

Lab 3

IMPORTANT NOTICE:

- Many pins of the AVR board are labelled wrongly.
- Many pins do not work.
- For each task, only use the pins given in the instructions.
- RDX4 and RDx3 are actually INT0 and INT1, respectively.
- The clock frequency of ATmega2560 on the board is 16MHz.

1. Objectives

In this lab, you will learn

- downloading a program into the AVR board and running it,
- interrupt-driven assembly programming, and
- assembly programming with ports and timer.

2. Preparations

Before coming in to the Laboratory, you should

- Read through the specifications of this lab in detail, trying to understand what you will be doing.
- Read through the sample code for controlling LEDs and the sample code for implementing external interrupts given in the subsequent descriptions.
- Read through the lecture notes on interrupts and the descriptions of Timer/Counter 0 in ATmega2560 Data Sheet to understand how to write an interrupt handler for Timer/Counter 0 Overflow Interrupt and external interrupts.
- Read through the lecture notes on ports.
- Download the Arduino IDE from <https://www.arduino.cc/en/Main/Software> and install it on your own computer. Make sure the USB driver is installed when you install the IDE.
- Download the zip file programmer.zip by clicking on “Downloader” under this lab, and extract the files in it and store them in your folder for this course on your own computer.

If your computer uses 32-bit Windows, you need to change the paths of avrdude.exe and avrdude.conf in the command of download.bat. For example:

```
"C:\Program Files\Arduino\hardware\tools\avr\bin\avrdude.exe" -C "C:\Program Files\Arduino\hardware\tools\avr\etc\avrdude.conf" -c wiring -p m2560 -P %1 -b 115200 -U flash:w:%2:i -D
```

3. Tasks

There are four compulsory tasks and no optional tasks in this lab. **Each task is worth 5 marks.** It may take several hours to read the relevant material before you can start. So you need to finish reading all the relevant material and work out most of the code at home.

3.1 AVR Development Board

This section introduces the AVR Microcontroller Development Board. In the subsequent labs you will use the development board to run your program instead of only simulating it using AVR Studio.

3.1.1 Board Handling Precautions

Due to the limited number of boards and two courses COMP2121 and COMP9032 using the boards, each group will be allocated one AVR development board at the beginning of the Week 7 lab. You are required to keep the board properly to avoid damaging it.

A few precautions are necessary when you use the AVR development board:

- Electrostatic discharge can destroy electronic equipment without giving any sign of doing so! Please always discharge yourself before handling the AVR microcontroller board, and observe any handling precautions as advised by the demonstrator. You can do this painlessly by touching a grounded conductor using a coin, a key or another metallic object instead of your finger.
- Short circuits may damage certain devices. Please remove any metal watch straps, buckles, rings and so on before handling the board.
- Always turn the power off before connecting or disconnecting any I/O subsystems.

3.1.2 Create New File

Create a new project and name it led. Its assembler code is shown as follows.

```
; led.asm
; Author:
; Date:
.include "m2560def.inc"
.def temp =r16
.equ PATTERN1 = 0x5B
.equ PATTERN2 = 0xAA
.cseg
.org 0x0
ser temp
out PORTC, temp          ; Write ones to all the LEDs
out DDRC, temp           ; PORTC is all outputs
out PORTF, temp          ; Enable pull-up resistors on PORTF
clr temp
out DDRF, temp           ; PORTF is all inputs
switch0:
sbic PINF, 0             ; Skip the next instruction if PB0 is pushed
rjmp switch1             ; If not pushed, check the other switch
ldi temp, PATTERN1       ; Store PATTERN1 to the LEDs if the switch was pushed
out PORTC, temp
switch1:
sbic PINF, 1             ; Skip the next instruction if PB1 is pushed
rjmp switch0             ; If not pushed, check the other switch
ldi temp, PATTERN2       ; Store PATTERN2 to the LEDs if the switch was pushed
out PORTC, temp
```

rjmp switch0 ; Now check PB0 again

Assemble the program using AVR Studio. You will find the executable file named led.hex in the Debug folder which is in the folder of the led project.

3.1.3 Download Program

Read the board introduction that describes how to download an executable to the board.

To run the program, do the following:

1. Use the provided patch cables to connect pins PC0-PC7 to the LED0-LED7, and PB0-PB1 (**on the second row of pins immediately above the motor**) to the PF0-PF1. Make sure you have the right order.
2. Plug the USB cable into the board and the PC
3. Download the executable file led.hex to the board.
4. After download led.hex, your program will be executed immediately.
5. Now push the two push buttons (green ones), and observe how LEDs change.
6. Push the RESET button (red one), and observe the behaviour of the board. The RESET button, when pushed, generates a RESET interrupt.

The two push buttons on the board are two switches. A switch has two outputs, 0 (low) or 1 (high). When a push button is not pushed, its output is 1. When a push button is pushed, its output is 0. The program led.asm continuously checks if any push button is pushed and sets the 8 LEDs to a particular pattern accordingly. Each AVR pin is strong enough to drive an LED. When a pin is configured as output and connected to an LED, writing 1 (0) to the pin will turn on (off) the LED.

3.2 Task A: Controlling LEDs with Push Buttons (5 marks)

Write an AVR assembly program that controls 8 LEDs using two push buttons. Your program must meet the following requirements:

- a. Use the bottom 4 LEDs LED0-LED3 and Port C to drive these 4 LEDs, where PC_i drives LED_i (i=0, 1, 2, 3). The pins of Port C are labelled with PC7-PC0.
- b. The 4 LEDs collectively denote a 4-bit number. When a LED is on, its bit is 1. Otherwise, its bit is 0. Initially, all the LEDs are on, indicating 0b1111.
- c. When PB0 is pushed, the 4-bit number will decrease by one. If the current value of the 4-bit number is 0b0000, pushing PB0 will produce 0b1111.
- d. When PB1 is pushed, the 4-bit number will increase by one. If the current value of the 4-bit number is 0b1111, pushing PB1 will produce 0b0000.

3.3 Task B: Controlling LEDs with Push Buttons and External Interrupts (5marks)

Modify your program for Task A to meet the following additional requirements:

- a. Your program is not allowed to check the output of each push button using a port. Instead, your program must use two external interrupts, INT0 (External Interrupt 0) and INT1 (External Interrupt 1). When PB0 is pushed, it generates an INT0 interrupt. The interrupt handler for INT0 decreases the 4-bit number by one by

setting the 4 LEDs to the equivalent state. When PB1 is pushed, it generates an INT1 interrupt. The interrupt handler for INT1 increases the 4-bit number by one by setting the 4 LEDs to the equivalent state.

- b. A falling edge generates an interrupt.

The following sample code may help you complete Task B.

```
.include "m2560def.inc"
.def temp = r16
.equ HIGH_LEDS = 0b11110000
.equ LOW_LEDS = 0b00001111
.cseg
.org 0x0
jmp RESET
.org INT0addr ; INT0addr is the address of EXT_INT0
jmp EXT_INT0
.org INT1addr ; INT1addr is the address of EXT_INT1
jmp EXT_INT1
```

```
RESET:
ldi temp, low(RAMEND)
out SPL, temp
ldi temp, high(RAMEND)
out SPH, temp
ser temp
out DDRC, temp
clr temp
out PORTC, temp
out DDRD, temp
out PORTD, temp
ldi temp, (2 << ISC10) | (2 << ISC00)
sts EICRA, temp
in temp, EIMSK
ori temp, (1 << INT0) | (1 << INT1)
out EIMSK, temp
sei
jmp main
```

```
EXT_INT0:
push temp
in temp, SREG
push temp
ldi temp, HIGH_LEDS
out PORTC, temp
pop temp
out SREG, temp
```

```
pop temp
reti
```

```
EXT_INT1:
push temp
in temp, SREG
push temp
ldi temp, LOW_LEDS
out PORTC, temp
pop temp
out SREG, temp
pop temp
reti
```

```
main:          ; main - does nothing but increment a counter
clr temp
loop:
inc temp
rjmp loop
```

To understand how the above code works, you need to read through the external interrupts, last part of the lecture notes on interrupts. Details about external interrupts are given in pages 112-117, ATmega2560 Data Sheet.

Notice that PD0 and PD1 are equivalent to INT0 and INT1, respectively.

3.3 Task C: Implementing A Clock Using Software Delay (5 marks)

Write a program to implement a software clock meeting the following requirements:

- Do not use any interrupt.
- The clock has only two components, minutes and seconds. For example, 2:30 denotes that 2 minutes and 30 seconds has passed since the program started to run. Use the 6 LEDs LED5-LED0 as a binary counter for seconds, and the 2 LEDs LED7-LED6 as a binary counter for minutes ignoring overflow. For example, if LED5:LED4:LED3:LED2:LED1:LED0 are off:off:on:on:on:on, it indicates 001111, which means 15 seconds. A similar counter mechanism is applied to the minute component.
- The clock should change whenever one second has passed.
- Use Port C to control LEDs. The pins of Port C are labelled with PC7-PC0.

Hint: You need to write a loop with an execution time of one second. Note that the frequency of the ATmega2560 on the board is 16 MHz. Delay using a loop is called software delay. The problem with the software delay is that it consumes CPU time without doing significant stuff.

3.4 Task D: Implementing A Clock Using Timer0 Overflow Interrupt (5 marks)

Modify your code for Task C to meet the following additional requirement:

- Do not use software delay. Instead, your code must use Timer0 Overflow Interrupt. The interrupt handler for Timer0 Overflow Interrupt calculates the time and displays the time using the 8 LEDS.

You may refer to the second counter example in the lecture notes on interrupts.

Deadline: Week 8