Embedded Systems
Lab 3 - Introduction to PMW and ADC modules

You are asked to prepare the first part before the lab. Lab duration: 45min

A laptop with a working installation of MPLABX IDE and your toolbox are required.

The time available to complete the lab being quite limited, your are asked to have prepared the section 2 before attending it. To complete this preparation in a decent amount of time, you are encouraged to share it evenly between the different members of your group. For practical reasons, we suggest you to keep the same groups for all the labs.

1 Introduction

In this lab, you will learn how to use the pulse width modulation (PWM) and analog digital converter (ADC) of your microcontroller. It will also be an opportunity to give you a good methodology for writing a program which has to interact with several new modules.

2 Preparation

For this lab, you will reuse the circuit from the previous lab with one slight modification which is to plug the wiper (sliding contact) of a potentiometer on RD1 and to move the LED which was at this location on RC2 as shown on the schematic. Please make this modification before attending the lab.

The aim of this lab is to guide you trough the configuration and the use of two important microcontroller modules presented during the exercise sessions which are the ADC and the PWM signal generator. Both of them are extremely useful to make the microcontroller interact with its environment. ADCs are can indeed be used to communicate information to the microcontroller, while PMW are signals widely used to drive various electrical loads.

In practice, the final goal of this lab will be to dim a LED with a PWM signal and to adjust the PWM duty cycle depending on an analog voltage given by a potentiometer. The common way to proceed for this kind of application is to read the value at the output of the potentiometer at regular time intervals and to update the duty cycle right after that.

When facing several new modules, it is illusory to try to make them work altogether from the beginning. This indeed leads to a bunch of mistakes which are often difficult to find just because everything is new. To avoid that, it is often recommended to proceed by incremental steps, which implies to test each module individually with simple usecases. This offers the possibility to fully understand the behaviour and all subtleties of each module before performing more complicated tasks. We will apply this idea in this lab. Before implementing the code for the complete application, you will be asked to test the PWM and the ADC modules separately.

For this lab, we provide a code sample whose pseudocode is given in section 4.3. For the questions which follow, we will assume that the microcontroller is a PIC16F1789 clocked at $4MHz$.

2.1 ADC Module

1. What is the purpose of an ADC?
2. Assuming we set Vref+ at $5V$ and Vref- at $2V$, what is the value of the analog voltage if an 8-bit ADC outputs 128?
3. An ADC conversion is performed in two main steps. Which are they? What is performed at each of them?
4. Which Special Function Registers (SFRs) must be modified to configure the ADC? How should their different configuration bits be set if we want to perform a 10-bit conversion on RD1 (channel 21 of the ADC) with VDD and VSS as positive and negative reference respectively and with sign and magnitude format?
5. Which event does trigger an ADC interrupt?
6. What is the operation performed by the pseudocode shown in section 4.1?
7. By using the skeleton as a basis, implement the pseudocode shown in section 4.1 in assembly code for the PIC16F1789.

2.2 PWM Module

1. What are the two main parameters of a PWM signal? What do they represent concretely?
2. How is timer 2 used by the PWM module?
3. Explain why the maximum duty cycle resolution decreases when the PWM frequency increases.
4. Which Special Function Registers (SFRs) must be configured to generate a $15kHz$ PWM signal on RC2? How should their different configuration bits be set?
5. What is the pseudocode shown in section 4.2 performing?
6. By using the skeleton as a basis, implement the pseudocode shown in section 4.2 in assembly code for the PIC16F1789.

3 Code Test

During the lab session, your main task will be to test your code on your circuit to make sure that everything runs as expected. Before attempting to check it, you first need to make sure that your electronic circuit is functional. For this, you first need to make sure that your circuit is correct. Then, power it with the correct input voltage. Once this is done, you can simply reupload the code that was given for lab 1 into your microcontroller. If everything runs smoothly and if your circuit is correct, you should have a LED that blinks with a period of $2096ms$.

Once you are sure that your circuit is correct, you can begin to debug both test codes. As for the previous lab, you can use the oscilloscope to analyse your signals and to help you for the debug phase. For the ADC test code, you should obtain a LED which turns on or of depending on the position of the potentiometer. More precisely, it should turn on when the voltage rises above $2.5V$. The PWM test code, should dim the LED with a sawtooth signal (turning it completely on then progressively reducing the intensity by reducing the duty cycle).

If you don't observe this behaviour, it means that your modifications probably contain a mistake, or does not follow the scheme given by the pseudocode. Use the LEDs as debugging tools by turning them on and off at some points of your code. This will give you an idea of what your code is performing at some given time instants. When doing this, the LEDs usually blink too fast to be analysed by the eye. You should therefore rely on the oscilloscope to analyse your signals.

Once both test codes are debugged and tested, you can aggregate them in one code as shown by the pseudocode at section 4.4. Once done, you should be able to dim the LED by turning the potentiometer. When this is the case, call the teaching assistant to show your result.

4 Pseudocode

4.1 ADC Configuration test

```
void interrupt_routine()
{
   if(timer interrupt flag == 1):
   {
      Turn ADC on;
      Wait for the acquisition time; // ~ 10μs
      Start conversion;
      Turn ADC off;
   }
   if(ADC interrupt flag ==1)
   {
      clear ADC interrupt flag;
      if(ADRESH > 127)
      {
         RD0 = 1;
      }
      else
      {
         RD0 = 0;
      }
   }
   retfie; // assembly instruction to return from interrupt routine
}

void main()
{
   Configure RD port;
   Configure RC port;
   Configure oscillator;
   Configure ADC converter; // for exact parameters, see related question
   Configure Timer 1;
   Configure Timer 1 to trigger an interrupt after 100ms;
   Enable required interrupts;
   RD0 = 0;
   while(1)
   {
      nop;
   }
}
```

4.2 PWM configuration test

```
void interrupt_routine()
{
   if(timer interrupt flag == 1):
   {
      CCPR2L -= 16;
   }
   retfie; // assembly instruction to return from interrupt routine
}
void main()
{
   Configure RD port;
   Configure RC port;
   Configure oscillator;
   Configure Timer 2 for use with PWM module;
   Setup PWM;
   Configure Timer 1;
   Configure Timer 1 to trigger an interrupt after 100ms;
   Enable required interrupts;
   while(1)
   {
      nop;
   }
}
```

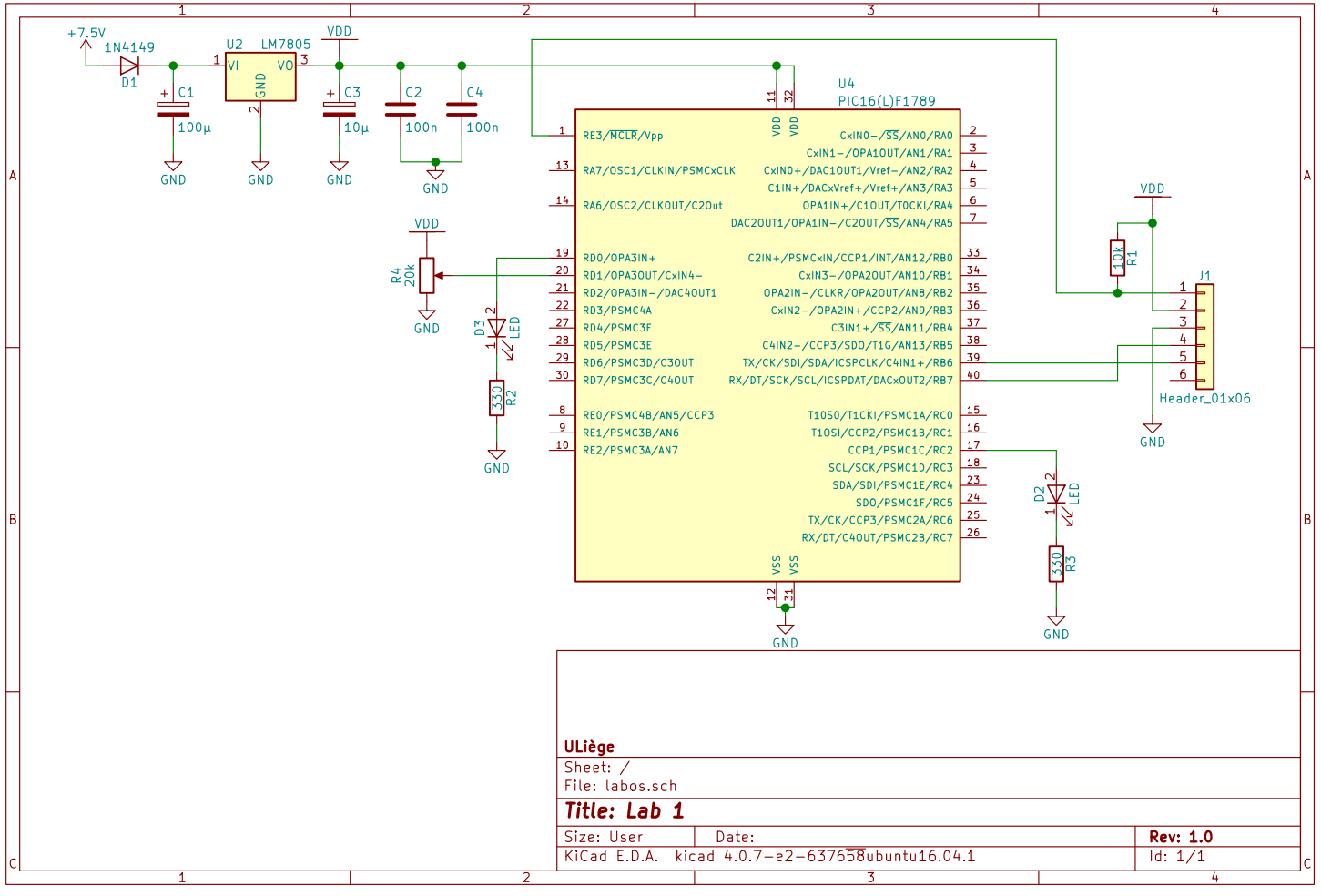
4.3 Provided Skeleton

```
void interrupt_routine()
{
   if(timer interrupt flag == 1):
   {
      Clear timer 1 interrupt flag;
      Reset timer 1 to 53035;
   }
   retfie; // assembly instruction to return from interrupt routine
}
void main()
{
   Configure RD port;
   Configure RC port;
   Configure oscillator;
   Configure Timer 1;
   Configure Timer 1 to trigger an interrupt after 100ms;
   Enable Timer 1 interrupt;
   while(1)
   {
      nop;
   }
}
```

4.4 Full Application

```
void interrupt_routine()
{
   if(timer interrupt flag == 1):
   {
      Clear timer 1 interrupt flag;
      Reset timer 1 to 53035;
      Turn ADC on;
      Wait for the acquisition time; // ~ 10μs
      Start conversion;
   }
   if(ADC interrupt flag ==1)
   {
      Clear ADC interrupt flag;
      CCPR2L = ADRESH;
      Turn ADC off;
   }
   retfie; // assembly instruction to return from interrupt routine
}

void main()
{
   Configure RD port;
   Configure RC port;
   Configure oscillator;
   Configure ADC converter; // for exact parameters, see related question
   Configure Timer 2 for use with PWM module;
   Setup PWM;
   Configure Timer 1;
   Configure Timer 1 to trigger an interrupt after 100ms;
   Enable required interrupts;
   while(1)
   {
      nop;
   }
}
```



ULiège

Sheet: /

File: labos.sch

Title: Lab 1

Size: User	Date:	**Rev: 1.0**
KiCad E.D.A. kicad 4.0.7-e2-637658ubuntu16.04.1		Id: 1/1