MSP432 Microcontroller Lab

CS 473 Embedded Systems

By: Abhi Kamboj

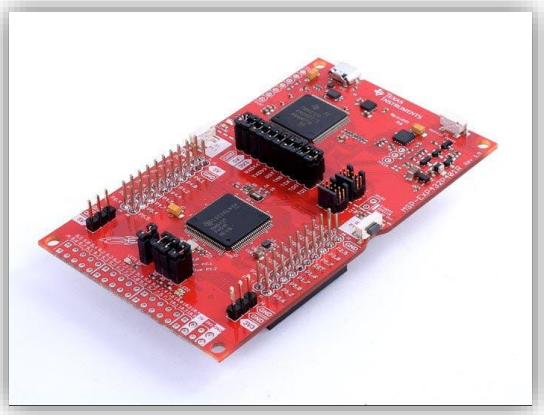

Table of Contents

Introduction:

In this lab I was tasked with using the Timer and Analog to Digital Converter (ADC) modules of the MSP432P401R TI microcontroller to read a voltage from a potentiometer and output a Pulse Width Modulation (PWM) signal on the General Purpose Input/Output (GPIO) pins to control a servo motor. The servo is controlled by a pulse width of 1 to 2 ms out of a 20 ms period, so I used the timer to generate a PWM of 20 ms and an interrupt every 20 ms. The timer interrupt would start a conversion of the potentiometer's voltage value using the ADC. Once the conversion was finished, the ADC would raise another interrupt and adjust the timer to change the duty cycle of PWM. Finally, the adjusted PWM would be outputted on the GPIO pin where the servo was attached, and the servo's arm would rotate accordingly.

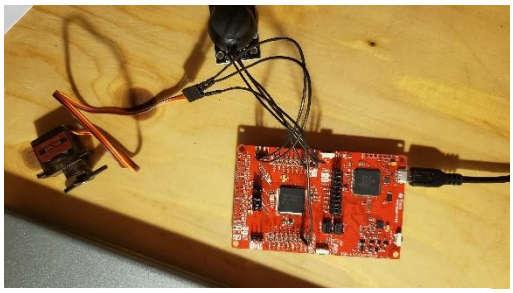

Figure 1: Assembled Circuit

Materials and Methods:

Clock:

In order to use the Timer module, I needed to use an internal clock, so I decided to use the Auxiliary Clock (ACLK). I chose the ACLK because it is easily selectable by setting the timer module's TASSEL bits to '01' and it is also available for the ADC if I needed it. For the ACLK module's source, I used the very-low-power low-frequency oscillator (VLOCLK). Given the amount of time needed for the interrupts (20 ms), I would not be needing a high frequency or high-power consumption clock, so I concluded the VLOCLK was an appropriate choice.

Timer:

I used the Timer_A module to generate interrupts every 20 ms and to create a PWM. In order to control the servo, I needed to create a duty cycle between 5-10% on the 20 ms PWM. I accomplished this by setting the timer to Up-Mode and using the Capture Control Registers zero (CCR0) and one (CCR1). Both were set to compare mode, and CCR1 was set to the output mode Set/Reset. With these settings the timer counts to the value in CCR0 and resets. Every time the timer reaches the value in CCR1 it sets the OUT1 bit and when the timer reaches CCR0 it resets OUT1 creating a PWM as shown in Figure 2. The OUT1 value is mapped to P2.4, so that port was enabled, and the servo motor was attached to it. I also enabled interrupts for CCR0 and created an interrupt handler to initiate the analog to digital signal conversion.

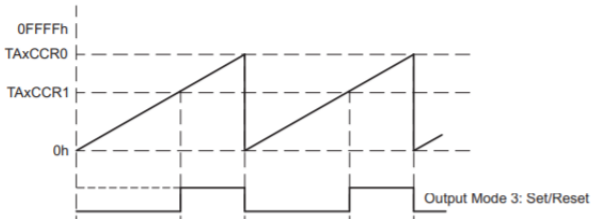

Figure 2: Timer in Up Mode, CCR1 in Set/Reset mode. Image from datasheet.

ADC:

I used the ADC in single-channel single-conversion mode. In this mode, I would initiate a conversion by setting the ADC set conversion (SC) bit during the timer's interrupt handle. Then, after the conversion was finished the bit would automatically reset and trigger an interrupt. The interrupt read the converted value from ADC memory register 0 and changed the timer's value accordingly to adjust the PWM.

I setup P4.0 on the board to input the analog signal to the ADC by enabling P4.0 to use its primary module, setting it to input direction and then setting the ADC14INCH bits to '01101' for input A13 which is mapped to P4.0. The potentiometer was then attached to P4.0 and a reference voltage of 3.0 V as that is the reference voltage the ADC assumes.

Testing:

Each components functionality was tested individually before I combined them all. The two main devices that were used for testing were the LEDs on the board and the logic analyzer.

LEDs:

Lots of basic functionality was easy to test with the LEDs. When testing the timer, I flashed the LEDs to make sure the timer and clock was working. To check if an interrupt ever occurred, or whether a function was being called, I could just flash the LED. In addition, using the LEDs allowed me to get familiar with how to configure the ports on the board and their modules which was necessary for many parts of the lab.

Logic analyzer:

The logic analyzer was used for most of the testing and precise measurements. First, it was used to test the ACLK. The ACLK signal was outputted to P4.2 and the frequency was measured with the logic analyzer. This allowed me to confirm the frequency was 32.72 kHz and decide what divisors to use in the timer module. The Logic analyzer was also used to test the ADC. Given analog signals, it was easy to see the conversion by attaching the logic analyzer to P2.4 where the converted signal was outputted. For the timers, the logic analyzer was a great way to visualize the PWM and get exact measurements.

Results and Performance:

After all the testing that went into each of the individual components the final project worked very well. After I had assembled all the components physically and put all the software together, I had a few small syntax bugs, but I faced no race conditions or logical errors in the code. Tilting the knob of the potentiometer in the x direction was indeed rotating the servo motor by about 45 degrees in each direction. The functionality was further confirmed by the logic analyzer. I used an adapter allowing the logic analyzer and servo to be attached to the same pin on the board and displayed the PWM. It was indeed a pulse of 20 ms with a duty cycle of about 5-10% (1-2 ms high). As shown in the Figure 3 and Figure 4 there was a slight error of about .2 ms, however the system still functioned as intended.

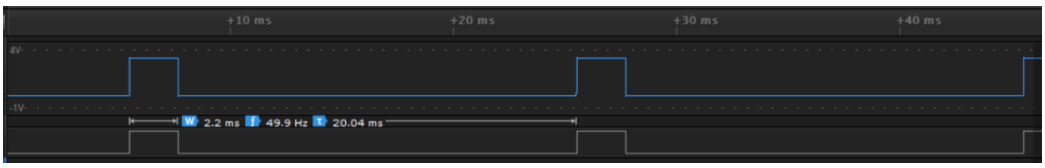

Figure 3: PWM on P2.4 (servo control) when potentiometer pushed in +x direction

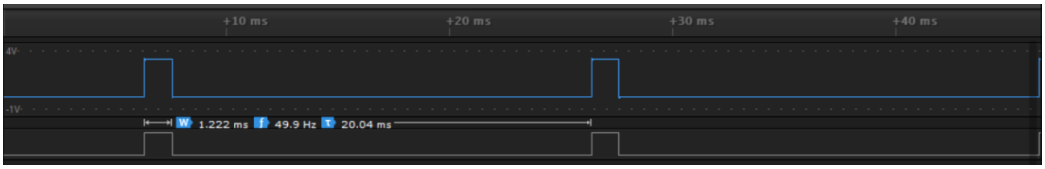

Figure 4: PWM on P2.4 (servo control) when potentiometer pushed in -x direction

Source Code:

```
#include "msp.h"

//file scope variable used by multiple functions
static unsigned clockfreq;

/* setup_ACLK:
 * Intializes and enables ACLK with VLOCLK to a freq of 32.720 kHz
 * Also outputs clk signal to pin P4.2
 * Returns: clock freq of 32,720 Hz
 */
unsigned setup_ACLK(){
    //set up clock -
    CS->CTL1 |=CS_CTL1_SELA__VLOCLK;
    CS->CTL1 |= CS_CTL1_DIVA__128;
    CS->CLKEN |= CS_CLKEN_ACLK_EN;
    //note with this config clk freq: 32.720 kHz (checked with logic analyzer)

    //set P4.2 to primary module ACLK
    P4->SEL1 &= ~0x04;
    P4->SEL0 |= 0x04;
    //set port 4.2 to output mode
    P4->DIR |= 0x04;
    return 32720;
}

/* setup_Timer:
 * Sets the timer_A0 using CCR0 and CCR1.
 * Generates a PWM of 20 ms using clock compare toggle/reset mode
 * Enables interrupts from timer CCR0
 * Ouputs pwm to port 2.4
 */
void setup_Timer(){
    double seconds = 20.0/1000;
    unsigned timer_ctl=0;
    unsigned P2_4 = 0x10;
    double duty_cycle=.9;

   //OUT1 is mapped TA0.1 which is primary module of port 2.4
    P2->SEL1 &= ~P2_4;
    P2->SEL0 |= P2_4; //select primary module
    P2->DIR |= P2_4; //output mode


    //TIMER_A0
    timer_ctl |= TIMER_A_CTL_SSEL__ACLK //set TASSEL, so it uses ACLK
        | TIMER_A_CTL_MC__UP //count up
        | TIMER_A_CTL_ID__8; // divide by 8

    TIMER_A0->CTL = TIMER_A_CTL_MC__STOP; //stop the timer (just set's to 0)
    TIMER_A0->CTL |= TIMER_A_CTL_CLR; //clear the clock
    TIMER_A0->CTL |= timer_ctl;

    //set up timer
```

```
    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_OUT //output to high
                | TIMER_A_CCTLN_CCIE //enable interrupts
                | TIMER_A_CCTLN_OUTMOD_0; //outmode 0-->does nothing
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CAP; //set to compare mode

    TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUT //output to high
                    | TIMER_A_CCTLN_CCIE //enable interrupts
                    | TIMER_A_CCTLN_OUTMOD_3 //outmode 3 set/reset
                    | TIMER_A_CCTLN_SCCI; // synchronize compare

    TIMER_A0->CCTL[1] &= ~TIMER_A_CCTLN_CAP; //set to compare mode

    TIMER_A0->CCR[0]=seconds*clockfreq/8;//just count up to the given seconds
    printf("CCR0: %d from sec: %d clockfreq: %d\n",TIMER_A0->CCR[0],seconds,clockfreq);


    //now use CCR1 for the duty cycle (percent high)
    TIMER_A0->CCR[1]= (1.0-duty_cycle)*seconds*clockfreq/8;//count up to duty cycle
    printf("CCR1: %d\n",TIMER_A0->CCR[1]);

    //these are needed to enable interrupts
        NVIC_EnableIRQ(TA0_0_IRQn);
        NVIC_SetPriority(TA0_0_IRQn,4);
}

/* setup_ADC
 * inputs the signal from port 4.0
 * uses ACLK for sampling and MEM register 0 for holding converted values
 */
void setup_ADC(){
    //select port 4.0 primary module input
    P4->DIR &= ~1UL; //0 input dir
    P4->SEL0 |= 1UL; //select 01 primary module
    P4->SEL1 &= ~1UL;

    setup_ACLK();
    ADC14->CTL0 &= ~ADC14_CTL0_ENC; //disable it
    ADC14->CTL0 = ADC14_CTL0_SSEL__ACLK; //select ACLK
    ADC14->CTL0 = ADC14_CTL0_ON; //turn it on lol

    ADC14->CTL1 = 0; //make sure all of these are zero

    ADC14->MCTL[0] = ADC14_MCTLN_INCH_13; //make the 0th register input from A13
(P4.0)
    ADC14->IER0 = 1UL;

    NVIC_EnableIRQ(ADC14_IRQn);
    NVIC_SetPriority(ADC14_IRQn,4);

   // ADC14->MEM[0]=0;

    ADC14->CTL0 |= ADC14_CTL0_ENC; //enable it
    //ADC14->CTL0 |= ADC14_CTL0_SC; //initiates conversion (done in timer interrupt)
}
```

```
/* ADC interrupt handler
 * Changes duty cycle of timer PWM
 */
void ADC14_IRQHandler(){
    unsigned val = ADC14->MEM[0];
    printf("Conversion is: %u\n",val); //can changes this to 2's complement too if
easier in setup

    //convert value from 0-255 to 1ms-2ms
    double ms = (double)val/255.0 + 1;

    //now CCR1 will be high for that amount of time
    //(do 20-ms because looking at pg 792 shows OUT1 is low for the amount of time in
CCR1)
    TIMER_A0->CCR[1]= (20-ms)/1000.0*clockfreq/8;//count up to duty cycle


    //printf("CCR1: %d\n",TIMER_A0->CCR[1]);

    //clear interrutp
    ADC14->CLRIFGR0 |= 1UL;
}


/* Timer interrupt handler --called every 20 ms
 * Starts ADC conversion
 */
void TA0_0_IRQHandler(){
    //can use this code to test interrupts

    //toggle the ADC14 SC bit (starts and stops the conversion)
    ADC14->CTL0 ^= ADC14_CTL0_SC;

    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; //clear flag
    //note we need to clear interrupt flag
    //TIMER_A0->CTL &= ~TIMER_A_CTL_IE;
}

/**
 * main.c
 */
void main(void)
{
      WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;         // stop watchdog timer

      clockfreq = setup_ACLK();
      setup_Timer();
      setup_ADC();
      //attach potentiometer to P4.0 (see beginning of setup_ADC())
      //now the pwm should be ouputted to P2.4 (see at beginning of setup_timer())
      //check with logic analyzer and then attach servo to P2.4
}
```