WDT
            WDT_A_CTL_HOLD;


    CS->KEY = CS_KEY_VAL; // unlock CS registers
    CS->CTL0 = 0; // clear register CTL0
    CS->CTL0 = CS_CTL0_DCORSEL_0; //set 1.5 MHz
    CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3; // select clock
sources
    CS->KEY = 0; // lock the CS registers

    // Configure GPIO
    P1->DIR |= BIT0;
    P1->OUT |= BIT0;

    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE; // TACCR0 interrupt enabled
    TIMER_A0->CCR[0] = 0xFFFF;
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK, continuous mode
            TIMER_A_CTL_MC__CONTINUOUS |
            TIMER_A_CTL_ID__1;

    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;    // Enable sleep on exit from ISR

    // Enable global interrupt
    __enable_irq();

    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);

    //MCLK out on pin 4.3
    P4->SEL0 |= BIT3;
    P4->DIR |= BIT3;

    P6->DIR |= BIT0;

    while (1)
    {
        __sleep();

        __no_operation();                   // For debugger
    }
}

// Timer A0 interrupt service routine
// Step 2 CCRO = 172 * 2, 64 *2
// STEP 4 CCRO = 256
// step 5 ccro = 60
void TA0_0_IRQHandler(void) {
    static int x = 0;
```

```
    x++;
    //P6->OUT |= BIT0;
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
    if (x >= 232 && P1->OUT & BIT0) {
        //TIMER_A0->CCR[0] += 256;
        P1->OUT &= ~BIT0;
        x = 0;
    } else if (x >= 232) {
        //TIMER_A0->CCR[0] += 256;
        P1->OUT |= BIT0;
        x = 0;
    }
    //P6->OUT &= ~BIT0;
}
```

2-bit counter main.c

```
#include "msp.h"

#define FREQ_24_MHz     24000000

int main(void) {
    WDT_A->CTL = WDT_A_CTL_PW |             // Stop WDT
            WDT_A_CTL_HOLD;


    CS->KEY = CS_KEY_VAL; // unlock CS registers
    CS->CTL0 = 0; // clear register CTL0
    CS->CTL0 = CS_CTL0_DCORSEL_0; //set 1.5 MHz
    CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3; // select clock
sources
    CS->KEY = 0; // lock the CS registers

    // Configure GPIO
    P1->DIR |= BIT0;
    P1->OUT |= BIT0;

    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE; // TACCR0 interrupt enabled
    TIMER_A0->CCR[0] = 1500;
    TIMER_A0->CCTL[1] = TIMER_A_CCTLN_CCIE; // TACCR1 interrupt enabled
    TIMER_A0->CCR[1] = 750;
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK, continuous mode
            TIMER_A_CTL_MC__CONTINUOUS |
            TIMER_A_CTL_ID__1;

    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;    // Enable sleep on exit from ISR

    // Enable global interrupt
    __enable_irq();

    NVIC->ISER[0] = 1 << ((TA0_N_IRQn) & 31);
    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);

    //MCLK out on pin 4.3
```

```
    //P4->SEL0 |= BIT3;
    //P4->DIR |= BIT3;

    P6->DIR |= BIT0;
    P6->OUT |= BIT0;

    while (1)
    {
        __sleep();

        __no_operation();                   // For debugger
    }
}



// Timer A0 interrupt service routine
// Step 2 CCRO = 172 * 2, 64 *2
// STEP 4 CCRO = 256
// step 5 ccro = 60
void TA0_0_IRQHandler(void) {

    //P6->OUT |= BIT0;

    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
    if (P1->OUT & BIT0) {
        TIMER_A0->CCR[0] += 1500;
        P1->OUT &= ~BIT0;
      //  x = 0;
    } else {
        TIMER_A0->CCR[0] += 1500;
        P1->OUT |= BIT0;
    //    x = 0;
    }
}
void TA0_N_IRQHandler(void) {
    if(TIMER_A0->CCTL[1]&TIMER_A_CCTLN_CCIFG)
    {
        TIMER_A0->CCTL[1] &= ~TIMER_A_CCTLN_CCIFG;
        if (P6->OUT & BIT0) {
            TIMER_A0->CCR[1] += 1500;
            P6->OUT &= ~BIT0;
          //  x = 0;
        } else {
            TIMER_A0->CCR[1] += 1500;
            P6->OUT |= BIT0;
        //    x = 0;
        }
        if (P1->OUT & BIT0) {

            P1->OUT &= ~BIT0;
          //  x = 0;
        } else {

            P1->OUT |= BIT0;
```

```
        //    x = 0;
        }
    }
    //P6->OUT &= ~BIT0;
}
```