Embedded Systems

Lab 1.0

Richard Ismer

Dimitri Jacquemont

August 31, 2023

1 Problem Statement

This lab consists in modulating a PWM with the result of an ADC conversion. A joystic acting as a variable resistor is fed to the ADC, which computes a value based of the joystic's orientation. This value is processed and a new PWM value (T_{on}) is computed by the MCU. The PWM modulated by this conversion is then used to drive a servo motor (from $0°$ to $90°$). For this project, the MSP432P401R development board is used, a joystic (variable resistor) and servo-motor.

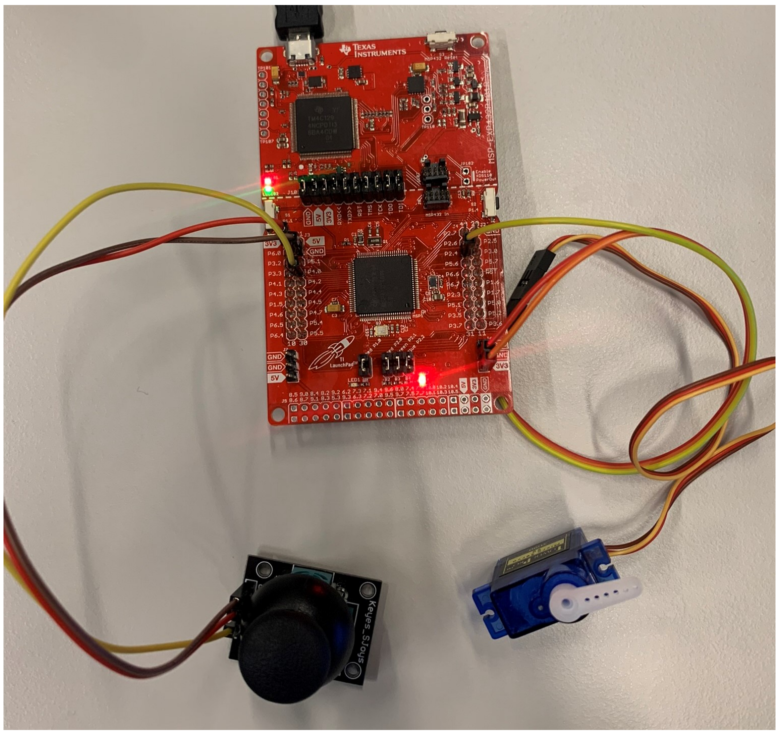


Figure 1: Project's wiring and connection

2 High-level Description

For this project we will be using a joystick as our input and a servo as our output. To process the signal from the joystick we will be using the 14 bit native ADC (Analog to Digital converter) installed on the board to be able to use this signal later on as a digital signal. To create the PWM (Pulse Width Modulated signal) we will be using a first timer (namely Timer A0) with 2 CCR (Compare Capture Register). We choose this timer because it could be bound with the output pins easily. A second timer (Timer A1) will be used to trigger a conversion from the ADC. Both the ADC and the second timer will have interrupt enable. We can see on the following picture the global view of the different peripherals used in this lab.

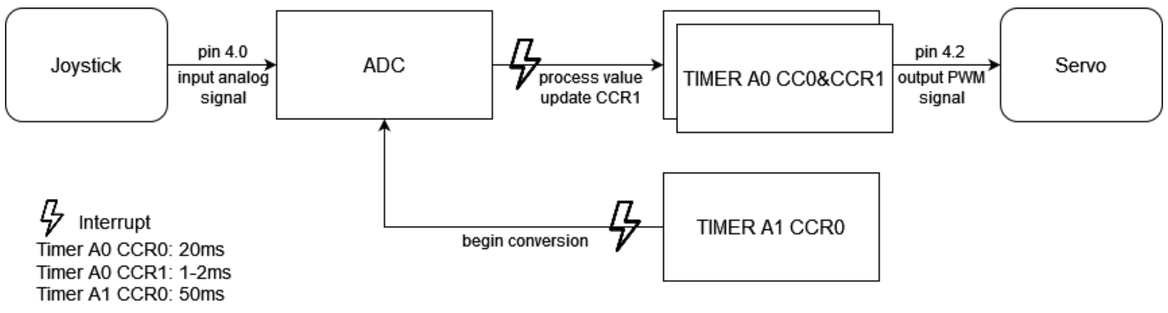


Figure 2: Global view of the peripherals and their interaction

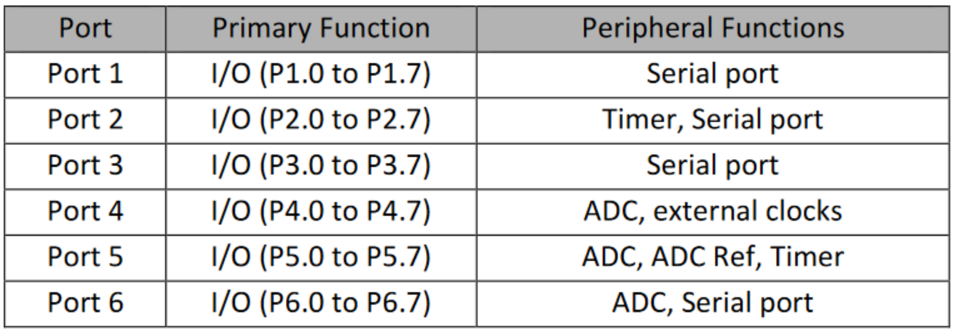
3 Low-level Description

3.1 PIN I/O

The board we are using has 6 GPIO ports that can be configured for different peripherals. Each port has 8 pins that can be configured independently. In our case we will need 2 pins:

- The first one for the output PWM connected to the servo
- The second one for the input signal of the joystick

As shown in the figure below, the timers can be bound to port 2 and the ADC to port 4. These are the 2 ports we will be using in this lab. But not each timer's CCR (see further section) can be bound with every pin of port 2. We must chose carefully which CCR to chose and to which pin it will be bound. The ports P2.4 to P2.7 can be respectively bound with the timers A0.1 to A0.4 (which correspond to CCR1 to CCR4 of timer A0).



Port	Primary Function	Peripheral Functions
Port 1	I/O (P1.0 to P1.7)	Serial port
Port 2	I/O (P2.0 to P2.7)	Timer, Serial port
Port 3	I/O (P3.0 to P3.7)	Serial port
Port 4	I/O (P4.0 to P4.7)	ADC, external clocks
Port 5	I/O (P5.0 to P5.7)	ADC, ADC Ref, Timer
Port 6	I/O (P6.0 to P6.7)	ADC, Serial port

TABLE 1. *PRIMARY AND SECONDARY FUNCTIONS FOR MSP432P401R PORTS*

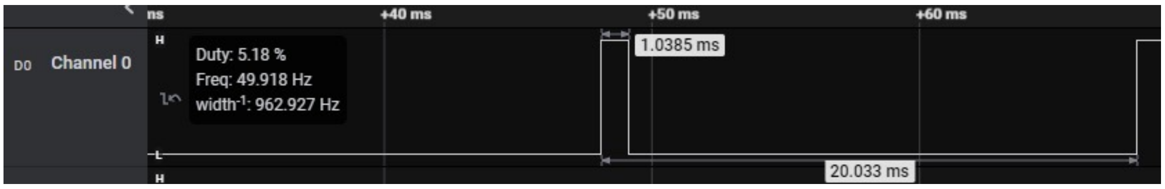
3.2 Clock

For this Lab we had to choose a clock for our peripherals. There are 7 different sources that can be chosen and that will produce a clock signal. We decided to chose the LFXTCLK that has a frequency of 32 768Hz since there was no specific clock requirements for this project. A high frequency clock is not needed to achieve a good precision (as we will see in a further section).

To be able to use the LFXTCLK, it is wired to the SMCLK (Sub-system Master CLocK) by selecting LFXTCLK as its source and by enabling the SMCLK so that the peripherals can use it. The SMCLK has no divider enabled, the output signal is the same as the source (32 768Hz).

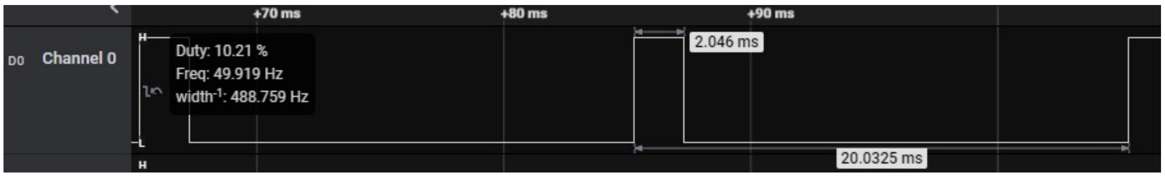
3.3 Timer A0

Timer A0 is used for the PWM generation driving the servo motor. This PWM needs to have a period of 20ms, and a $1ms \leq T_{on} \leq 2ms$.



Duty: 5.18 %
Freq: 49.918 Hz
width^{-1}: 962.927 Hz

Figure 3: PWM with T_{on} at 1ms



Duty: 10.21 %
Freq: 49.919 Hz
width^{-1}: 488.759 Hz

Figure 4: PWM with T_{on} at 2ms

The timer first takes SMCLK as its clock's input with the setting bits TASSEL. In order to yield maximum precision, no prescaler are being used. CCR0 and CCR1 are in charge of the timer's overflow (CCR0) and capture and compare value (CCR1).

The values for CCR1 and CCR0 are timer counts and need to be computed. The following formula is used.

$$time\,[s] = \frac{prescaler \cdot timerCnt}{frequency\,[Hz]} \qquad \Leftrightarrow \qquad timerCnt = \frac{time\,[s] \cdot frequency\,[Hz]}{prescaler}$$

For this application, 20ms for CCR0 and 1ms for CCR1 are required, and $CCR0 = 655$ as well as $CCR1 = 32$ are obtained. CCR1 will later be modified from the ADC interrupt to modulate the PWM signal and thus turn the servo-motor. The signal has to be high for $0 \leq CNT < CCR1$ and low for $CCR1 \leq CNT < CCR2$. The OUTMOD 6 (Toggle/Set) was thus selected. CCR1 is directly connected to the output port P2.4, and the signal is output once the pin's register are set (output).

```
void setupTimerA0(void) {
    // setup the timer A0
    TIMER_A0->CTL |= TIMER_A_CTL_SSEL__SMCLK;
    TIMER_A0->CTL |= TIMER_A_CTL_MC_1;

    // set the value for CCR0
    TIMER_A0->CCR[0] = count_to_milli(20);
    // set the value for CCR1
    TIMER_A0->CCR[1] = count_to_milli(1);

    // set the outmode for the timer
    TIMER_A0->CCTL[1] |= TIMER_A_CCTLN_OUTMOD_6;
}
```

3.4 TIMER A1

This second timer triggers an interrupt starting the ADC conversion every 50ms. Once again, this timer takes in SMCLK and no prescaler is applied on it. It is set to count upward until it reaches CCR0, at which point it overflows and restart from 0. Using the same formula as before, $CCR0 = 1638$ is obtained since we want a period of 50ms. This timer's overflow interrupt is then enabled.

```
void setupTimerA1(void) {
    // setup the timer A0
    TIMER_A1->CTL |= TIMER_A_CTL_SSEL__SMCLK;
    TIMER_A1->CTL |= TIMER_A_CTL_MC_1;

    // set the value for CCR0
    TIMER_A1->CCR[0] = count_to_milli(50);

    // enable interrupt for timer
    NVIC_EnableIRQ(TA1_0_IRQn);
    TIMER_A1->CCTL[0] = TIMER_A_CCTLN_CCIE;
}
```

In this timer's interrupt is the ADC conversion triggered.

```
void TA1_0_IRQHandler(void) {
    ADC14->CTL0 |= ADC14_CTL0_SC;
}
```

3.5 ADC

The ADC (Analog-to-Digital Convert) is used in our lab to convert an analog signal from a joystick to digital signal that can be used by the timer to create a PWM. The analog signal from the joystick is compared to a reference value (in our case the VSS) and outputs a value from 0 (ground level compared to VSS) to 2^7 (16383 when equal to 3V3).

The ADC is first enabled (ADC14_CTL0_ON) and the SMCLK with no prescaler is selected as its source. Next up, the "sample-and-hold pulse mode" as well as the memory space in which the converted value will be sent are selected. In our case the memory 13 of the ADC is chosen since it can be bind with the pin 4.0 of the board. We then enable the IRQ in order to process the resulting value at the end of the conversion. At that point the ADC is all setup and ready to go.

```
void setupADC(void) {
    ADC14->CTL0 |= ADC14_CTL0_ON;

    // setup control
    ADC14->CTL0 |= ADC14_CTL0_SHP;
    ADC14->CTL0 |= ADC14_CTL0_SSEL_4;
    ADC14->CTL0 |= ADC14_CTL0_CONSEQ_0;

    // setup memory control
    ADC14->MCTL[0] = ADC14_MCTLN_INCH_13;

    //enable interrupt on ADC
    NVIC_EnableIRQ(ADC14_IRQn);
    ADC14->IER0 = ADC14_IER0_IE0;

    ADC14->CTL0 |= ADC14_CTL0_ENC;
}
```

The interrupt for the ADC is addressed in the next section.

3.6 Conversion ADC - PWM

The signal feed to the servo is a PWM signal of period 20 milliseconds with a high signal lasting between 1 and 2 milliseconds and a low signal for the rest of the period. We can see on the figure below what the signal looks like.

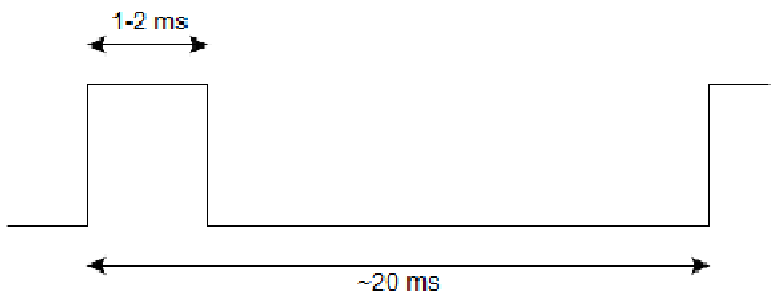


Figure 5: PWM signal

In our first tests the signal feed into the servo was not as precise as we wanted. The timings were a bit to short (around -10%). To have a precise signal we used a linear function to re-scale the output value of the ADC. The code used to update the value to the first timer's CCR1 is shown below.

```c
void ADC14_IRQHandler(void) {
    // set the period for the servo from 1 to 2ms
    // ADC min value is 0x0000
    // ADC max value is 0x2FFF (=16383)

    // read value in ADC
    int result = ADC14->MEM[0];
    // scale to have the interval between 1 and 2 ms
    int value = (int)((count_to_milli(1) * result) / ADC_MAX_VALUE) +
   count_to_milli(1);
    // rescale to be between 1 and 2 ms with more precision
    float time = VALUE_MIN + (value / (VALUE_MAX - VALUE_MIN));
    // write value in Timer A0 CCR1
    TIMER_A0->CCR[1] = (int)time;

}
```

4 Conclusion

Our system meets the project's requirement. The full range of the servo is used (90°), and the precision and accuracy meet the need of any application. This lab is therefore a success and a lot was learnt along the way.