Laboratory Activity 1: MSP432 Microcontroller

Embedded Systems

Karen Andrea Flores Yánez

Contents

List of Figures

List of Tables

Introduction

The aim of this laboratory activity is to understand how the different programmable interfaces of the MSP-EXP432P401R LaunchPad^{TM} work. For this purpose, a program has been implemented that allows to the microcontroller, MSP432P401R, generates a Pulse Width Modulated (PWM) signal with a width that is proportional to an analog input signal.

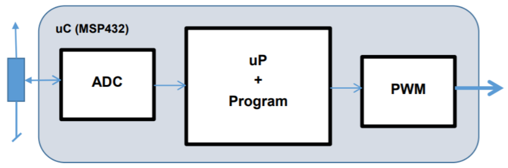


Figure 1: General system block schematic

The hardware needed to carry out this project were: MSP-EXP432P401R LaunchPad^{TM}[1], Keyes_SJoys joystick and MG90S micro servo motor[2].

The connections between the different devices is shown in the **Figure 2**

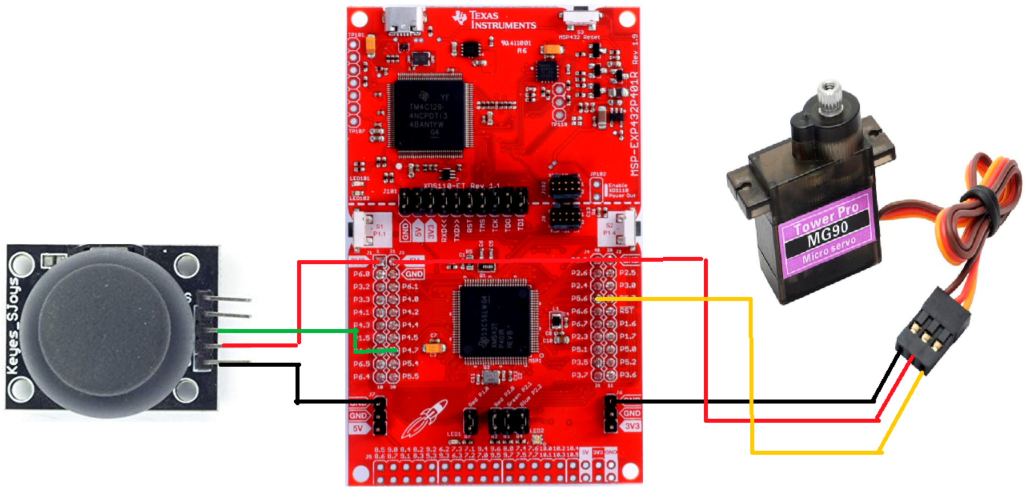


Figure 2: Hardware interconnections

A summary of the pinout used in this activity is shown in **Table 1**.

Pin	Function
P4.7	Analog input
P5.6	Digital output

Table 1. Pinout used

Finally, the Integrated Development Environment (IDE) used to program the board was Code Composer Studio v8.2.0. This IDE comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow.

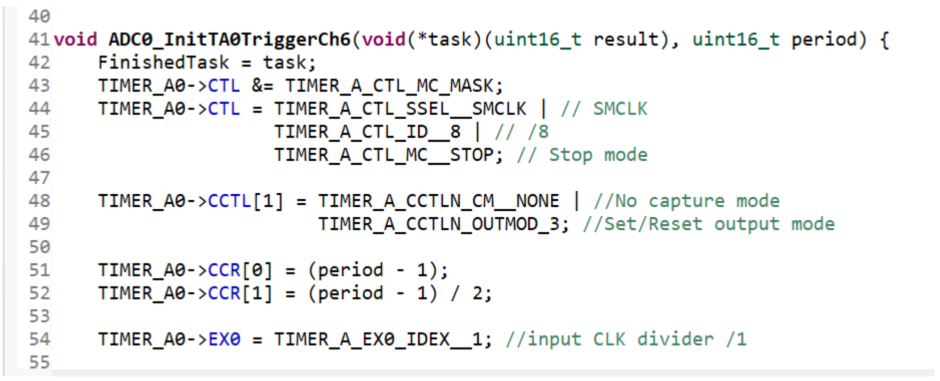
1. Implementation

1.1 Clock System

First of all, it was very important to know the clock frequency. As I readed in the datasheet [1], the default clock frequency is $f_{clk} = 1.5MHz$. I decided to work with tha value because I did not have any requirements related to the speed or low power consumption.

1.2 Periodic capture of the analog signal from the joystick

I decided to use the **Timer A0** to generate periodical interrupts that enables the ADC converter to start a new conversion of the analog signal generated by the joystick. The Timer A0 configuration is shown in the **Figure 1.1**.



```c

void ADC0_InitTA0TriggerCh6(void(*task)(uint16_t result), uint16_t period) {
    FinishedTask = task;
    TIMER_A0->CTL &= TIMER_A_CTL_MC_MASK;
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK
                    TIMER_A_CTL_ID__8 | // /8
                    TIMER_A_CTL_MC__STOP; // Stop mode

    TIMER_A0->CCTL[1] = TIMER_A_CCTLN_CM__NONE | //No capture mode
                        TIMER_A_CCTLN_OUTMOD_3; //Set/Reset output mode

    TIMER_A0->CCR[0] = (period - 1);
    TIMER_A0->CCR[1] = (period - 1) / 2;

    TIMER_A0->EX0 = TIMER_A_EX0_IDEX__1; //input CLK divider /1

```

Figure 1.1. Timer A0 configuration

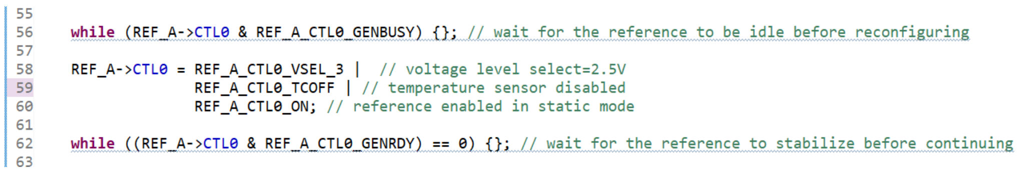
I used the SMCLK clock whose frequenzy, as I said before, is by default $1.5MHz$. Then I divided that value by 8, obtaining a clock resolution:

$$Resolution(\mu s) = \frac{8}{1.5Mhz} = 5.33\mu s \tag{1.1}$$

The **Timer A0** is configured in **UP mode** (last line of code of the **Figure 1.3**) and it enables to start a new conversion in the ADC each ~50ms.

Previous to configure de ADC, I have used the Reference Module, **Reference_A**, a general-purpose reference system that is used to generate voltage references required for other analog modules available on a given device such as ADCs, DACs, comparators, or LCD controller.

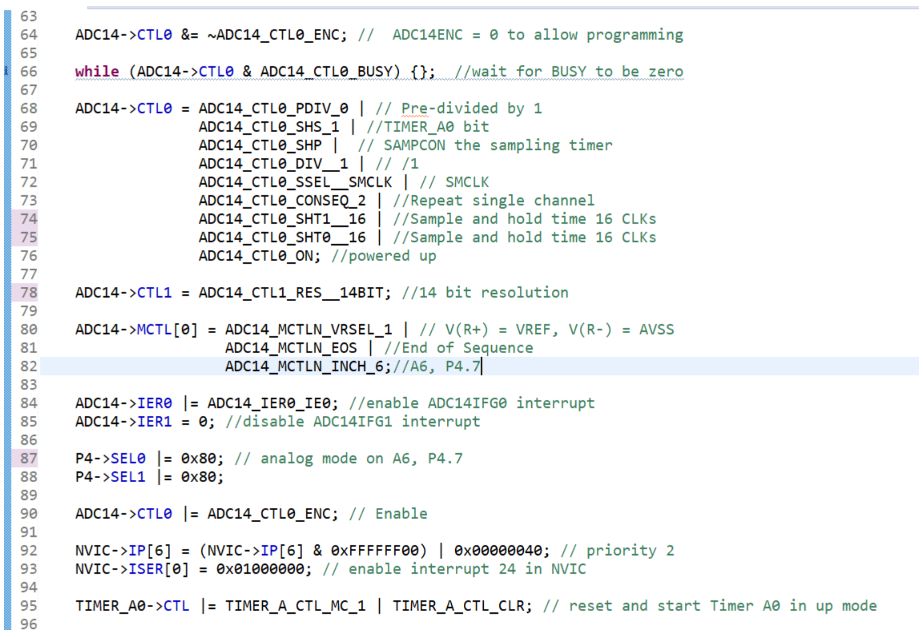
The reference voltage selected was 2.5 V and the temperature sensor connected directly to the ADC was disabled because I did not use it in my project.



```
while (REF_A->CTL0 & REF_A_CTL0_GENBUSY) {}; // wait for the reference to be idle before reconfiguring

REF_A->CTL0 = REF_A_CTL0_VSEL_3 |  // voltage level select=2.5V
              REF_A_CTL0_TCOFF | // temperature sensor disabled
              REF_A_CTL0_ON; // reference enabled in static mode

while ((REF_A->CTL0 & REF_A_CTL0_GENRDY) == 0) {}; // wait for the reference to stabilize before continuing
```

Figure 1.2. REF_A configuration

To convert the analog input signal of the joystick I used the **channel 6** of the **ADC**. It generates an interrupt by itself each time it hasconverted the sampled value of the joystick. The ADC configuration is showed in **Figure 1.3**.

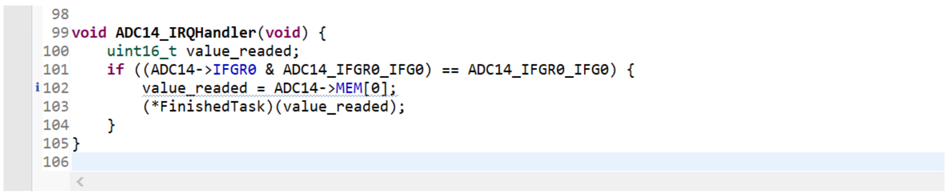


```
ADC14->CTL0 &= ~ADC14_CTL0_ENC; //  ADC14ENC = 0 to allow programming

while (ADC14->CTL0 & ADC14_CTL0_BUSY) {};  //wait for BUSY to be zero

ADC14->CTL0 = ADC14_CTL0_PDIV_0 | // Pre-divided by 1
              ADC14_CTL0_SHS_1 | //TIMER_A0 bit
              ADC14_CTL0_SHP |  // SAMPCON the sampling timer
              ADC14_CTL0_DIV__1 | // /1
              ADC14_CTL0_SSEL__SMCLK | // SMCLK
              ADC14_CTL0_CONSEQ_2 | //Repeat single channel
              ADC14_CTL0_SHT1__16 | //Sample and hold time 16 CLKs
              ADC14_CTL0_SHT0__16 | //Sample and hold time 16 CLKs
              ADC14_CTL0_ON; //powered up

ADC14->CTL1 = ADC14_CTL1_RES__14BIT; //14 bit resolution

ADC14->MCTL[0] = ADC14_MCTLN_VRSEL_1 | // V(R+) = VREF, V(R-) = AVSS
                 ADC14_MCTLN_EOS | //End of Sequence
                 ADC14_MCTLN_INCH_6;//A6, P4.7

ADC14->IER0 |= ADC14_IER0_IE0; //enable ADC14IFG0 interrupt
ADC14->IER1 = 0; //disable ADC14IFG1 interrupt

P4->SEL0 |= 0x80; // analog mode on A6, P4.7
P4->SEL1 |= 0x80;

ADC14->CTL0 |= ADC14_CTL0_ENC; // Enable

NVIC->IP[6] = (NVIC->IP[6] & 0xFFFFFF00) | 0x00000040; // priority 2
NVIC->ISER[0] = 0x01000000; // enable interrupt 24 in NVIC

TIMER_A0->CTL |= TIMER_A_CTL_MC_1 | TIMER_A_CTL_CLR; // reset and start Timer A0 in up mode
```

Figure 1.3. ADC configuration

As I used the pin P4.7 for the analog input, I programmed the ADC to make a sequence of conversions of that channel. I decided to use the full resolution of the ADC (14 bits) and a sample and hold time of 16 clock cycles, long enough for the input to stabilize.

When the ADC has finished the conversion, an interrupt is generated and the value readed is stored in the register MEM0 as you can see in the **Figure 1.4**.

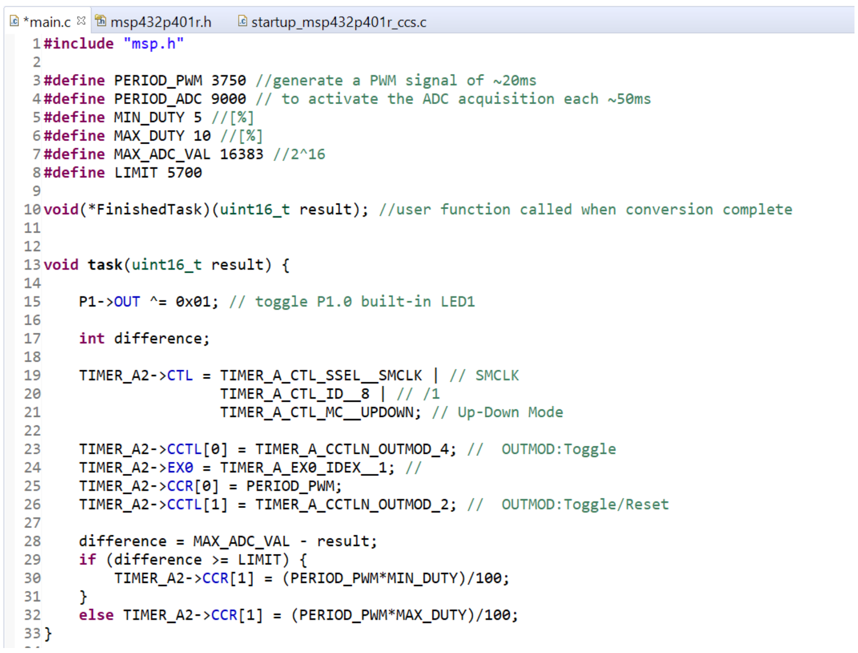


```c

void ADC14_IRQHandler(void) {
    uint16_t value_readed;
    if ((ADC14->IFGR0 & ADC14_IFGR0_IFG0) == ADC14_IFGR0_IFG0) {
        value_readed = ADC14->MEM[0];
        (*FinishedTask)(value_readed);
    }
}

```

Figure 1.4. ADC IRQ Handler

1.3 Pulse Width Modulation signal generated

Once the ADC conversion is obtained, the valued readed is passed to a **task** to generate a PWM signal whose duty cycle depends on that value readed. To generate this signal I used the **Timer A2**, its configuration is showed in the **Figure 1.5**.



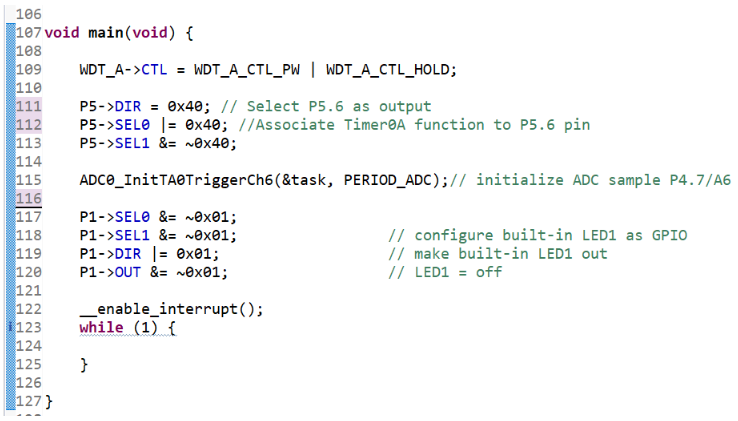
```c
#include "msp.h"

#define PERIOD_PWM 3750 //generate a PWM signal of ~20ms
#define PERIOD_ADC 9000 // to activate the ADC acquisition each ~50ms
#define MIN_DUTY 5 //[%]
#define MAX_DUTY 10 //[%]
#define MAX_ADC_VAL 16383 //2^16
#define LIMIT 5700

void(*FinishedTask)(uint16_t result); //user function called when conversion complete


void task(uint16_t result) {

    P1->OUT ^= 0x01; // toggle P1.0 built-in LED1

    int difference;

    TIMER_A2->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK
                    TIMER_A_CTL_ID__8 | // /1
                    TIMER_A_CTL_MC__UPDOWN; // Up-Down Mode

    TIMER_A2->CCTL[0] = TIMER_A_CCTLN_OUTMOD_4; //  OUTMOD:Toggle
    TIMER_A2->EX0 = TIMER_A_EX0_IDEX__1; //
    TIMER_A2->CCR[0] = PERIOD_PWM;
    TIMER_A2->CCTL[1] = TIMER_A_CCTLN_OUTMOD_2; //  OUTMOD:Toggle/Reset

    difference = MAX_ADC_VAL - result;
    if (difference >= LIMIT) {
        TIMER_A2->CCR[1] = (PERIOD_PWM*MIN_DUTY)/100;
    }
    else TIMER_A2->CCR[1] = (PERIOD_PWM*MAX_DUTY)/100;
}
```

Figure 1.5. Timer A2 configuration

Timer A2 also uses the SMCLK and the same divisor (/8) but now it is configured in **Up-Down** count mode. The value saved in the CCR0 is the period of the PWM signal, $\sim 20ms$, meanwhile the value saved in the CCR1 is the period in which the pwm signal is on and its value depends on the conversion done by the ADC.

1.4 Demostration

Taking into account all the previous steps in the **Figure 1.6** it is shown the main function of my program.



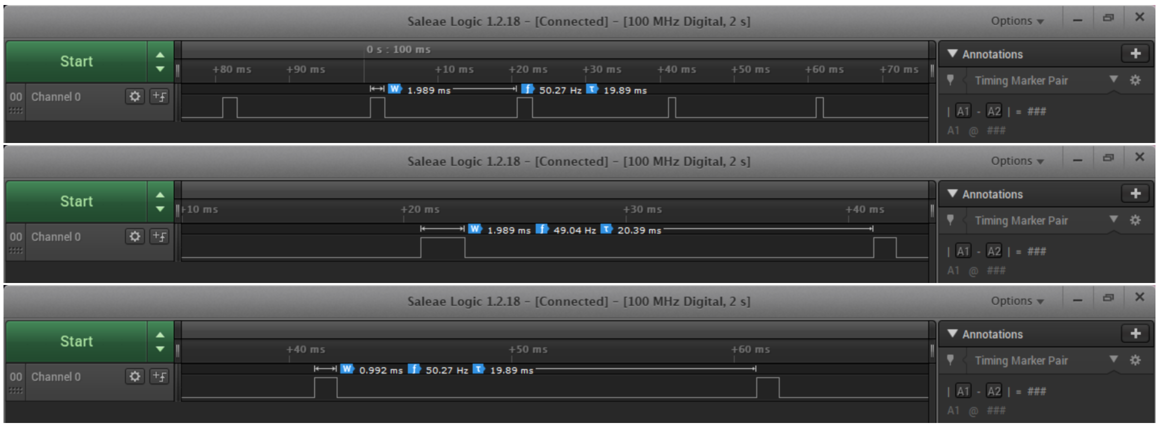
```

void main(void) {

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;

    P5->DIR = 0x40; // Select P5.6 as output
    P5->SEL0 |= 0x40; //Associate Timer0A function to P5.6 pin
    P5->SEL1 &= ~0x40;

    ADC0_InitTA0TriggerCh6(&task, PERIOD_ADC);// initialize ADC sample P4.7/A6

    P1->SEL0 &= ~0x01;
    P1->SEL1 &= ~0x01;                 // configure built-in LED1 as GPIO
    P1->DIR |= 0x01;                   // make built-in LED1 out
    P1->OUT &= ~0x01;                  // LED1 = off

    __enable_interrupt();
    while (1) {

    }

}
```

Figure 1.6. Main function

The values of the macros **PERIOD_PWM** and **PERIOD_ADC** are the values stored in the CCRx timers' registers. They have been calculated using the following equation:

$$(T_{signal} - 1) = Resolution \times CCRx \tag{1.2}$$

Using the logical analyzer I checked if the signal generated was the desired one and I obtained the following results:



```
Saleae Logic 1.2.18 - [Connected] - [100 MHz Digital, 2 s]
W 1.989 ms   f 50.27 Hz   T 19.89 ms

Saleae Logic 1.2.18 - [Connected] - [100 MHz Digital, 2 s]
W 1.989 ms   f 49.04 Hz   T 20.39 ms

Saleae Logic 1.2.18 - [Connected] - [100 MHz Digital, 2 s]
W 0.992 ms   f 50.27 Hz   T 19.89 ms
```

Figure 1.7. Logic analyzer results

The result shows us when the joystick is pressed it generates a signal are summarized in the **Table 1.1**

	Joystick ON	Joystick OFF
T_ON	1.989 ms	0.992 ms
T	20.39 ms	19.89 ms
Duty Cycle	10%	5%

Table 1.1. Results obtained

After seeing the results obtained, I can conclude this report by saying that I have achieved the proposed objectives.

Bibliography

[1] Texas Instrument. Msp432p401r simplelink^{TM} microcontroller launchpad^{TM} development kit (msp-exp432p401r) user guide. http://www.ti.com/lit/ug/slau597f/slau597f.pdf.

[2] Metal gear servo. https://engineering.tamu.edu/media/4247823/ds-servo-mg90s.pdf.