EE4930 Advanced Microprocessor

Section 011/021, Winter 2022/23

Professor: Kerry R. Widder, Ph.D.

Electrical Engineering and Computer Science Department

Milwaukee School of Engineering

Laboratory 2: “MSP432”

Name: Jorge Jurado-Garcia

Courses: EE 4930 011

Date: 12/9/2022

# Objectives

This lab objectives are two gain experience with using the MSP432 microcontroller and its peripherals. For specifically using the Analog-to digital (A/D), Timer(s) (TimerA/Timer32), GPIO, NVIC peripherals. Afterwards an implementation of the LCD screen was done to show specific parameters described in the lab specifications document.

# Description

For this lab, I connected a potentiometer to 3.3V and ground and using a A0 read the voltage depending on the potentiometer resistance. The A/D peripheral was setup for single-channel-single conversion mode with 10-bits resolution and using the SMCLK. Interrupt handling was enabled for this peripheral and the ISER was also set. The A/D interrupt handler converted the A/D reading as a percentage and generated a PWM duty cycle based of the A/D percentage. This A/D peripheral was enabled by a Timer32 peripheral with a clock division of 256 size of a 16-bit counter. This allowed me to sample my ADC converter every 0.25 seconds. **I chose this period because its not to fast in which my interrupt will always be triggered and but also not slow enough in which new samples do come back to the adc converter**. **It is a nice medium in which my main can still run without any large errors arising.** Next a different TIMER was used to generate a PWM with UP MODE and generation. This timer was enabled on A2.3 with the highest number of ticks being at 6500. This enabled the PWM to have a maximum speed of 16.38m. An interrupt on the GPIO pin is enabled for updating the LCD screen.

# Conclusion

Overall, this lab was a success, and I was able to complete the needed objectives within time. I enjoyed figuring out what type of peripherals to use and how to implement them together in order to achieve a smooth program. Next time round I hope to use more of the extra registers that the GPIO, ADC, and Interrupt peripherals have and hopefully learn how to implement those in a different lab. Some struggles that I had when starting out is deciding where to start and how my code should be developed, after a while I ended up choosing that the best course of action was to start off with the TIMER32, ADC, PWM, GPIO, and then LCD screen. My reasoning with this is because it exactly follows chronical to the project specifications, and I can easily verify each peripheral first and their integration at each phase.

# Attachments

## Main

/\* Copyright (C) 2022 MSOE

\*

\* All Rights Reserved

\* You may not use, distribute or modify this code without the

\* express written permission of MSOE

\*

\* Contact Info

\* jorgejuradogarcia2@gmail.com

\* 608-312-5950

\*

\*/

**#include** <stdio.h>

**#include** "msp432.h"

**#include** "msoe\_lib\_all.h"

**#include** "defines.h"

// Lab modules

**#include** "EE4930\_LAB1.h"

**#include** "EE4930\_LAB2.h"

**#define** LAB1\_ON FALSE

**#define** LAB2\_ON TRUE

**#define** CLEAR 21

/\*

\* main.c

\*/

**unsigned** adc\_val;

**float** ADC\_Percentage;

**float** PWM\_dutycycle;

**char** LCD\_SET\_FLAG=0;

**int** Calculated\_dutycycle;

**int** **main**(**void**){

**#if** LAB1\_ON

Init\_Lab1();

**while**(1)

{

Lab1\_Poll();

}

**#endif**

**#if** LAB2\_ON

Clock\_Init\_48MHz();

Init\_Lab2();

NVIC->ISER[0] |= ( 1<<24 ); //NVIC for ADC14 at ISER[24]

NVIC->ISER[1] |= ( 1<<3 ); //NVIC for PORT 1 at ISER[35]

NVIC->ISER[0] |= ( 1<<25 ); //NVIC for TIMER\_32\_1 at ISER[25]

\_\_enable\_interrupt();

**while**(1)

{

**if**(LCD\_SET\_FLAG == TRUE)

{

LCD\_goto\_xy(5, 3); // start at row 3, column 0

LCD\_print\_udec5(adc\_val);

LCD\_goto\_xy(7, 4); // start at row 4, column 0

LCD\_print\_udec5((**int**)PWM\_dutycycle);

LCD\_SET\_FLAG = FALSE;

}

}

} // end main

// Interrupt Handler for ADC14

**void** **ADC14\_IRQHandler**()

{

**int** readIV = ADC14->IV; // Reading the IV register to clear

// read A/D result

adc\_val = ADC14->MEM[0];

ADC\_Percentage = 100 \* (**float**)adc\_val/1024;

PWM\_dutycycle = 100 - ADC\_Percentage;

//printf("A/D reading = %d, ADC % = %f \n", adc\_val, ADC\_Percentage);

//printf("Duty Cycle = %f\n", PWM\_dutycycle);

Calculated\_dutycycle = Set\_Duty( PWM\_dutycycle );

}

// Interrupt Handler for SW1

**void** **PORT1\_IRQHandler**()

{

**int** readIV = P1->IV; // Reading IV register to clear

// and set a LCD\_flag to start to update with new values

LCD\_SET\_FLAG = TRUE;

}

//Interrupt Handler for TIMRE32\_Int1

**void** **T32\_INT1\_IRQHandler**()

{

TIMER32\_1->INTCLR = CLEAR;

// start a new A/D conversion

ADC14->CTL0 |= ADC14\_CTL0\_SC;

//printf("starting ADC Timer\n");

}

**#endif**

## EE4030\_LAB2.h

/\*

\* EE4930\_LAB2.h

\*

\* Created on: Dec 1, 2022

\* Author: jurado-garciaj

\* Specifications:

\* Perform A/D conversion on a analog input with Potentiometer

\* A/D specs:

\* Single-channel single-conversion mode

\* 10-bit resolution

\* Use a timer and its interrupt to start a new A/D conversion

\* Use A/D interrupt for handling of new conversion values

\* Read A/D reading display the raw reading and duty cycle on LCD when SW1 is pressed

\* Use an Interrupt with P1.1 to start this action.

\*

\* Use a Timer to generate a PWM signal with a duty cycle being the percentage inverted

\* if the input is at 20%, set the duty cycle to 80%,

\* if at 0%, set the duty cycle to 30%,

\*

\* Initialize GPIO. Setup inputs and outputs as follows:

\*

\* Port pin in/out Pullup/down Connect

\* P1.1 In UP s1

\* P5.5 IN UP Pot

\* P6.6 Out N/A Logic Pro

\*

\*\*\*\*\*\*\*\*\*\*\*\* Nokia LCD interface reference \*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Red SparkFun Nokia 5110 (LCD-10168)

// -----------------------------------

// Signal (Nokia 5110) LaunchPad pin

// 3.3V (VCC, pin 1) power

// Ground (GND, pin 2) ground

// UCA3STE (SCE, pin 3) connected to P9.4

// Reset (RST, pin 4) connected to P9.3

// Data/Command (D/C, pin 5) connected to P9.2

// UCA3SIMO (DN, pin 6) connected to P9.7

// UCA3CLK (SCLK, pin 7) connected to P9.5

// back light (LED, pin 8) not connected, consists of 4 3.3 V white LEDs which draw ~80mA total

\*/

**#ifndef** APP\_EE4930\_LAB2\_H\_

**#define** APP\_EE4930\_LAB2\_H\_

**extern** **void** **Init\_Lab2**( **void** );

**extern** **int** **Set\_Duty**(**float** duty);

**#endif** /\* APP\_EE4930\_LAB2\_H\_ \*/

## EE4930\_LAB2.c

/\*

\* EE4930\_LAB2.h

\*

\* Created on: Dec 1, 2022

\* Author: Jorge Jurado-Garcia

\* Specifications:

\* Perform A/D conversion on a analog input with Potentiometer

\* A/D specs:

\* Single-channel single-conversion mode

\* 10-bit resolution

\* Use a timer and its interrupt to start a new A/D conversion

\* Use A/D interrupt for handling of new conversion values

\* Read A/D reading display the raw reading and duty cycle on LCD when SW1 is pressed

\* Use an Interrupt with P1.1 to start this action.

\*

\* Use a Timer to generate a PWM signal with a duty cycle being the percentage inverted

\* if the input is at 20%, set the duty cycle to 80%,

\* if at 0%, set the duty cycle to 30%,

\*

\* Use a Scope to see the PWM wave or a LED as a visual indication

\*

\* Initialize GPIO. Setup inputs and outputs as follows:

\*

\* Port pin in/out Pullup/down Connect

\* P1.1 In UP s1

\* P5.5 IN NONE A0

\* P6.6 OUT N/A Logic Pro/TA23

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\* Nokia LCD interface reference \*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Red SparkFun Nokia 5110 (LCD-10168)

// -----------------------------------

// Signal (Nokia 5110) LaunchPad pin

// 3.3V (VCC, pin 1) power

// Ground (GND, pin 2) ground

// UCA3STE (SCE, pin 3) connected to P9.4

// Reset (RST, pin 4) connected to P9.3

// Data/Command (D/C, pin 5) connected to P9.2

// UCA3SIMO (DN, pin 6) connected to P9.7

// UCA3CLK (SCLK, pin 7) connected to P9.5

// back light (LED, pin 8) not connected, consists of 4 3.3 V white LEDs which draw ~80mA total

\*

\* We will be using TIMER\_32 to START ADC CONVERTER after the timer expires and reaches zero

\* My idea is to generate start the ADC converter every second this will have to be based on the CPU CLock

\* Reasoning on why to use TIMER\_32 in order to start the ADC converter is for its simplicity and easy to use.

\* There is no need for to make it make for complex then it needs to be

\* 48 MHZ per tick therefore for one second we will need

\* 1/ (48 MHZ/256) Times per tick

\* on a 32 bit Timer time after overflow is: 0.25 seconds almost 0.5 second

\*

\*

\*

NOTE:

An interrupt is generated when the full 32-bit counter reaches zero and is only cleared when the

T32INTCLRx register is written to. A register holds the value until the interrupt is cleared. The most

significant carry bit of the counter detects the counter reaching zero.

The interrupts can be masked by writing 0 to the Interrupt Enable bit in the T32CONTROLx register. Both

the raw interrupt status, prior to masking, and the final interrupt status, after masking, can be read from

status registers. The interrupts from the individual counters, after masking, are logically ORed into a

combined interrupt, TIMINTC, provides an additional interrupt condition from the Timer32 peripheral. Thus,

the module supports three interrupts in total – TIMINT1, TIMINT2, and TIMINTC.

\*

\*

\*

\* Set master clock (MCLK) to HFXTCLK with divide by 1 - 48MHz

\* Set high speed sub-system clock (HSMCLK) to HFXTCLK with divide by 2 - 24MHz (max allowed)

\* Set low speed sub-system clock (SMCLK) to HFXTCLK with divide by 4 - 12MHz (max allowed)

\* Set the auxiliary clock (ACLK) to REFOCLK - 32KHz

\* Set the backup clock (BCLK) to REFOCLK - 32KHz

\*

\*/

// Included files

**#include** <stdio.h>

**#include** "msp432.h"

**#include** "msoe\_lib\_all.h"

**#include** "defines.h"

// Drivers

**#include** <MSP432P401R\_GPIO.h>

**void** **Init\_\_Timer32**( **void** );

**void** **Init\_A2D**( **void** );

**void** **Init\_TimerA**( **void** );

**int** **Set\_Duty**(**float** duty);

**void** **Init\_Lab2**( **void** )

{

Stop\_watchdog(); // stop Watch dog timer

Set\_ports\_to\_out(); //sets all the ports to outputs to prevent floating inputs

LCD\_Config(); // Configures the LCD display screen clears display afterwards

// SW1

GPIO\_setAsInputPinWithPullUpResistor( GPIO\_PORT\_P1, GPIO\_PIN1 ); //Set P1.1 as an Input with Pull up resistor high

P1->IES &= ~GPIO\_PIN1; // enable edge select

P1->IE |= GPIO\_PIN1; // enable interrupt

//PWM PINS

GPIO\_setAsOutputPin( GPIO\_PORT\_P6, GPIO\_PIN6); // P6.6 as an output pin

// input on A0 into MEM0

P6->SEL0 |= GPIO\_PIN6; // use with TA0.2

P6->SEL1 &= ~GPIO\_PIN6; // use with TA0.2

//A0

GPIO\_setAsInputPin(GPIO\_PORT\_P5, GPIO\_PIN5); //Set P5.5 as input for A0

// input on A0 into MEM0

P5->SEL0 |= BIT5; // use with A0

P5->SEL1 |= BIT5; // use with A0

//Printing "EE4930-011" on the first line of the LCD screen

LCD\_goto\_xy(0,1);

LCD\_print\_str("EE4930-011");

LCD\_goto\_xy(0,2);

LCD\_print\_str("Lab 2 ");

LCD\_goto\_xy(0,3);

LCD\_print\_str("ADC:");

LCD\_goto\_xy(0,4);

LCD\_print\_str("DUTY %:");

Init\_A2D();

Init\_TimerA();

Init\_\_Timer32();

}

**void** **Init\_TimerA**( **void** )

{

// SET the TIMER in UP MODE WITH RESER/SET MODE

// CLOCK SOURCE SMCLK 3MHZ

// DIVIDE by 8

// Mode Control: Up Mode

// Interrupt for right now disabled

TIMER\_A2->CTL |= (TIMER\_A\_CTL\_SSEL\_\_SMCLK|TIMER\_A\_CTL\_ID\_0|TIMER\_A\_CTL\_MC\_\_UP);

//TA0.2

// Set output mode to Reset and SET

TIMER\_A2->CCTL[3] |= (TIMER\_A\_CCTLN\_OUTMOD\_7);

// Capture Control register to TIMER\_Ax Capture.Compare

//12 MHZ /3 = 4,000,000 input frequency

// T\_timer = 1/f\_systemclock = 1/4 MHz = 0.25 uS

// CTL Frequency = 100K HZ frequency

// If using the full 16-bit timer we have 65535 ticks

// For maximum speed before tick overflow we will have

// t\_overflow = tick\_max \* T\_timer = 65535 \* 0.25 us = 16.38375 mS

//This is fast be good enough for our info

TIMER\_A2->CCR[0] = 6500;

//change this file in the Capture control register to change duty cycle

TIMER\_A2->CCR[3] = 3250;

}

**void** **Init\_\_Timer32**( **void** )

{

// Set master clock (MCLK) to HFXTCLK with divide by 1 - 48MHz

TIMER32\_1->LOAD |= 46875; //Value in which timer will get reloaded

/\*

\* TIMER CLOCK SPEED: 48MHz MHz

\* TImer disabled

\* Periodic mode Enabled

\* Timer Interrupt enabled

\* Clock division of 256

\* Clock After Division in 48MHz/256 = 0.1875 MHz = 187,500 Hz

\* Size of 16-bit counter

\* Ticks 16 bits = 65535 ticks

\* Timer\_sec = 1/187500 = 5.333\*10^(-6)

\* Timer After expiration = 5.333\*10^(-6) \* 46875 = 0.25 seconds

\* Wrapping mode

\* 32-bit counter

\*/

/\*

\*

\*/

TIMER32\_1->CONTROL |= 0x6A;

// ENABLE TIMER

TIMER32\_1->CONTROL |= 0x80;

}

**void** **Init\_A2D**( **void** )

{

// Setting up the control register

ADC14->CTL0 |= ADC14\_CTL0\_SHT0\_5 | ADC14\_CTL0\_SHP | ADC14\_CTL0\_SSEL\_\_SMCLK | ADC14\_CTL0\_ON | ADC14\_CTL0\_CONSEQ\_0;

// Sampling time, S&H=96, ADC14 on, SMCLK, with 10 Bit Resolutions

// make sure read the reset operation first

// CSTARTADDx this gets the start address, these bits select which ADC14 conversion

// memory registers is used for a single conversion or for the first conversion in a sequence.

// setting the memory to MEM0

ADC14->CTL1 &= ~ADC14\_CTL1\_RES\_2;

ADC14->CTL1 |= (0 << ADC14\_CTL1\_CSTARTADD\_OFS);

//Memory conversion control 0 register being set to get data for A0

ADC14->MCTL[0] |= ADC14\_MCTLN\_INCH\_0;

// Enabled interrupt information for IER0 register

ADC14->IER0 |= ADC14\_IER0\_IE0;

// Interrupt enable 1 register information with overflow-interrupt enabled

//ADC14->IER1 |= ADC14\_IER1\_OVIE;

// Interrupt Vector Register for MEM0 0Ch

ADC14->IV |= 0x0C;

// enable the conversion

ADC14->CTL0 |= ADC14\_CTL0\_ENC;

}

**int** **Set\_Duty**(**float** duty){

//Duty 0-100

//will need to grab current CCR[0] register value

//and also grab the its current duty cycle also

//current CCR[0] value:

**int** current\_val = TIMER\_A2->CCR[0];

//which direction

//pwmA which is P2.6 or PWMA

**float** new\_val = (**float**)( current\_val\*(duty/100) );

**printf**("Inserted CCR3 value for timer %d\n", (**int**)new\_val);

TIMER\_A2->CCR[3] = (**int**) new\_val;

**return** (**float**)(100\*new\_val)/current\_val;

}

## Schematic

Chart, diagram

Description automatically generated

## PWM OUTPUT

A picture containing text, electronics, scoreboard

Description automatically generated

A picture containing graphical user interface

Description automatically generated