Embedded Systems
Lab 2 - Introduction to interrupts

A laptop with a working installation of MPLABX IDE is required.

1 Introduction

In this lab, you will learn how to make a good use of your microcontroller resources by enabling interrupts. For this you will be asked to modify the code given to the previous lab to make it more efficient while performing the same task.

Lab sessions are mandatory. Since we cannot organize presential lab sessions due to the current pandemic, we'll ask you to do them remotly and send a report containing your answers to the questions we ask in this document. Please, keep your answers short and to the point. There is no need for unecessary filling.

Note 1: For practical reasons, we ask you to keep the same groups for all the labs.
Note 2: You will be asked to send your report before the deadline stated on the exercise sessions website.

2 Preparation

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 this lab, you will have to reuse the circuit from the previous lab with one slight modification which is to add a LED on RD1 as shown on the schematic.

The aim of this lab is to show you how to use the different features of your microcontroller to take some load off of the ALU in order to gain a non negligible amount of precious computing resources. Microcontrollers, although being powerful in their category, have indeed a limited computing power compared to other computing platforms. It is therefore necessary to pay a special attention when writing code for embedded applications in order to squeeze as much power as possible out of your microcontroller.

The interrupt system is one of the most important tools that enables this. It indeed enables you to perform some simple repetitive and/or time consuming tasks in background, leaving time for other more complex tasks to be performed. Additionally, interrupts offer the possibility to have an extremely high reactivity to some events and to precisely manage timings. The latest being almost impossible without them.

From this point of view, the code given for the previous lab was highly inefficient. As shown in the pseudo-code at section 4, the ALU has to actively track the time elapsed in order to trigger events (here making a LED blink). This leaves little to no time for other tasks. Time tracking can however be performed in background simply by making the timer trigger interrupts at precise time intervals as shown in the second pseudocode.

The questions asked hereunder refer to the sections 3, 8, 13, 23 and 30 of the PIC16F1789 datasheet. You are not asked to read them from top to bottom, but rather understand the

structure and the type of information given in each of them in order to be able to pick the information you are looking for efficiently.

1. To which address of the program memory does an interrupt request make the program counter jump?
2. When entering the interrupt routine, how can you determine the interrupt source which triggered the interrupt when you have several interrupt sources enabled?
3. Which Special Function Registers (SFRs) must be modified to enable interrupts for the timer 1? How should their different configuration bits be set?
4. Which timer event does trigger a timer interrupt?
5. By knowing that the clock frequency is $4MHz$ and that the timer 1 is configured to use the instruction clock and a 1:2 prescaler, how should you configure TMR1H and TMR1L to trigger an interrupt after $0.05s$?

3 Coding exercise

With the preparation, you have all the tools to begin to use interrupts.

1. Modify the assembly code given for the previous lab to make it work as described in the pseudocode of section 4.2. Give your code in your report.
2. How many instructions have to be executed between the moment at which you switch RD1 to 1 and the one when it is turned back to 0?
3. How long will RD1 remain turned on?
4. Start the MPLABX simulator.

   By analyzing the pseudocode, we can see that the LEDs plugged on RD0 and RD1 should blink with a period exactly equal to $500ms$ and $50ms$ respectively. Furthermore, the LED plugged on RD1 should remain turned on only during the interrupt routine and therefore only during the time that you computed at the last question of the preparation.

   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. Try to find what's wrong and correct your code. Don't hesistate to use other tools provided by the simulator to help yourseleves.

   Once this is done, validate your answer to the previous question using two breakpoints (one on the line RD1=1 and the second on RD1=0) and the stopwatch of the simulator. Run your code from breakpoint to breakpoint until you enter your interrupt routine for the 4th time. At that moment, make a printscreen of the stopwatch pannel and include it in your report.

4 Pseudocode

4.1 Blinking LED by active waiting

```
#define value_counter 3

uint8 counter

void main()
{
   Configure RD port;
   Configure oscillator;
   Configure Timer 1;
   RD0 = 0;
   counter = value_counter;
   while(1)
   {
      Wait for ~65 ms;
      Reset timer registers;
      if(! --counter)
      }
         RD0 = ! RD0;
         counter = value_counter;
      }
   }
}
```

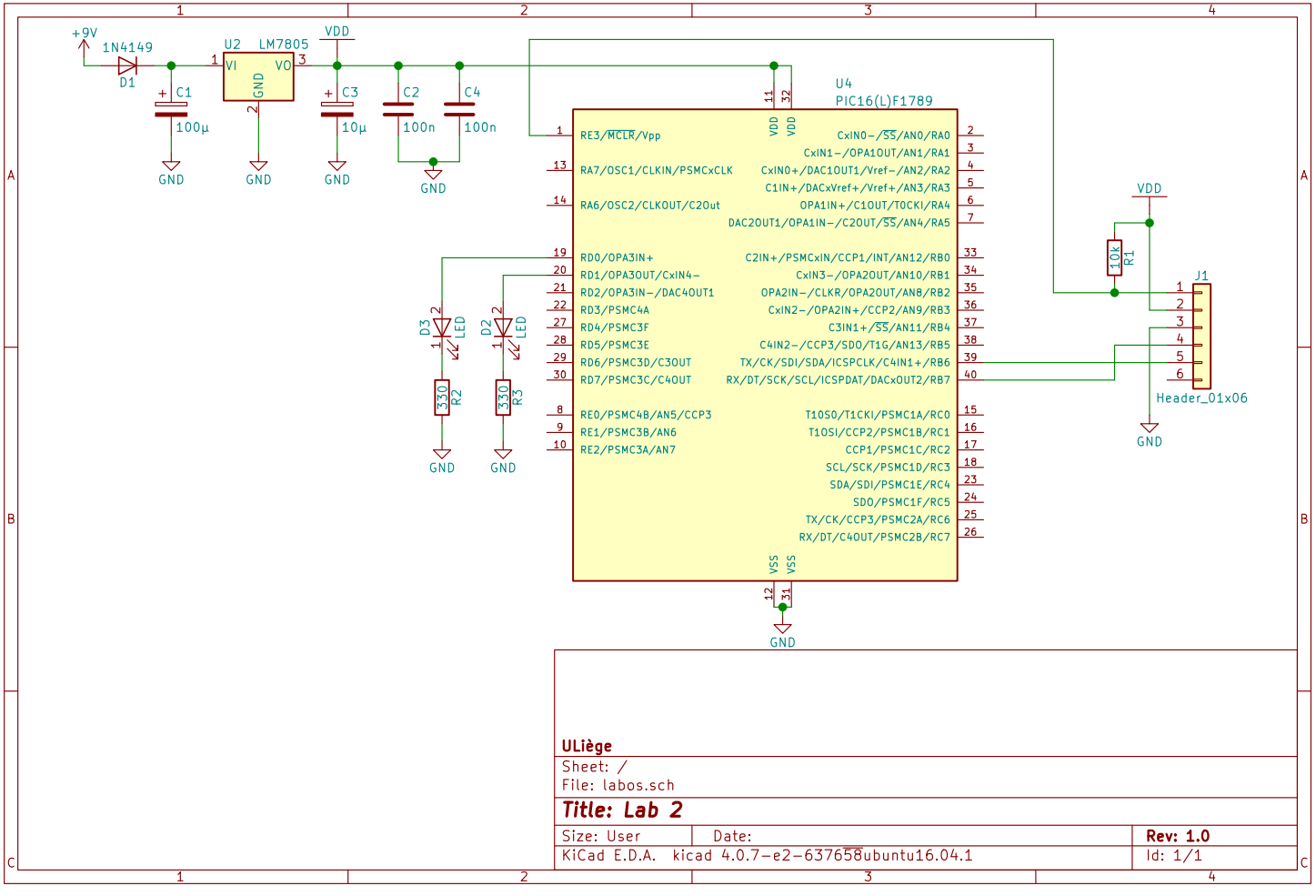
4.2 Blinking LED with timer interrupt

```
#define value_counter 5

uint8 counter

void interrupt_routine()
{
   if(timer interrupt flag == 1):
   {
      RD1 = 1
      Reset timer registers for 50ms;
      if(! --counter)
      {
         RD0 = ! RD0;
         counter = value_counter;
      }
      clear interrupt flag;
      RD1 = 0;
   }
   retfie; // assembly instruction to return from interrupt routine
}

void main()
{
   Configure RD port;
   Configure oscillator;
   Configure Timer 1;
   Configure Timer 1 to trigger an interrupt after 50ms;
   RD0 = 0;
   counter = value_counter;
   while(1)
   {
      nop
   }
}
```



ULiège

Sheet: /

File: labos.sch

Title: Lab 2

Size: User | Date: | **Rev: 1.0**

KiCad E.D.A. kicad 4.0.7-e2-637658ubuntu16.04.1 | Id: 1/1