



**Teaching online electronics, microcontrollers and
programming in Higher Education**

**Output 2: Online Course for Microcontrollers:
syllabus, open educational resources**

Practice leaflet: Module_2-3 external – RB port
change interrupts

Lead Partner: International Hellenic University (IHU)

Authors: Theodosios Sapounidis [IHU], Aristotelis Kazakopoulos [IHU], Aggelos Giakoumis [IHU], Sokratis Tselegkaridis [IHU]

Declaration

This report has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

© Copyright 2021 - 2023 the [ENGINE](#) Consortium

Warsaw University of Technology (Poland)

International Hellenic University (IHU) (Greece)

European Lab for Educational Technology- EDUMOTIVA (Greece)

University of Padova (Italy)

University of Applied Sciences in Tarnow (Poland)

All rights reserved.



This document is licensed to the public under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

Executive summary	4
Chapter 1: Overview	5
Chapter 2: Activities	8
2.1 Activity 1. Flash a LED.....	8
2.2 Activity 2. Create a moving dot	10
2.3 Activity 3. RB4~RB7 on change interrupt	13
2.4 Activity 4. Simple alarm system.....	16
Chapter 3: Recapitulation.....	20
References	21
Appendix. Figures with high resolution.....	22

Executive summary

In this Module we will use PIC18F4550 external interrupts.

Chapter 1: Overview

Table 1. Overview

Title / short summary	3. External interrupts – RB port change interrupts
Expected learning outcomes	<ul style="list-style-type: none">• The student will be able to handle external interrupts:<ul style="list-style-type: none">◦ INT0 (RB0)◦ INT1 (RB1)◦ INT2 (RB2)• The student will be able to handle RB port change interrupts (on pins RB4~RB7)• The student will be able to load and animate a microcontroller program in the Proteus Design Suite
Keywords	External interrupts, RB interrupts
Duration	<p>The duration of the module_2-3 is 3 hours</p> <ul style="list-style-type: none">• Presentation of the module_2-3 by the teacher, 30 minutes• 1st activity, flash a LED with INT1, 40 minutes• 2nd activity, create a moving dot with INT2, 35 minutes• 3rd activity, RB4~RB7 on change interrupt, 35 minutes• 4th activity, a simple alarm system, 40 minutes

Involved	<p>The teacher:</p> <p>Presents the slides associated with the module _2-3 and answers question</p> <p>The students:</p> <p>Draw circuits in Proteus Schematic, write programs in C language, load programs to a microcontroller and run the simulation using the Proteus Design Suite</p>
Assignment	<p>At the end of the Module_2-3 will be given:</p> <ul style="list-style-type: none"> • Open Project
Educational tools and equipment	<ul style="list-style-type: none"> • Material: PC • Software: CCS C compiler, Proteus Design Suite
Prerequisites / pre-existing knowledge	<ul style="list-style-type: none"> • The student must be familiarized with the Proteus Design Suite (link1) • The student must be completed Module_2-1 and Module_2-2
Educational content	<ul style="list-style-type: none"> • CCS C Compiler manual (C Compiler Reference Manual) • MICROCHIP, PIC18F2455/2550/4455/4550 Data Sheet • Module_2-3 slides • Module_2-3 Evaluation leaflet • Module_2-3 Open project leaflet • Module_2-3 Programs, Schematic Proteus (Compressed folder)

Tips

Tip. Requirement / operation of the compiler about RB change interrupts.

The value of PORTB (or one of the pins RB4, RB5, RB6, RB7) must be read to clear the interrupt flag (IF).

If the IF is not cleared when exiting the interrupt service routine (ISR), the IF is raised and the program re-enters the ISR.

Chapter 2: Activities

2.1 Activity 1. Flash a LED

The purpose of this activity is to flash a LED twice, through the interrupt service routine of RB1 (INT1).

Table 2. Activity 1

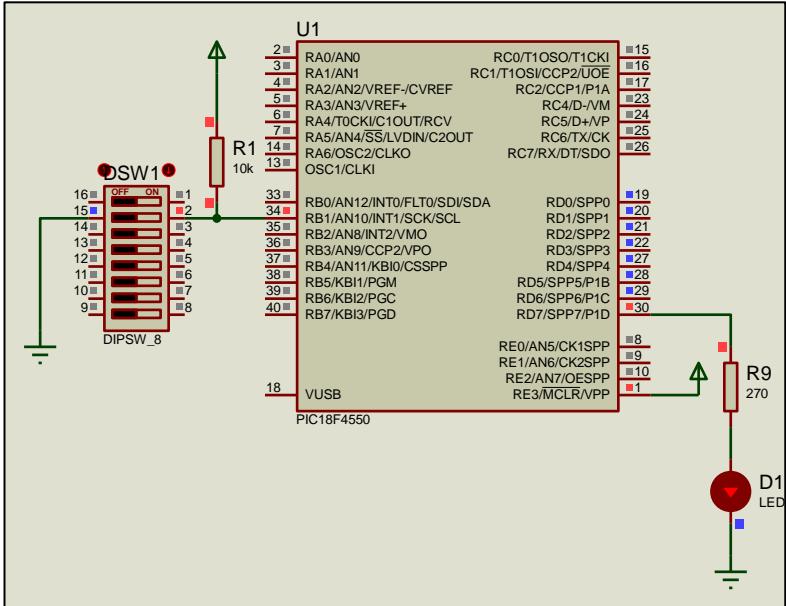
Activity 1st (40 minutes)	<p>Step 1. The circuit is drawn in the Proteus Design Suite.</p> <p>Step 2. The program in C language is written.</p> <p>Step 3. The program is compiled with the use of CCS C compiler to the microcontroller machine code.</p> <p>Step 4. The machine code is loaded to the microcontroller.</p> <p>Step 5. The animation is activated.</p> <p>Step 6. Modifications and discussion.</p>
Step 1 (10 minutes)	<p>Draw the circuit of the picture in the Proteus Design Suite.</p>  <p>The circuit diagram shows a PIC18F4550 microcontroller (U1) with pin 18 labeled VUSB. Pin 1 is connected to a 10k resistor (R1) which goes to ground. Pin 1 is also connected to the top terminal of a dip switch (DSW1). The bottom terminal of DSW1 is connected to pin 16 of the microcontroller. Pin 16 is labeled OFF/ON. Pin 1 is also connected to the non-inverting input of an inverter (represented by a triangle symbol). The output of the inverter is connected to pin 14 of the microcontroller. Pin 14 is labeled DIPSW_8. Pin 14 is also connected to the non-inverting input of another inverter. The output of this second inverter is connected to pin 33 of the microcontroller. Pin 33 is labeled RBO/AN12/INT0/FLT0/SDI/SDA. Pin 33 is also connected to the non-inverting input of a third inverter. The output of this third inverter is connected to pin 34 of the microcontroller. Pin 34 is labeled RB1/AN10/INT1/SCK/SCL. Pin 34 is also connected to the non-inverting input of a fourth inverter. The output of this fourth inverter is connected to pin 35 of the microcontroller. Pin 35 is labeled RB2/AN8/INT2/VMO. Pin 35 is also connected to the non-inverting input of a fifth inverter. The output of this fifth inverter is connected to pin 36 of the microcontroller. Pin 36 is labeled RB3/AN9/CCP2/PO. Pin 36 is also connected to the non-inverting input of a sixth inverter. The output of this sixth inverter is connected to pin 37 of the microcontroller. Pin 37 is labeled RB4/AN11/KB10/CSPP. Pin 37 is also connected to the non-inverting input of a seventh inverter. The output of this seventh inverter is connected to pin 38 of the microcontroller. Pin 38 is labeled RB5/KB11/PGM. Pin 38 is also connected to the non-inverting input of a eighth inverter. The output of this eighth inverter is connected to pin 39 of the microcontroller. Pin 39 is labeled RB6/KB12/PGC. Pin 39 is also connected to the non-inverting input of a ninth inverter. The output of this ninth inverter is connected to pin 40 of the microcontroller. Pin 40 is labeled RB7/KB13/PGD. Pin 40 is also connected to the non-inverting input of a tenth inverter. The output of this tenth inverter is connected to pin 15 of the microcontroller. Pin 15 is labeled RA0/AN0. Pin 15 is also connected to the non-inverting input of a eleventh inverter. The output of this eleventh inverter is connected to pin 16 of the microcontroller. Pin 16 is labeled RA1/AN1. Pin 16 is also connected to the non-inverting input of a twelfth inverter. The output of this twelfth inverter is connected to pin 17 of the microcontroller. Pin 17 is labeled RA2/AN2/VREF-/CVREF. Pin 17 is also connected to the non-inverting input of a thirteenth inverter. The output of this thirteenth inverter is connected to pin 18 of the microcontroller. Pin 18 is labeled RE0/AN5/CK1SPP. Pin 18 is also connected to the non-inverting input of a fourteenth inverter. The output of this fourteenth inverter is connected to pin 19 of the microcontroller. Pin 19 is labeled RD0/SPP0. Pin 19 is also connected to the non-inverting input of a fifteenth inverter. The output of this fifteenth inverter is connected to pin 20 of the microcontroller. Pin 20 is labeled RD1/SPP1. Pin 20 is also connected to the non-inverting input of a sixteenth inverter. The output of this sixteenth inverter is connected to pin 21 of the microcontroller. Pin 21 is labeled RD2/SPP2. Pin 21 is also connected to the non-inverting input of a seventeenth inverter. The output of this seventeenth inverter is connected to pin 22 of the microcontroller. Pin 22 is labeled RD3/SPP3. Pin 22 is also connected to the non-inverting input of a eighteenth inverter. The output of this eighteenth inverter is connected to pin 23 of the microcontroller. Pin 23 is labeled RC4/D-/VM. Pin 23 is also connected to the non-inverting input of a nineteenth inverter. The output of this nineteenth inverter is connected to pin 24 of the microcontroller. Pin 24 is labeled RC5/D+/VP. Pin 24 is also connected to the non-inverting input of a twentieth inverter. The output of this twentieth inverter is connected to pin 25 of the microcontroller. Pin 25 is labeled RC6/TX/CK. Pin 25 is also connected to the non-inverting input of a twenty-first inverter. The output of this twenty-first inverter is connected to pin 26 of the microcontroller. Pin 26 is labeled RC7/RX/DT/SDO. Pin 26 is also connected to the non-inverting input of a twenty-second inverter. The output of this twenty-second inverter is connected to pin 1 of the microcontroller. Pin 1 is labeled RE1/AN6/CK2SPP. Pin 1 is also connected to the non-inverting input of a twenty-third inverter. The output of this twenty-third inverter is connected to pin 2 of the microcontroller. Pin 2 is labeled RE2/AN7/OESPP. Pin 2 is also connected to the non-inverting input of a twenty-fourth inverter. The output of this twenty-fourth inverter is connected to pin 3 of the microcontroller. Pin 3 is labeled RE3/MCLR/VPP. Pin 3 is also connected to the non-inverting input of a twenty-fifth inverter. The output of this twenty-fifth inverter is connected to pin 4 of the microcontroller. Pin 4 is labeled D1 LED. Pin 4 is connected to a 270 ohm resistor (R9) which goes to ground. The other end of R9 is connected to the anode of the LED (D1). The cathode of D1 is connected to ground.</p>

Figure 1. INT1 - flash a LED

**Step 2
(10 minutes)**

Write in CCS C Compiler the program in C language

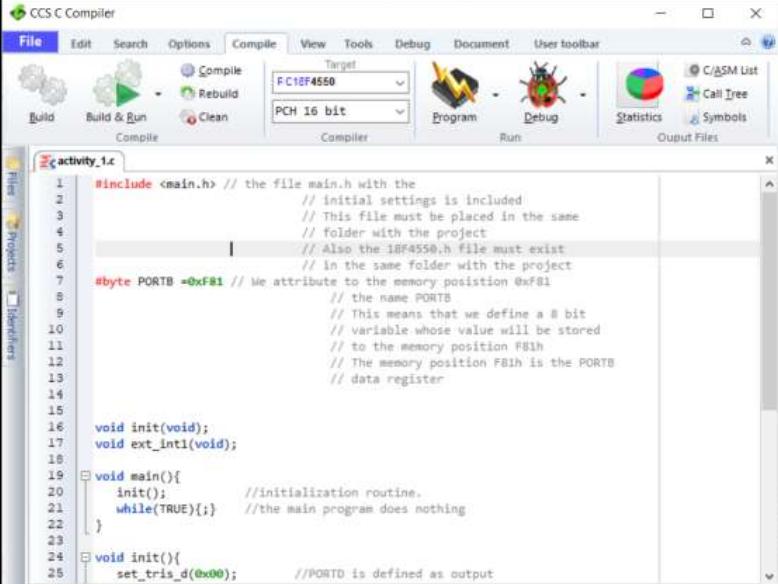
```
#include <main.h> // the file main.h with the
                  // initial settings
is included
                  // This file must be
placed in the same
                  // folder with the
project
                  // Also the 18F4550.h
file must exist
                  // in the same folder
with the project
#define PORTD =0xF83 // We attribute to the memory
position 0xF83
                  // the name PORTD
                  // This means
that we define a 8 bit
                  // variable whose
value will be stored
                  // to the memory
position F83h
                  // The memory
position F83h is the PORTD
                  // data register

void init(void);
void ext_int1(void);

void main(){
    init();           //initialization routine
    while(TRUE){;}   //the main program does
nothing
}

void init(){
    set_tris_d(0x00); //PORTD is defined as
output
    PORTD = 0b00000000; //The PORTD data
register is given the value 0
    ext_int_edge(1, L_TO_H); //Activation of the
interrupt from RB1
                                //during the
transition from 0 to 1 (raising edge)
    enable_interrupts(GLOBAL); //Enable global
interrupts
    enable_interrupts(INT_EXT1); //Enable external
interrupt by RB1
}

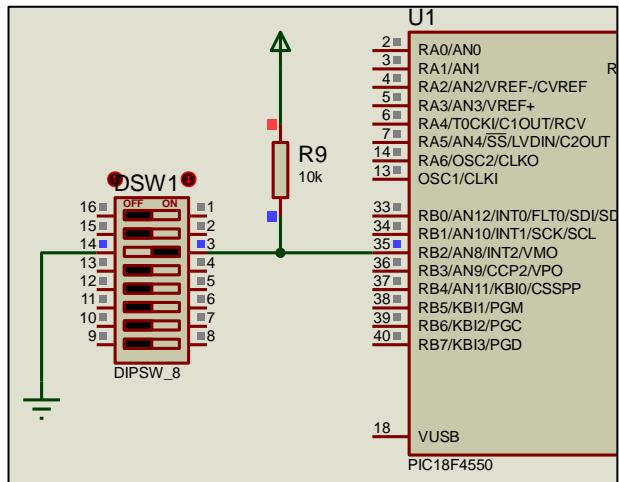
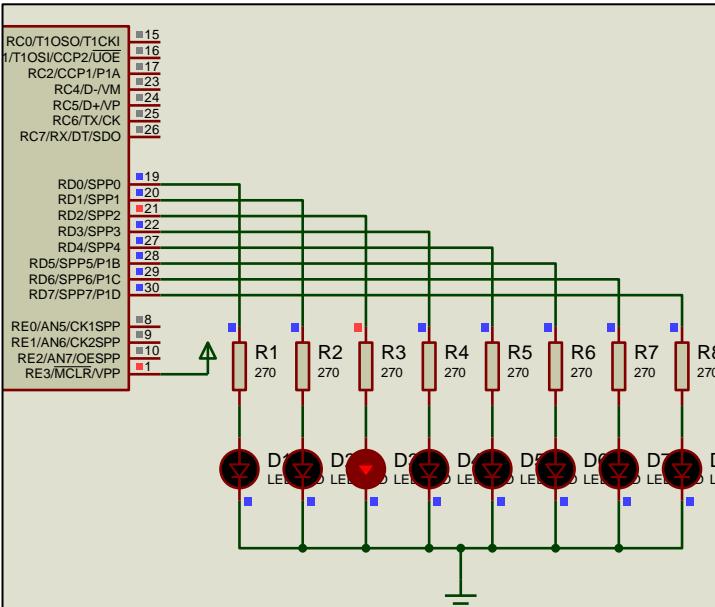
#INT_EXT1 HIGH //External interrupt by RB1
void ext_int1(){
    int i;
    for(i=1;i<3; i++){ //performed twice
        output_high(PIN_D7); //LED is on
        delay_ms(200);      //wait for 0.2s
        output_low(PIN_D7); //LED is off
        delay_ms(200);     //wait for 0.2s
    }
}
```

Step 3 (4 minutes)	<pre>}</pre> <p>Compile the program in C in order to create the program in the microcontroller machine code (hex file).</p> 
Step 4 (1 minutes)	Load to the microcontroller the hex file (program in machine code) that was created from the CCS C Compiler.
Step 5 (5 minute)	Run the simulation and check the correct operation of the circuit.
Step 6 (10 minutes)	<p>Suggested modifications and discussion:</p> <ul style="list-style-type: none"> • modify the code and circuit accordingly so that the interrupt occurs from RB0. Run and check the simulation

2.2 Activity 2. Create a moving dot

The purpose of this activity is to interrupt the main program. In the main program 8 LEDs flash. When an interrupt occurs from INT2, the LEDs create a moving dot.

Table 3. Activity 2

Activity 2 nd (35 minutes)	<p>Step 1. The circuit is drawn in the Proteus Design Suite.</p> <p>Step 2. The program in C language is written.</p> <p>Step 3. The program is compiled with the use of CCS C compiler to the microcontroller machine code. The machine code is loaded to the flash memory of the microcontroller.</p> <p>Step 4. The animation is activated.</p>
Step 1 (15 minutes)	<p>Draw the circuit of the picture at the Proteus Design Suite.</p>  <p>Figure 3(a). INT2 and LEDs</p>  <p>Figure 3(b). INT2 and LEDs</p>

**Step 2
(10 minutes)**

Write in CCS C Compiler the program in C language

```
#include <main.h> // the file main.h with the
// initial settings
is included
// This file must be
placed in the same
// folder with the
project
// Also the 18F4550.h
file must exist
// in the same folder
with the project
#define PORTD =0xF83 // We attribute to the memory
position 0xF83
// the name PORTD
// This means
that we define a 8 bit
// variable
whose value will be stored
// to the memory
position F83h
// The memory
position F83h is the PORTD
// data register

void init(void);
void ext_int2(void);

void main(){
    init();           //initialization routine
    while(TRUE){      //flash 8 LEDs
        PORTD=0b11111111;
        delay_ms(100);
        PORTD=0b00000000;
        delay_ms(100);
    }
}

//initialization routine
void init(){
    set_tris_d(0x00);           //PORTD is defined
as output
    PORTD = 0b00000000;         //The PORTD data
register is given the value 0

    ext_int_edge(2, H_TO_L);    //Activation of
the interrupt from RB2
                                //during the
transition from 1 to 0 (falling edge)

    enable_interrupts(GLOBAL);  //Enable global
interrupts
    enable_interrupts(INT_EXT2); //Enable external
interrupt by RB2
}

//external interrupt by RB2
#INT_EXT2
```

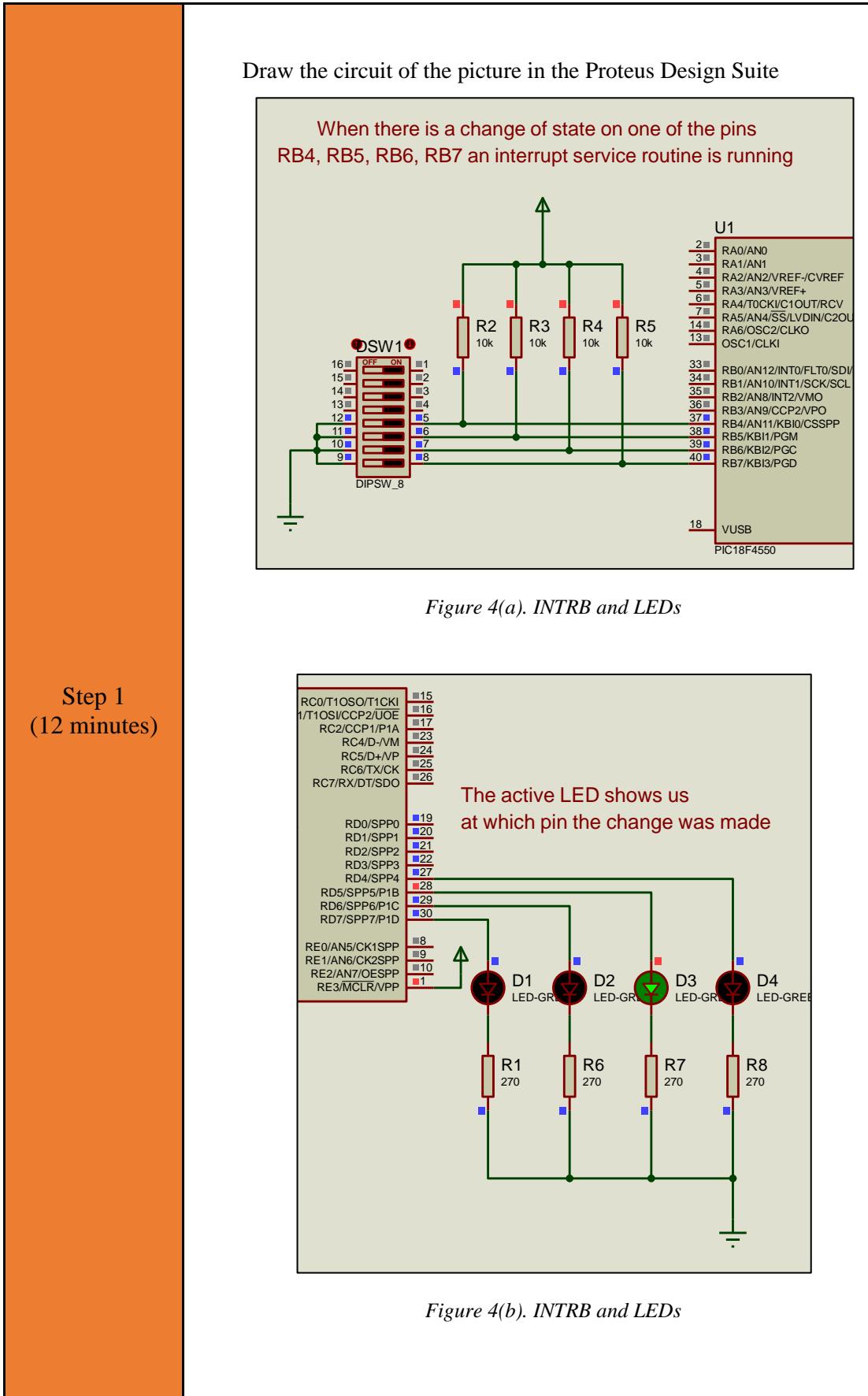
	<pre> void ext_int2() { //moving dot PORTD=0b00000000; delay_ms(200); PORTD=0b10000000; delay_ms(200); PORTD=0b01000000; delay_ms(200); PORTD=0b00100000; delay_ms(200); PORTD=0b00010000; delay_ms(200); PORTD=0b00001000; delay_ms(200); PORTD=0b00000100; delay_ms(200); PORTD=0b00000010; delay_ms(200); PORTD=0b00000001; delay_ms(200); PORTD=0b00000010; delay_ms(200); PORTD=0b000000100; delay_ms(200); PORTD=0b000001000; delay_ms(200); PORTD=0b000010000; delay_ms(200); PORTD=0b000100000; delay_ms(200); PORTD=0b010000000; delay_ms(200); PORTD=0b100000000; delay_ms(200); PORTD=0b000000000; delay_ms(200); } </pre>
Step 3 (5 minutes)	Use the CCS C Compiler to translate the programm from C language to the microcontroller machine code. Load to the microcontroller the hex file (machine code) that was created from the CCS Compiler.
Step 4 (5 minutes)	Run the simulation and check the correct operation of the circuit.

2.3 Activity 3. RB4~RB7 on change interrupt

The purpose of this activity is to handles interrupts by state changes in RB4, RB5, RB6, and RB7. When an interrupt occurs, the corresponding LED connected to the PORTD is activated.

Table 4. Activity 3

Activity 3 rd (35 minutes)	<p>Step 1. The circuit is drawn at the Proteus Design Suite.</p> <p>Step 2. The program in C language is written.</p> <p>Step 3. The program is compiled with the use of CCS C compiler to the microcontroller machine code (the hex.file is created). The program in machine code is loaded to the microcontroller.</p> <p>Step4. The animation is activated.</p>
--	--



**Step 2
(13 minutes)**

Write in CCS C Compiler the program in C language

```
#include <main.h> // the file main.h with the
                  // initial settings
is included
                  // This file must be
placed in the same
                  // folder with the
project
                  // Also the 18F4550.h
file must exist
                  // in the same folder
with the project
#define PORTD =0xF83 // We attribute to the memory
position 0xF83
                  // the name PORTD
                  // This means
that we define a 8 bit
                  // variable
whose value will be stored
                  // to the memory
position F83h
                  // The memory
position F83h is the PORTD
                  // data register

#define PORTB=0xF81 // We attribute to the memory
position 0xF81
                  // the name PORTD
                  // This means
that we define a 8 bit
                  // variable
whose value will be stored
                  // to the memory
position F81h
                  // The memory
position F81h is the PORTD
                  // data register

void rb(void) ;      //Interrupt service routine
statement (from RB4, RB5, RB6, RB7)
void init(void);
int8 lastPORTB; //Global variable to hold the last
value of PORTB

void main(){
    init();      //call the initialization routine
    while(TRUE){;} //the main program does nothing
}

//initialization routine
void init(){
    set_tris_d(0x00);          //PORTD is defined
as output
    PORTD = 0b00000000;        //The PORTD data
register is given the value 0
    lastPORTB=PORTB;
```

	<pre> enable_interrupts(GLOBAL); //Enable global interrupts enable_interrupts(INT_RB); //Enable change interrupt by RB4, RB5, RB6, RB7 } //PORTB change interrupt #INT_RB void rb (void){ int8 changes; //Define an 8bit variable changes = lastPORTB ^ PORTB; //The changed bit becomes 1 and appears in the corresponding position in the change variable lastPORTB=PORTB; //The new PORTB value is transferred to the lastPORTB variable PORTD=changes; //The changed bit is displayed in PORTD delay_ms (100); //delay to avoid bounces } </pre>
Step 3 (5 minutes)	Compile the program in order to create the hex.file (program in machine code). Load the program (hex.file) to the microcontroller.
Step 4 (5 minutes)	Run the simulation and check the correct operation of the circuit.

2.4 Activity 4. Simple alarm system

The purpose of this activity is to create a simple alarm system. The system sensors are simulated by 4 switches connected to RB4 ~ RB7. The alarm works as follows: a switch in RB0 arms or disarms the system. If the system is armed and one of the 4 switches changes state, then the microcontroller activates an LED (or a buzzer) for 6 seconds. The sensor / switch that gave the alarm is displayed in PORTD.

Table 5. Activity 4

Activity 4 rd (40 minutes)	<p>Step 1. The circuit is drawn at the Proteus Design Suite.</p> <p>Step 2. The program in C language is written.</p> <p>Step 3. The program is compiled with the use of CCS C compiler to the microcontroller machine code (the hex.file is created). The program in machine code is loaded to the microcontroller.</p>
--	---

	<p>Step4. The animation is activated.</p> <p>Draw the circuit of the picture in the Proteus Design Suite.</p> <div style="border: 1px solid black; padding: 10px;"> <p>The alarm is armed or disarmed from the switch in PB0</p> <p>When RB0 is 1 the alarm is armed When RB0 is 0 the alarm is off</p> <p>The switches on RB4, RB5, RB6 and RB7 correspond to the four sensor/switches</p> </div> <p style="text-align: center;"><i>Figure 5(a). A simple alarm system</i></p>
<p>Step 1 (15 minutes)</p>	<p>The LED that lights up corresponds to the switch on which the status was changed</p> <p style="text-align: center;"><i>Figure 5(b). A simple alarm system</i></p>

**Step 2
(15 minutes)**

Write in CCS C Compiler the program in C language

```
#include <main.h> // the file main.h with the
                  // initial settings is
                  // This file must be placed
                  // in the same
                  // folder with the project
                  // Also the 18F4550.h file
must exist
                  // in the same folder with
the project
#byte PORTD=0xF83 // We attribute to the memory position
0xF83
                  // the name PORTD
                  // This means that we
define a 8 bit
                  // variable whose
value will be stored
                  // to the memory
position F83h
                  // The memory position
F83h is the PORTD
                  // data register
#byte PORTB=0xF81 // We attribute to the memory position
0xF81
                  // the name PORTD
                  // This means that we
define a 8 bit
                  // variable whose
value will be stored
                  // to the memory
position F81h
                  // The memory position
F81h is the PORTD
                  // data register
#byte PORTC=0xF82 // We attribute to the memory position
0xF82
                  // the name PORTC
                  // This means that we
define a 8 bit
                  // variable whose
value will be stored
                  // to the memory
position F82h
                  // The memory position
F82h is the PORTC
                  // data register

//Declaration of functions, global variables
void init (void);           //initialization routine
void rb (void);              //interrupt service routine
statement (from RB4, RB5, RB6, RB7)

int8 lastPORTB;             //Global variable to hold the
last value of PORTB

void main(){
    init();                 //call the initialization routine
    while (TRUE) {}         //the main program does nothing
}

//interrupt service routine (change on RB4~RB7)
#INT_RB
void rb (void){
    int8 changes;           //Define an 8bit variable
```

	<pre> changes = lastPORTB ^ PORTB; //The changed bit becomes 1 and appears in the corresponding position in the change variable lastPORTB=PORTB; //The new PORTB value is transferred to the lastPORTB variable if(input(PIN_B0)==1){ output_high(PIN_C0); //alarm is activated PORTD=changes; //The changed bit of PORTB is displayed on PORTD a LED is on delay_ms(6000); //wait for 6 seconds output_low(PIN_C0); //alarm is de-activated PORTD=0x00; //LED is off } } //initialization routine void init (void){ set_tris_b(0xff); // PORTB is defined as input set_tris_d(0x00); // PORTD is defined as output set_tris_c(0x00); // PORTC is defined as output enable_interrupts(GLOBAL); //Enable global interrupts enable_interrupts(INT_RB); //Enable change interrupt_by RB4, RB5, RB6, RB7 PORTD=0x00; //The PORTD data register is given the value 0 PORTC=0x00; //The PORTC data register is given the value 0 lastPORTB=PORTB; //The new PORTB value is transferred to the lastPORTB variable } </pre>
Step 3 (5 minutes)	Compile the program in order to create the hex.file (program in machine code). Load the program (hex.file) to the microcontroller.
Step 4 (5 minutes)	Run the simulation and check the correct operation of the circuit.

Chapter 3: Recapitulation

- ☞ The schematic of the circuits was drawn with Proteus Design Suite
- ☞ External interrupts - RB port change interrupts were used to implement applications: flash a LED, create a moving dot, simple alarm system.
- ☞ The programs in C was written in CCS C Compiler.
- ☞ An interrupt service routine was used.
- ☞ The programs in C was compiled to the microcontroller machine code (hex file).
- ☞ The machine code was “loaded” to the microcontroller and the animation was activated.

References

CCS C Compiler Manual. Ccsinfo.com. (2021). Retrieved from https://www.ccsinfo.com/downloads/ccs_c_manual.pdf.

PIC18F2455/2550/4455/4550 Data Sheet. Ww1.microchip.com. (2006). Retrieved from <https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>.

Proteus Tutorial : Getting Started with Proteus PCB Design (Version 8.6). Youtube.com. (2017). Retrieved from <https://www.youtube.com/watch?v=GYAHwYUUs34>.

Simple LED Circuits. Electronics Hub. (2017). Retrieved from <https://www.electronicshub.org/simple-led-circuits/>.

Appendix. Figures with high resolution

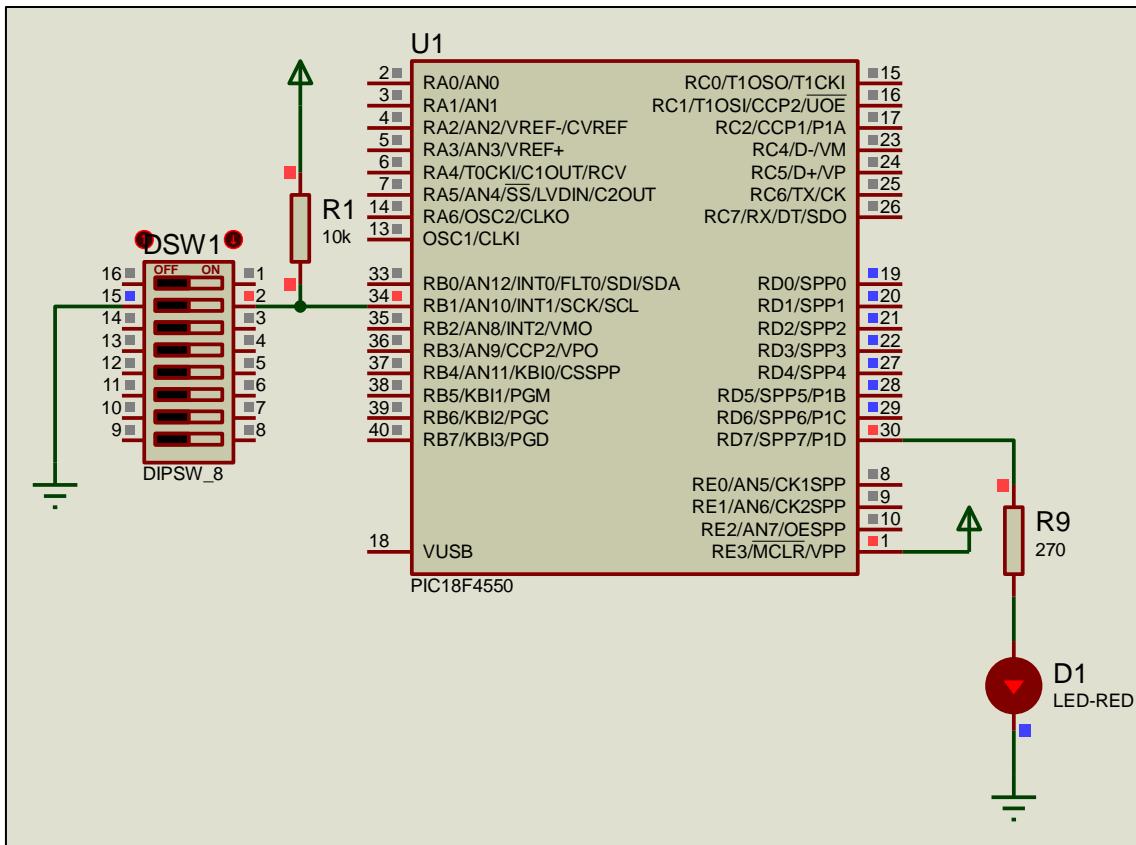


Figure 1. INT1 - flash a LED

The screenshot shows the CCS C Compiler software interface. The main window has a menu bar with File, Edit, Search, Options, Compile, View, Tools, Debug, Document, and User toolbar. Below the menu is a toolbar with icons for Build, Build & Run, Clean, Target (set to PIC18F4550), PCH 16 bit, Program, Debug, Statistics, Call Tree, and Symbols. On the left, there's a sidebar with Files, Projects, and Folders. The main code editor window displays the file activity_1.c with the following content:

```
1 #include <main.h> // the file main.h with the
2 // initial settings is included
3 // This file must be placed in the same
4 // folder with the project
5 // Also the 18F4550.h file must exist
6 // in the same folder with the project
7 #byte PORTB =0xF81 // We attribute to the memory position @xF81
8 // the name PORTB
9 // This means that we define a 8 bit
10 // variable whose value will be stored
11 // to the memory position F81h
12 // The memory position F81h is the PORTB
13 // data register
14
15
16 void init(void);
17 void ext_int1(void);
18
19 void main(){
20     init();           //initialization routine.
21     while(TRUE){}    //the main program does nothing
22 }
23
24 void init(){
25     set_tris_d(0x00); //PORTD is defined as output
```

Figure 2. CCS C Compiler, translation to machine code (hex file)

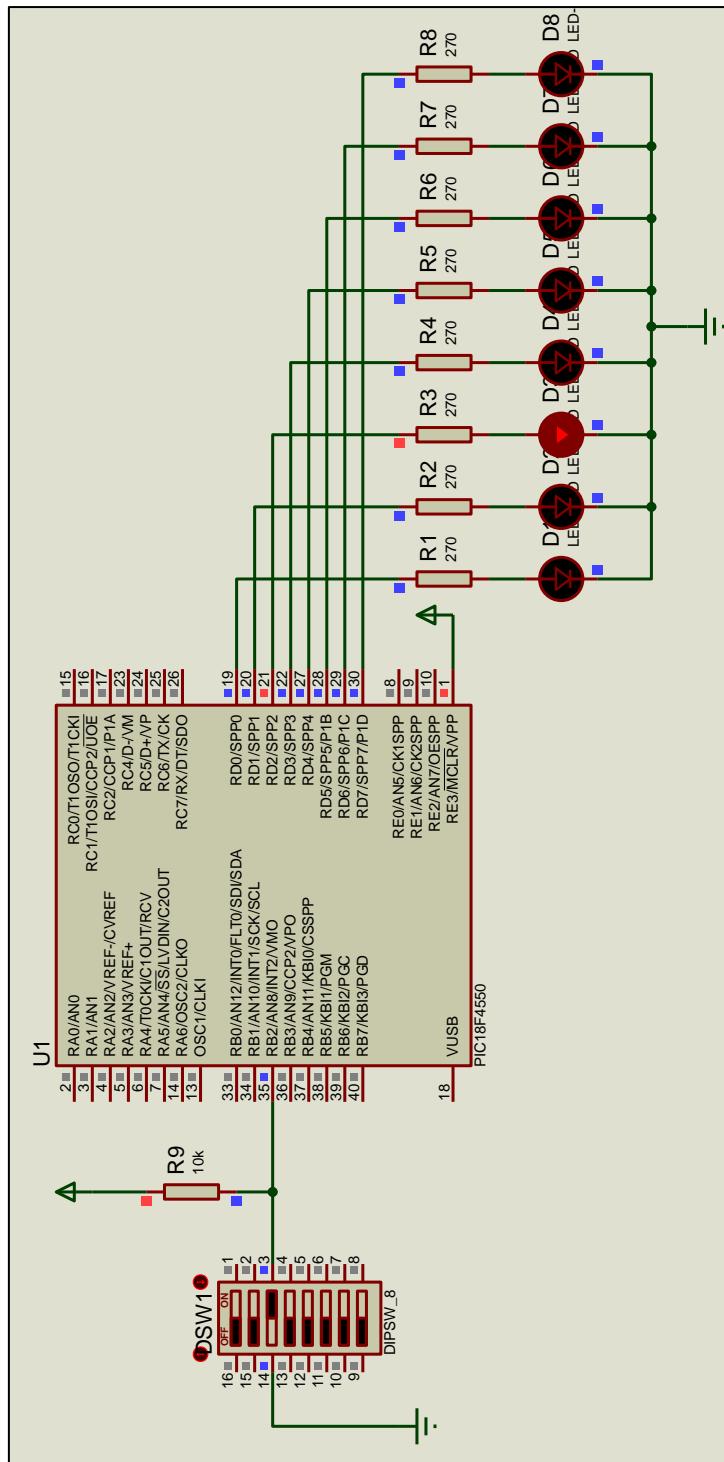


Figure 3. INT2 and LEDs

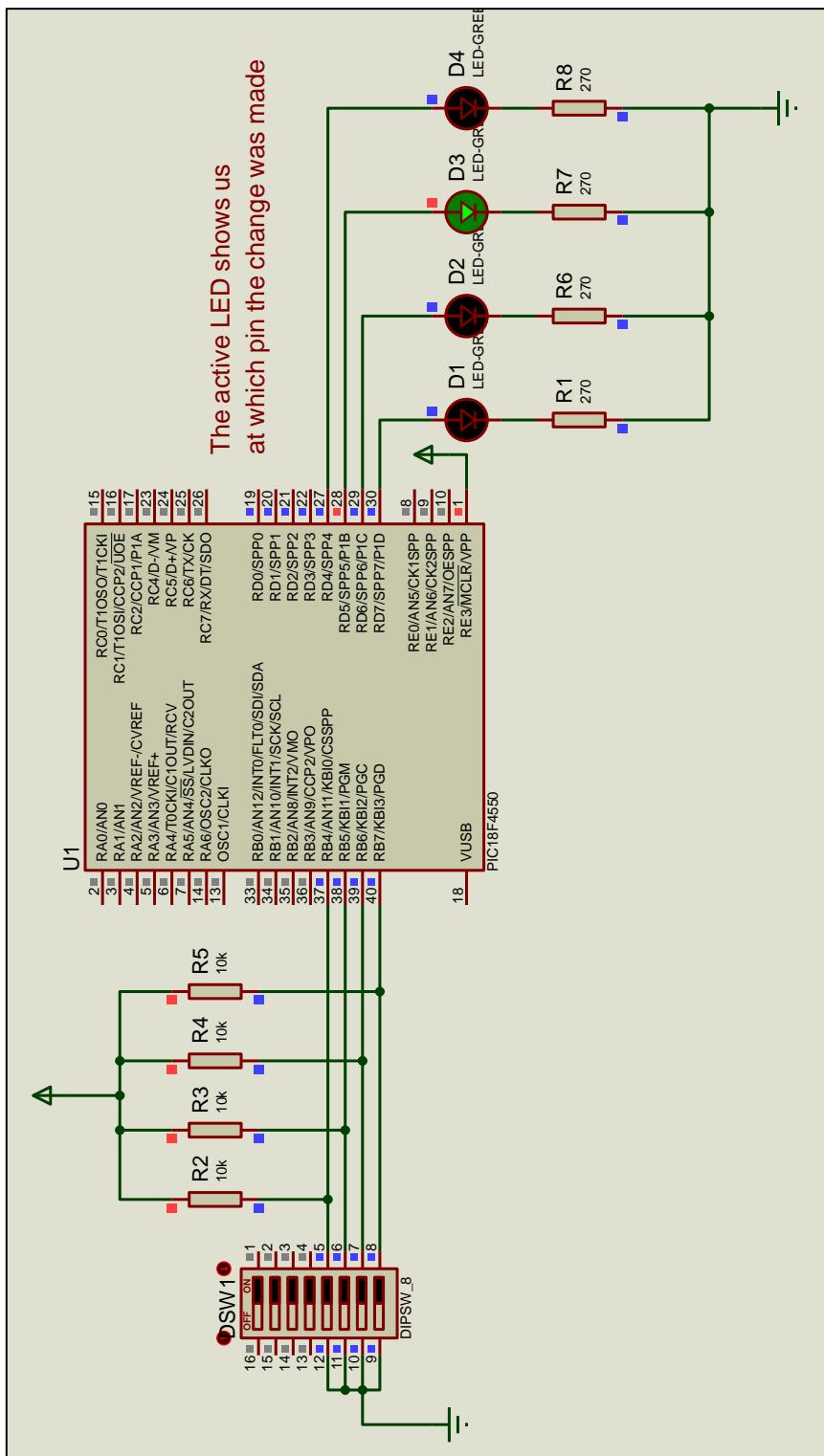


Figure 4. INTRB and LEDs

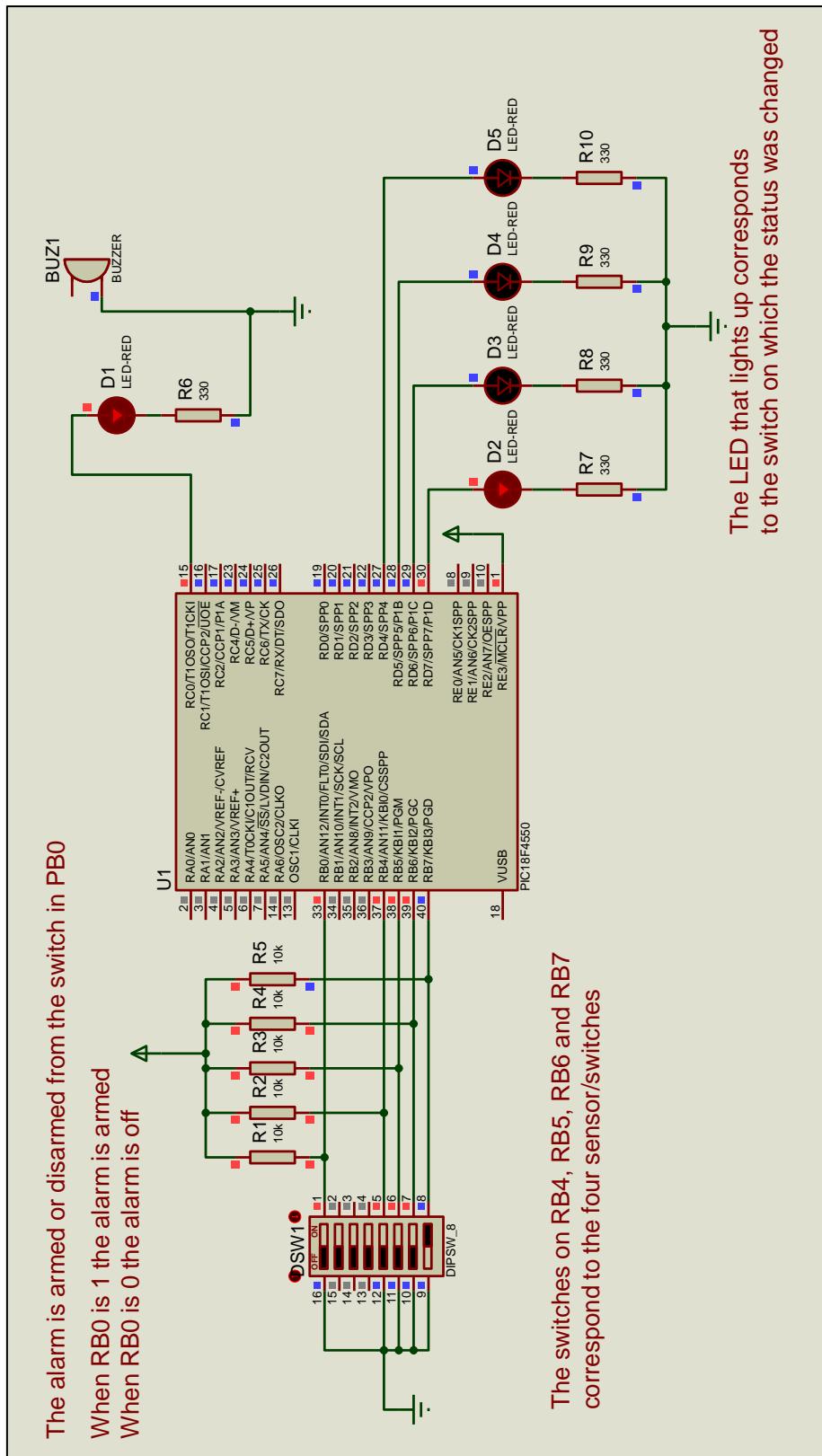


Figure 5. A simple alarm system

