EE4930 Advanced Embedded Systems
Section 011, Winter 2019/20

Draven Schilling
12-15-19

Lab 2 – MSP432

Objectives

The objective for this lab was to gain further insight into working with some more of the onboard peripherals the MSP432 has to offer, specifically the Timer and ADC peripherals. Another goal was to familiarize myself with the interrupt structure the MSP432 uses and to gain a bit more experience using the MSOE library and HAL functions.

Description

In this lab I Initialized two TimerA peripherals, one to be used as a 1 second interrupt and another to be used in PWM mode to produce a square wave output**. I used a period of 1 second because** it's noticeably intentionally slow but also precise to show it was intentional. I also initialized an ADC in 12 bit mode connected to a potentiometer which would read the pot output and convert it to a percentage 0-99 based on how far the dial was turned to the right. The ADC was setup in single conversion mode where samples were triggered by the 1 second timer. When the ADC was done converting, it would produce its own interrupt which would then read the value and store it for other uses. Simultaneously, whenever the percentage changed, the inverse (100-value), was set to the duty cycle of the Timer running in PWM mode. Additionally, whenever the pushbutton 1.1 was pressed, the LCD would update the display with the last sampled value of the ADC along with the calculated duty cycle (non-inverted).

Conclusions

Again, everything in this lab was relatively straightforward. The posted example code provided a good starting point for which I could understand in the documentation why certain bits were being set. For the most part, all of my code worked with little debugging except for the PWM timer which I ran into some problems with some setup bits I missed first time around. I think I am getting familiar enough with the documentation now that I can generally find and interpret what I am looking for.

Attachments

The first 3 pictures are the PWM output on pin P2.5 at different potentiometer positions. The approximate expected duty cycle is represented in the figure caption.

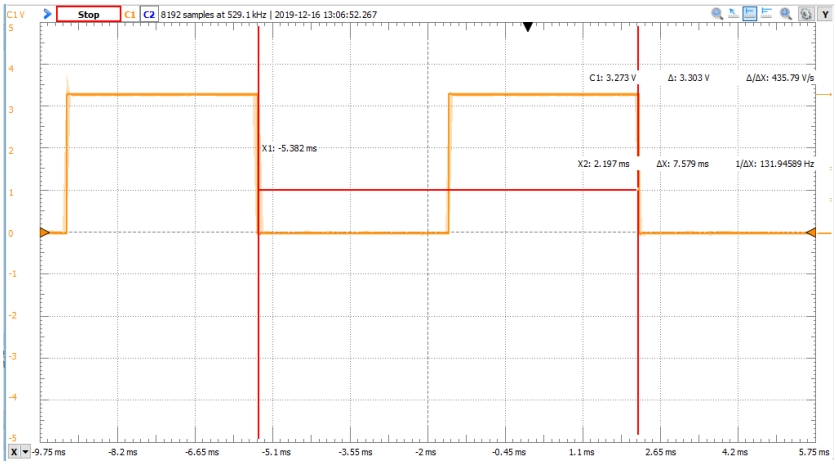


Figure 1 ~50% Duty Cycle

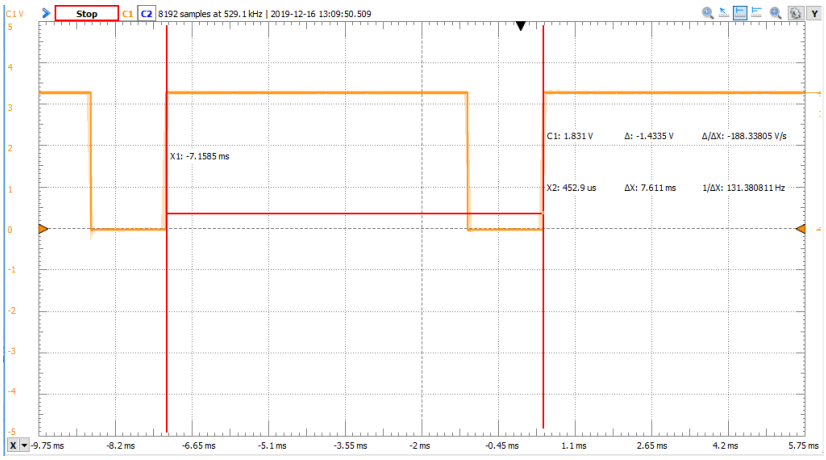


Figure 2 ~80% Duty Cycle

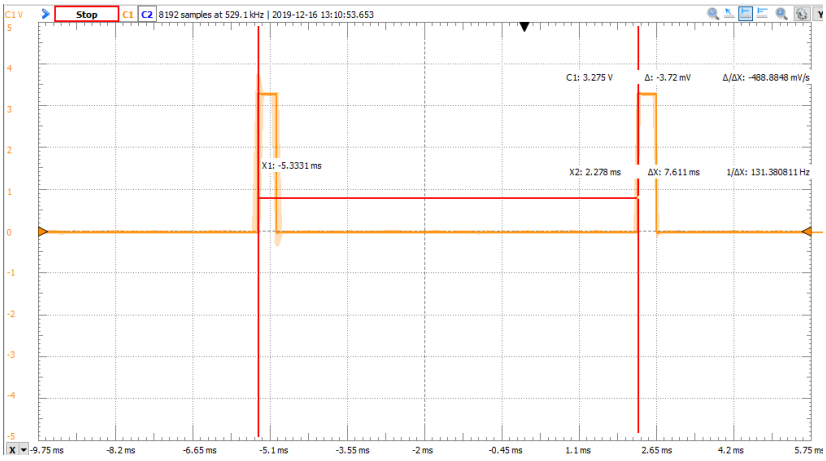
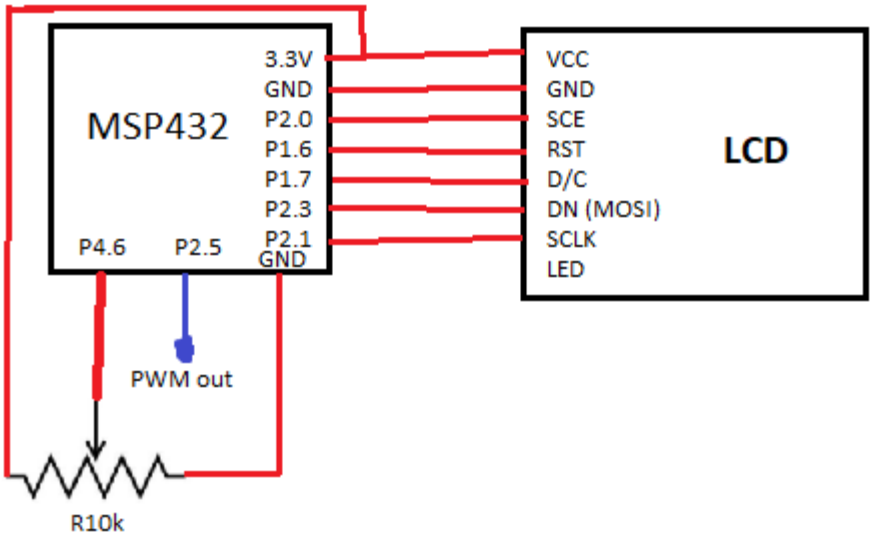


Figure 3 ~5% Duty Cycle

Wiring Schematic:



Source Code

```
#include "msp.h"
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include "msoe_lib_clk.h"
#include "msoe_lib_lcd.h"
#include "msoe_lib_delay.h"


/**
 * main.c
 */

#define ONE_HZ_PSC 46875 // for timers with 1/64 scale
#define PWM_PER 2865
#define ADC_RANGE 4096 // for 12 bit converstions

//Global Vars
int adc_val = 0;
uint8_t duty_cycle = 0;
```

```
void main(void)
{
       WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;          // stop watchdog timer
       //Clock_Init_48MHz();   // run system at 48 MHz (default is 3 MHz)

    // setup lcd
    LCD_Config();
    LCD_clear();
    LCD_home();
    LCD_contrast(10);

    //Setup Timers
    initTimA1();
    initTimA0();
    initADC();
    initPushbutton();

    char dc_display[8];
    uint8_t pushbutton;

    while(1){
        pushbutton = (P1->IN & 0b10) >> 1;

        if(!pushbutton){
            LCD_goto_xy(0,0);
            LCD_print_str("ADC:");
            LCD_print_udec5(adc_val);
            LCD_goto_xy(0,2);
            if(duty_cycle > 9)
                snprintf(dc_display, 8, "DC: %i%c", duty_cycle, '%');
            else
                snprintf(dc_display, 8, "DC: 0%i%c", duty_cycle, '%');
            LCD_print_str(dc_display);
        }
    }

}

// Initialize Output pin 2.5 to Timer A0 PWM
void initTimA0(){
    // set pin 2.5 to TimA0
    P2->SEL0 |= 1<<5;
    P2->SEL1 &= ~(1<<5);
    P2->DIR |= BIT5; // set pin 2.5 to output mode
    // Configure TimA0.2 to SMCLOCK, UP mode, Interrupt Enable on max
    TIMER_A0->CTL &= 0xFC00;
    TIMER_A0->CTL |= 0x0212;
    TIMER_A0->CCTL[2] |= TIMER_A_CCTLN_OUTMOD_7;  // Reset/set output mode
    TIMER_A0->CCR[0] = PWM_PER;  // set frequency as 1 kHz
    TIMER_A0->CCR[2] = ( PWM_PER * 50 ) / 100; // set duty cycle as 50% to start
    TIMER_A0->EX0 |= TIMER_A_EX0_IDEX__8;  // input clock factor of 8
    NVIC->ISER[0] |= (1<<9);  // enable TA0_N interrupt
}
```

```
//Initialize Timer A1 for 1s interrupt
void initTimA1(){
    // Configure TimA1 to SMCLOCK, UP mode, Interrupt Enable on max
    TIMER_A1->CTL &= 0xFC00;
    TIMER_A1->CTL |= 0x0212;
    TIMER_A1->CTL |= 0b11<<6; // add 1/8 divider
    TIMER_A1->CCR[0] = ONE_HZ_PSC;  // set frequency as 1 hz
    TIMER_A1->EX0 |= TIMER_A_EX0_IDEX__8;  // add another 1/8 divider
    NVIC->ISER[0] |= (1<<11);  // enable TA1_N interrupt
}

//Initialize ADC6 on pin 4.7
//Interrupt upon completed conversion
void initADC(){
    // start sampling on SC bit, source a timer, use SMCLK, single-channel single
conversion mode
    // 96 sample/hold time, turn core on
    ADC14->CTL0 &= 0x0;
    ADC14->CTL0 |= (1<<26) |  (1<<21) | (0b101<<12) | (0b101<<8) | (1<<4);
    // 12 bit resolution and use memory location 4 to start
    ADC14->CTL1 &= 0xF0000000;
    ADC14->CTL1 |= (0b10<<4) | (4<<16);
    ADC14->MCTL[4] |= 0x6;  // input on A6
    ADC14->IER0 |= (1<<4);  // enable interrupt for mem[4]
    ADC14->CTL0 |= 0b10;  // enable

    NVIC->ISER[0] |= (1<<24);  // enable ADC interrupt in NVIC
}

// initializes pushbutton P1.1 as GPIO input
void initPushbutton(void)
{
    P1->SEL0 &= ~0b10; // use GPIO function
    P1->SEL1 &= ~0b10;
    P1->DIR &= ~0b10;   // make input
    P1->REN |= 0b10; // allow pull up/down
    P1->OUT |= 0b10;  // setup as pull up
}

// Interrupt Handler for TA0 (PWM Timer)
void TA0_N_IRQHandler(void){
    uint16_t dummy = TIMER_A0->IV;
    //P4->OUT ^= BIT0;
}

// Interrupt Handler for TA1 (Queue ADC Timer)
void TA1_N_IRQHandler(void){
    uint16_t dummy = TIMER_A1->IV;
    ADC14->CTL0 |= 1; // start a new ADC conversion
}
```

```
// Interrupt Handler for ADC
// Read value and update TimA0.2 duty cycle
void ADC14_IRQHandler(void)
  {
      adc_val = ADC14->MEM[4];
      duty_cycle = (uint8_t)((((float)adc_val)/ADC_RANGE)*100);
      TIMER_A0->CCR[2] = ( PWM_PER * (100-duty_cycle) ) / 100; // invert duty cycle
for output
  }
```