

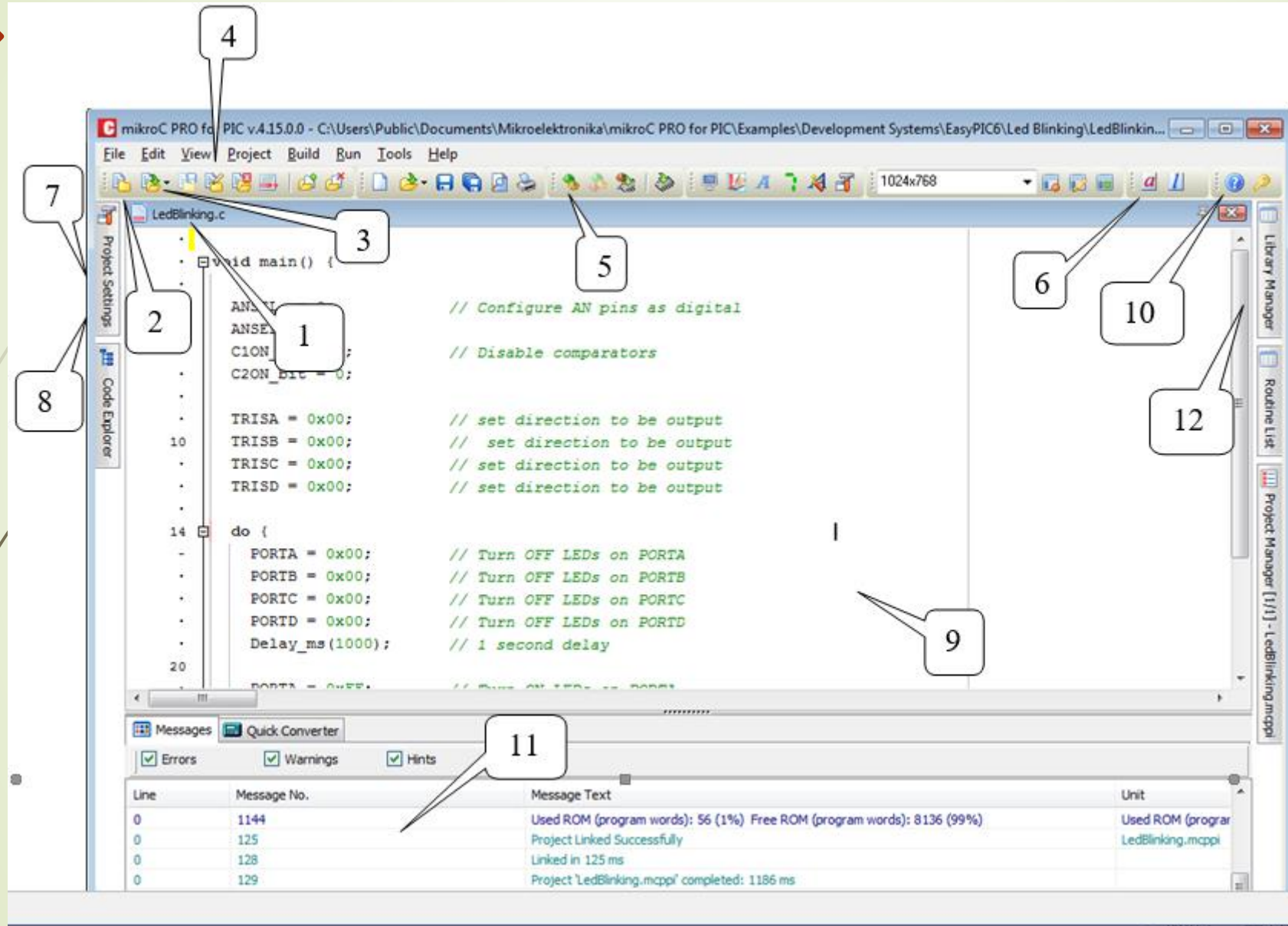
# **FIRST STEPS WITH THE PIC 16F84A TROUGH EXAMPLES IN MikroC**



# Introduction

Microcontrollers in general and the PICs in particular are components whose real implementation requires programming. The basic programming language of these components is the assembly language. Many difficulties related to the programming with this language (the mastering of the instruction set, the length of the source codes, the interpretation not obvious of the instructions...) led to the setting-up of the high-level languages, closer to the human language and easier to handle and understand. Among them, we enumerate MikroC and CCS (for a programming in of C language), PROTON (BASIC), MikroPascal (Pascal)... In this small tutorial, we will learn through examples how to program and simulate a microcontroller with MikroC and Proteus.

# Presentation of the MikroC interface



# Presentation of the MikroC interface

- 1 Name of the current file or program
- 2 **` New Project'**: allows to create a new project
- 3 **` Open Project'**: allows to open an already existing project
- 4 **` Project' Edict'**: allows modifying the basic parameters of the project (the PIC name, the name of the project, the frequency of the PIC actually programmed)
- 5 **` Build Project'**: allows to build the project and to generate the file of extension .HEX. It is the file, which is inserted in the PIC for a real implementation. It is this file which will be useful to us in simulation
- 6 **` View Assembly'**: allows to see the assembly version of the program written
- 7 Shows the name of the PIC we are programming. Ex 16F84A
- 8 Shows the frequency of the clock to be used during the realization
- 9 Represents the zone of edition of the source code
- 10 **` Help'**: allows to enter the help of the software MikroC
- 11 Presents the errors and the warnings relating to the current program. The posted messages give the nature of the error and its position in the source code or say if the program is well typed
- 12 The library manager allows to activate or deactivate some MiKroC's libraries

# Creation of a new project

**During the creation of a new project, a certain number of choices must be operated:**

- **The name of the project**
- **The path (it is preferable to create a file in advance where all the files of the project will be safeguarded)**
- **Name of the PIC to be programmed**
- **The choice of its frequency of clock**
- **The activation of certain additional parameters such as the Watchdog, the protection of code**
- **The type of oscillator (with quartz, RC circuit ...)**
- **And others.**

**When we click on the button ' New Project' (2), the following pages appear and all the parameters mentioned above can be located and adjusted.**

# Creation of a new project

New Project Wizard

Steps:

1. Project settings

2. Add files

3. Libraries

Project Settings:

Project Name: MyProject

Project folder: C:\Users\Public\Documents\Mikroelektronika\mikrot

Browse

Device name: P16F876A

Device clock: 8.000000 MHz

Open Edit Project window to set Configuration bits ☐

Enter project name, project folder, select device name and enter a device clock (for example: 80.000).

Checking 'Open Edit Project' option will open 'Edit Project' window after closing this wizard.  
This enables you to easily setup your device and project.

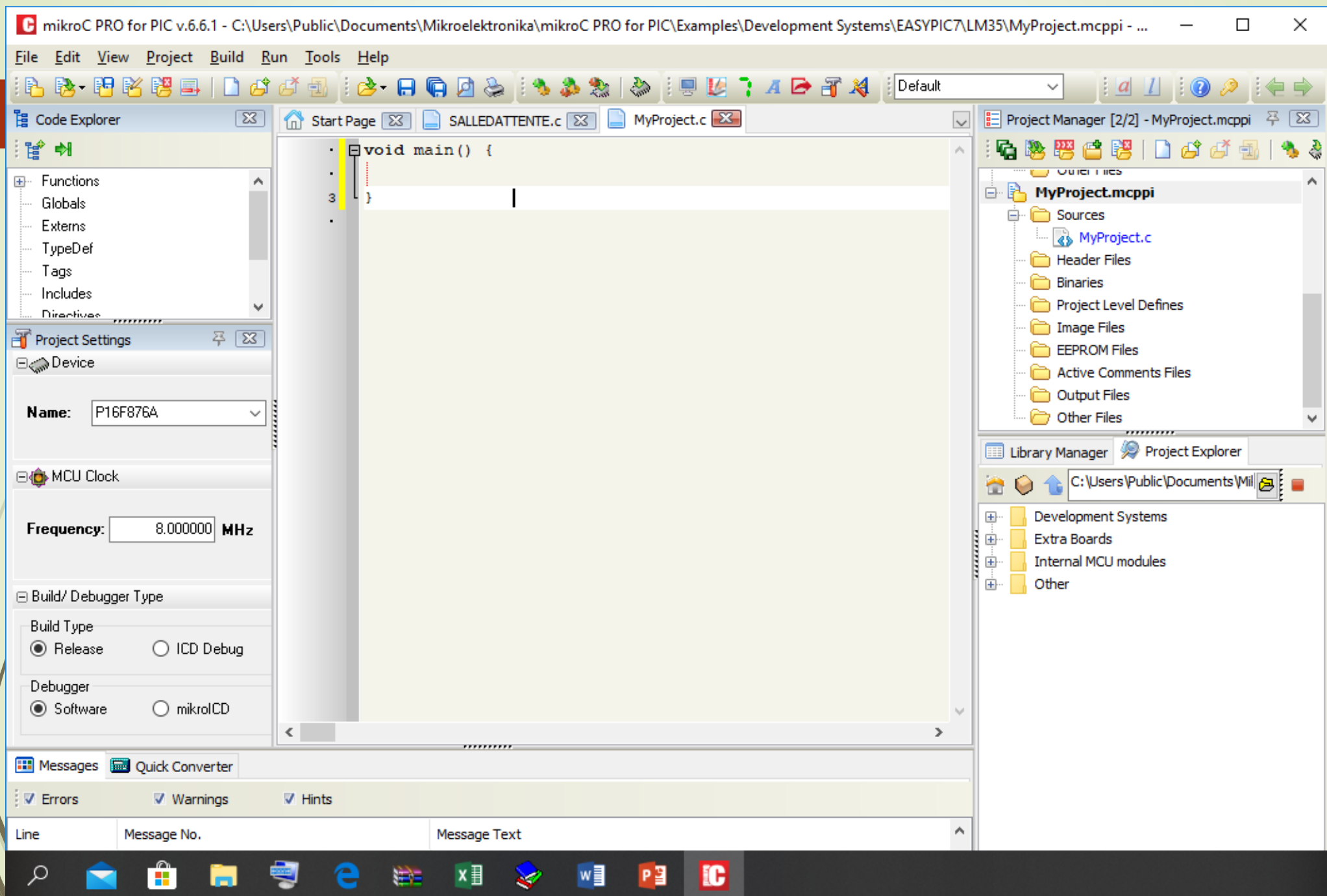
**Note: Project name and project folder must not be left empty.**

Back

Next

Cancel





# Structure of a MikroC program

```
/* Declaration of global variables*/
```

```
/* Declaration of functions if they exist*/
```

```
/* main Program*/
```

```
void main()
```

```
{
```

```
/* Initializations*/
```

```
/* Definition of an endless loop in which the program must run*/
```

```
do
```

```
{
```

```
/*Program to be carried out*/
```

```
} while (1); /*end of the endless loop*/
```

```
} /* end of program*/
```



# Let us explore the skeleton above presented

The comments on a line are given by `//` '. When it is for several lines, we use `/*` ' at the beginning and `*/` at the end of the comment. Like in C/C++ language

- Below *'Declaration of the global variables'* we will declare the variables to be used in all the program.

Example:

```
//Declaration of the global variables
```

```
unsigned short kp, cnt;    // integers
```

```
char txt[5 ];             //Table of six characters
```

```
float deci;               //a floating point Number
```

# Let us explore the skeleton above presented

- Below *'Declaration of functions if they exist'* we will declare functions. A function is a block of program that we can write and call in our main program as a single statement when the need be. We will see that option later
- Below *'Initializations'* we will introduce all that relates to the initialization of the PIC. For example to define which ports will be used and how their pins will be used (as input or as output) and the initial state of the port


```
TRISB = 0x00; // setting PORTB as output  
TRISA = 0x1F; // all PORTA is set as input
```

# Let us explore the skeleton above presented

In the endless loop *"do{ }while(1);"* we will put the program to be carried out by the PIC

Like in C/C++ , the endless loop can be realized using the following statements

```
do
{
    /* statements to be repeated*/
} while(1);
```

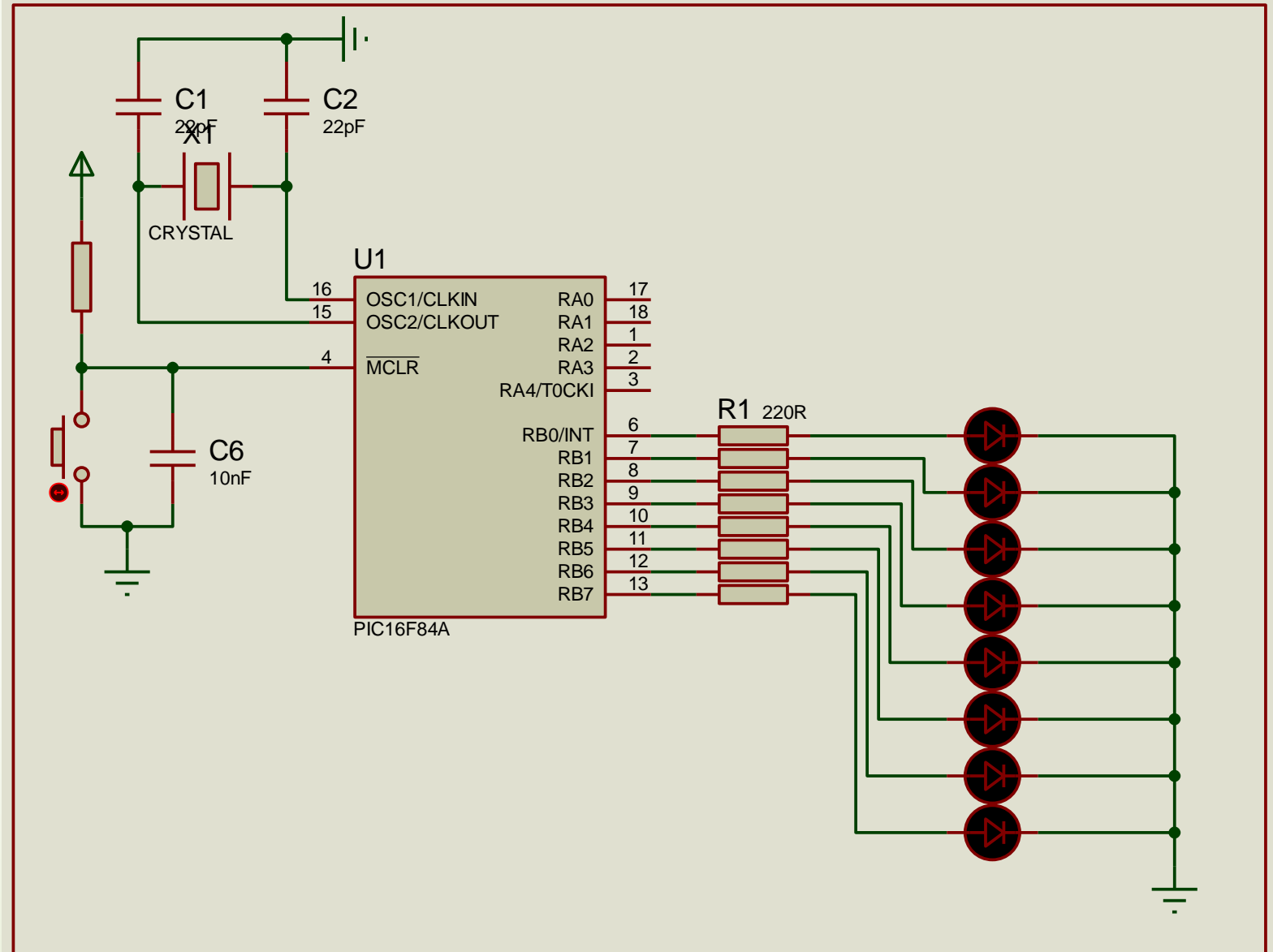


```
while(1)
{
    /* statements to be repeated*/
}
```

```
for(;;)
{
    /* statements to be repeated*/
}
```

# Our first project: To Blink all the LEDs connected to the PORTB of the microcontroller PIC16F84A

We start with the circuit



# The program

```
///////////////// Declaration of global variables  
/////////////////
```

```
///////////////// Declaration of functions //////////////////
```

```
///////////////// Main program //////////////////////////
```

```
void main()
```

```
{
```

```
    //Initializations //
```

```
    TRISB = 0x00; // setting PORTB as output
```

```
    TRISA = 0x1F; // all PORTA is set as input
```

```
    //Definition of the endless loop /// //
```

```
do
```

```
{
```

```
    //Program to carry out
```

```
    PORTB = 0b1111111; // PORTB high
```

```
    Delay_ms(500);      // 500 milliseconds delay
```

```
    PORTB = 0b00000000; // all PORTB off
```

```
    Delay_ms(500);      // 500 milliseconds delay
```

```
    } while(1); // endless loop
```

```
}          // end of the program
```

## Generation of the file of extension .HEX

For that, we will click on "Build Project" (5)

# Simulation with ISIS-Proteus

For the simulation to be effective, the program have to be loaded in our PIC. The procedure is as follows:

- Double click on the PIC
- Set the frequency, and browse to look at the .HEX file from the folder containing our project
- Click on the “PLAY” button to observe the behavior of the whole circuit in general and the PIC microcontroller in particular

**Edit Component**

Component Reference: U1 Hidden: ☐

Component Value: PIC16F84A Hidden: ☐

PCB Package: DIL18 ? Hide All

Program File: Timer&chenillard.HEX ? Hide All

Processor Clock Frequency: 4MHz Hide All

Program Configuration Word: 0x3FFB Hide All

Advanced Properties:

Randomize Program Memory? No Hide All

Other Properties:

☐ Exclude from Simulation ☐ Attach hierarchy module

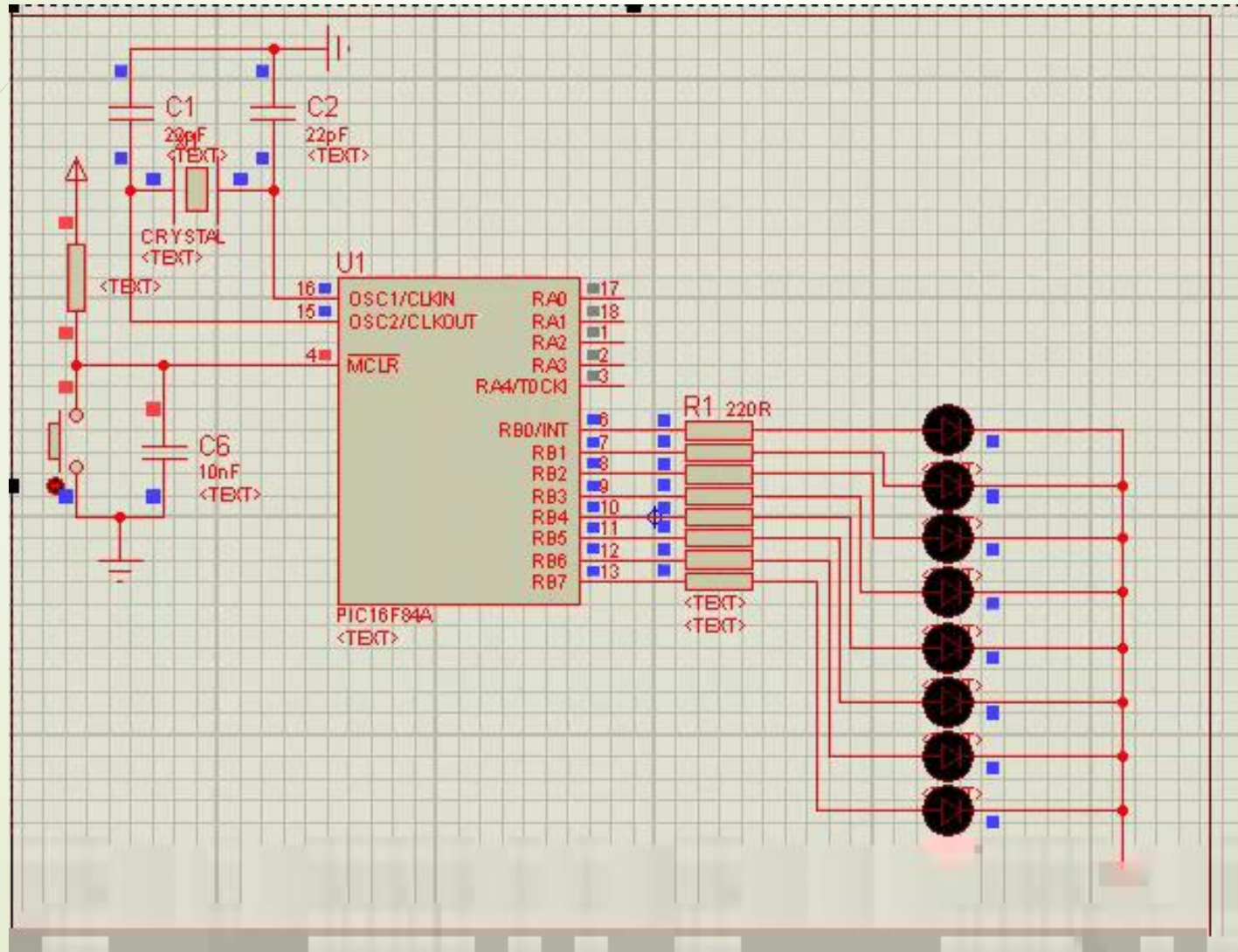
☐ Exclude from PCB Layout ☐ Hide common pins

☐ Edit all properties as text

Buttons: OK, Help, Data, Hidden Pins, Cancel



# Result of the simulation of Project1

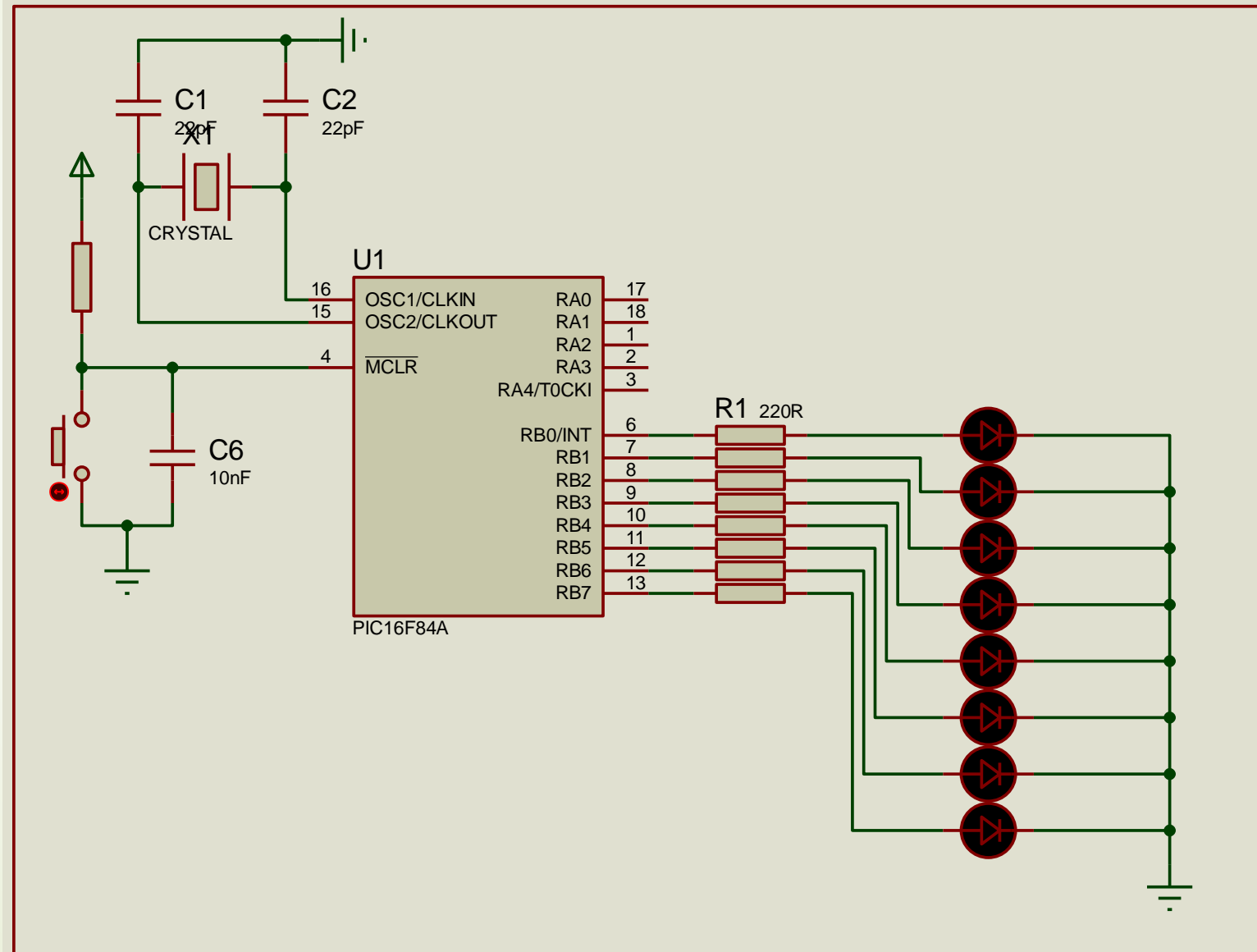


Play the video  
and observe

# Project 2: Gradually put ON and OFF the LEDs connected to the PORTB of the PIC16F84A

## - The diagram

We still consider the diagram of the previous project1.



# The program

```
////////// Declaration of global variables //////////

////////// Declaration of functions if they exist //////////

////////// Main Program //////////
void main()
{
    //Initializations //
    TRISB = 0x00; // setting all PORTB as output
    TRISA = 0x1F; // setting all PORTA as input
    PORTB = 0x00; // PORTB is initialized to 0

    //Definition of the endless loop in which the program will be running
    permanently /// //////////
    do
    {
        //Program to be executed

        // Sequence of lighting
        PORTB = 0b00000001; //
        Delay_ms(500); // 500 milliseconds break
        PORTB = 0b00000011; //
        Delay_ms(500); // 500 milliseconds break
        PORTB = 0b00000111; //
        Delay_ms(500); // 500 milliseconds break
        PORTB = 0b00001111; //
        Delay_ms(500); // 500 milliseconds break
        PORTB = 0b00011111; //
    }
}
```

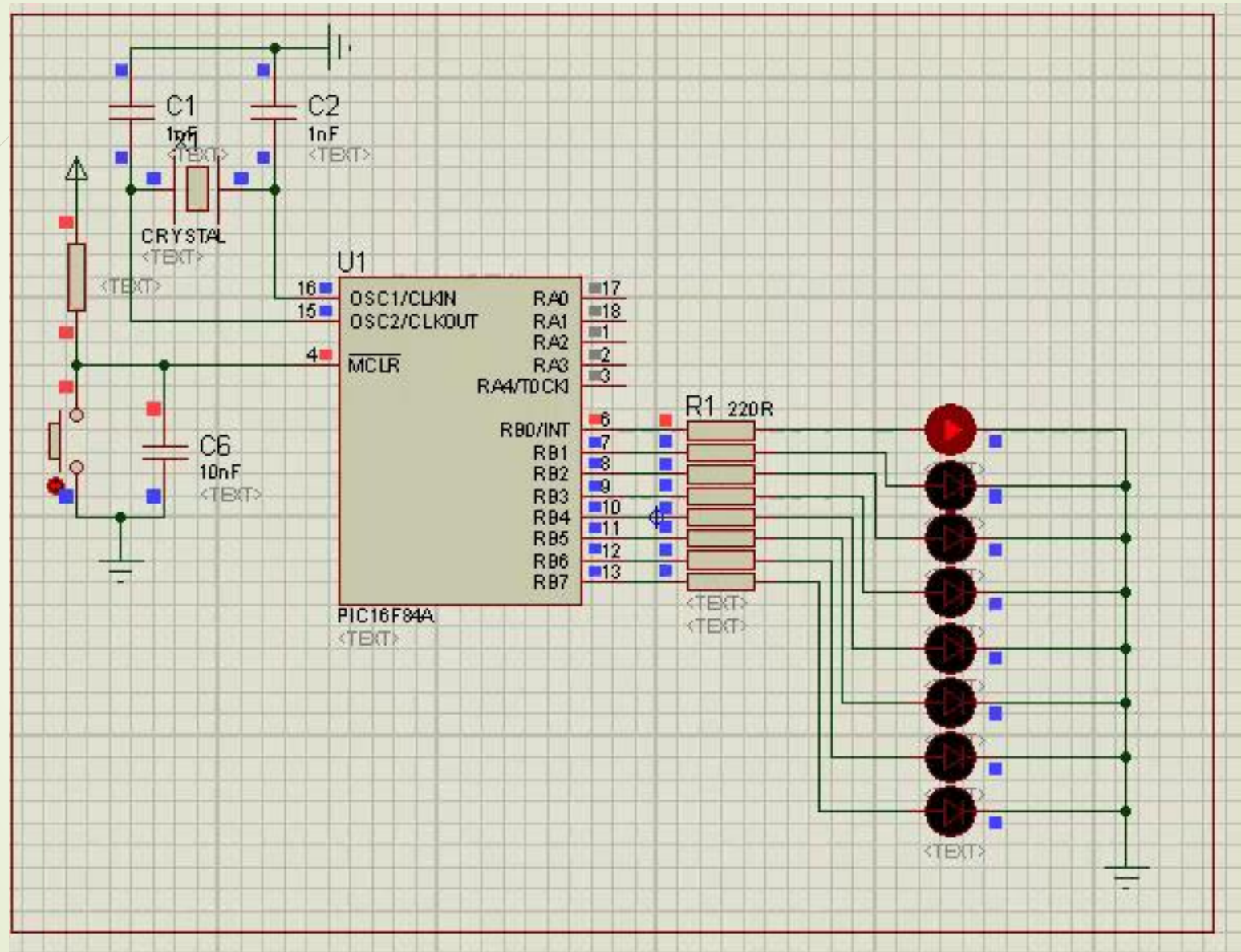
```
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00111111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b01111111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b11111111; //
Delay_ms(500); // 500 milliseconds break

// Sequence lighting
PORTB = 0b11111111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b01111111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00111111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00011111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00001111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00000111; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00000011; //
Delay_ms(500); // 500 milliseconds break
PORTB = 0b00000001; //
Delay_ms(500); // 500 milliseconds break
} while(1); // end of the endless loop
} // end of the program
```

Open the program with MikroC



# Result of the simulation



Play the video  
and observe

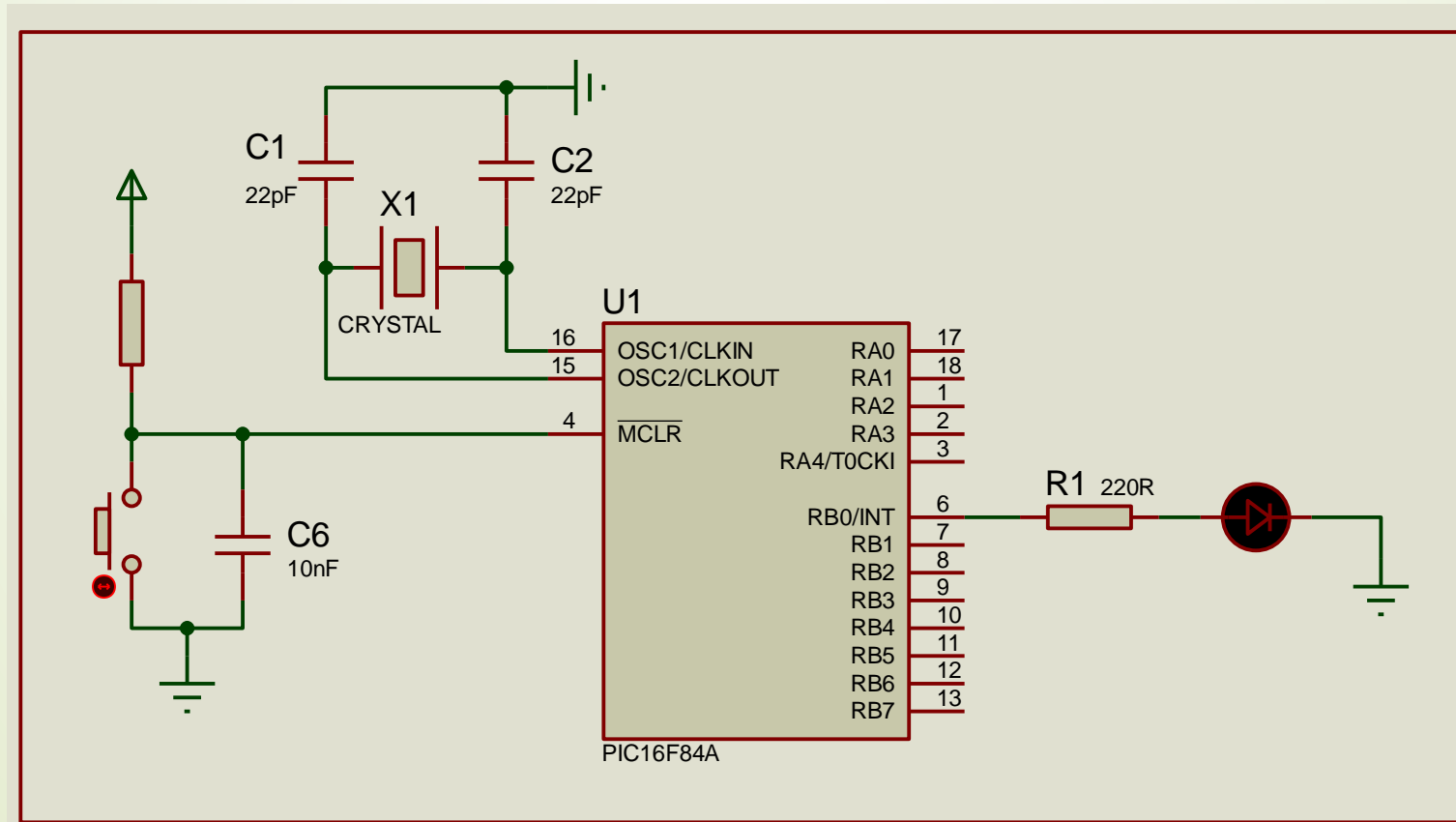
# Project 3: Blink the only LED connected to the RB0 pin

The aim of this project is to show how to target a particular pin and control it no matter the state of the pins of the same port. For that purpose, we will set the concerned pin (RB0) as output. It means that we will attribute the value zero to the bit corresponding to the concerned pin in the TRISB register.

Therefore, we have:

***TRISB = 0b1111110; // RB0 is set as output while the others are inputs***

The diagram



# The program

*///////////////// Declaration of global variables ///////////////////*

*///////////////// Declaration of functions if they exist ///////////////////*

*///////////////// Main Program ///////////////////*

*void main()*

*{*

*//////// Initializations //////////*

*TRISB = 0b1111110; // RB0 is set as output while the others are inputs*

*TRISA = 0b0001111; // all pins of PORTA set as input*

*//////// Definition of the endless loop in which the program will be  
permanently running //// //////////*

*do*

*{*

*//Programme to be executed*

*PORTB.F0 = 1; // RB0 is high or on*

*Delay\_ms(500); // 500 milliseconds break*

*PORTB.F0 = 0; // RB0 is low or off*

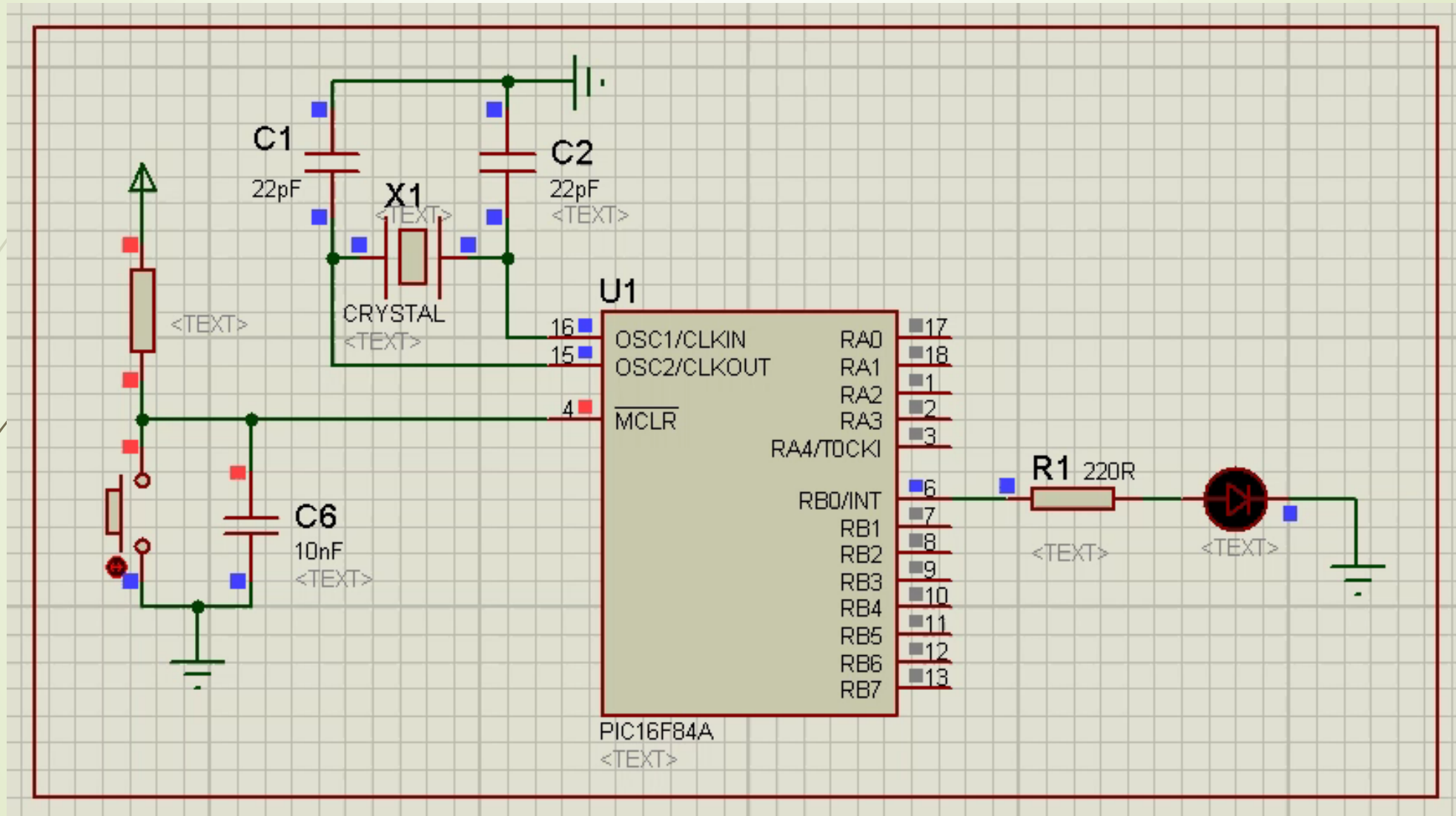
*Delay\_ms(500); // 500 milliseconds break*

*} while(1); // end of the endless loop*

*} // end of the program*



# Result of the simulation

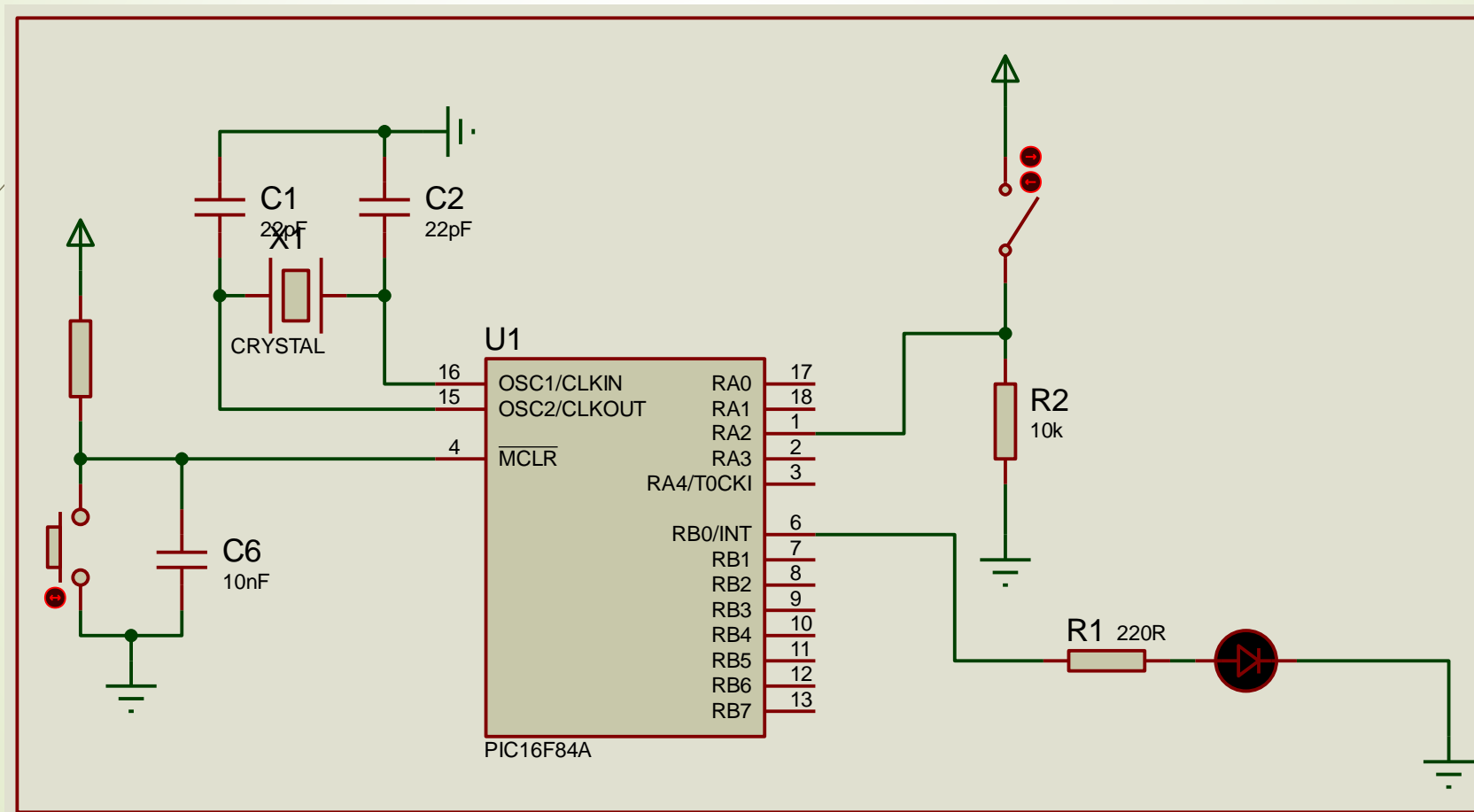


Play the video  
and observe

# Project 4: Testing the logic state of a pin before taking a decision

Since the beginning, we have been using ports only as output. In this example, we introduce the usage of ports as input. An input in a system can be a signal coming from a sensor. In this project, we will the LED connected to the RB0 pin under the condition that the switch is ON.

The diagram



# The program

*///////////////// Declaration of global variables ///////////////////*

*///////////////// Declaration of functions if they exist ///////////////////*

*///////////////// Main Program ///////////////////*

*void main()*

*{*

*//////// Initializations //////////*

*TRISB = 0b11111110; // RB0 is set as output while the others are inputs*

*TRISA = 0b00011111; // all pins of PORTA set as input*

*//////// Definition of the endless loop in which the program will be permanently running //// //////////*

*do*

*{*

*//Programme to be executed*

*if(PORTA.F2 == 1) // Testing if the switch is ON*

*{*

*// blinking*

*PORTB.F0 = 1; // RB0 is high or on*

*Delay\_ms(500); // 500 milliseconds break*

*PORTB.F0 = 0; // RB0 is low or off*

*Delay\_ms(500); // 500 milliseconds break*

*}*

*else*

*{*

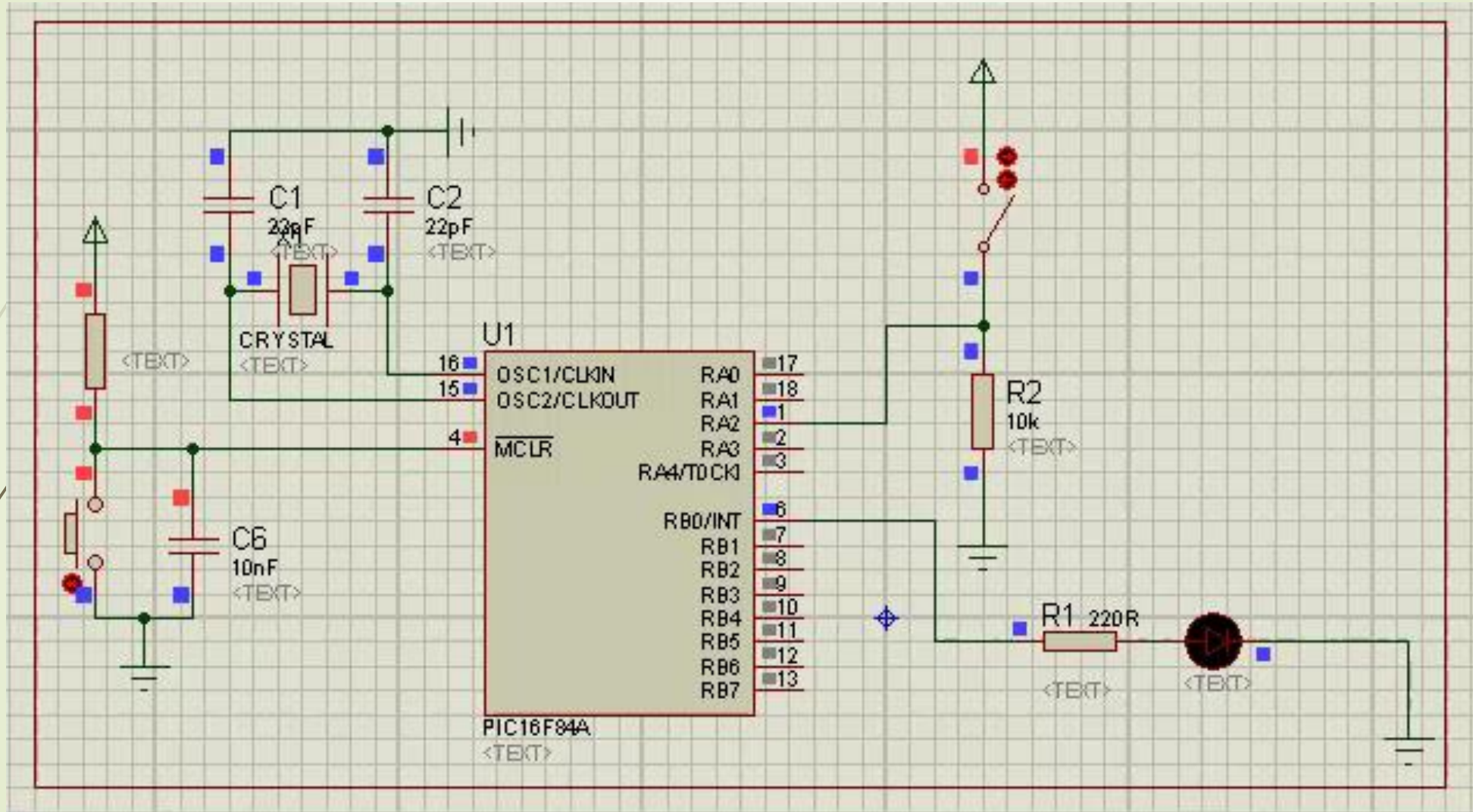
*PORTB.F0 = 0; // put RB0 off if RA2 is at logic 0 (switch off)*

*}*

*} while (1); // end of the endless loop*

*} // end of the program*

# Result of the simulation



Play the video  
and observe



# Project 5: Using buttons

In the MikroC environment, buttons are managed with a function called ***Button***. The syntax to use that function is as follows:

***Button (&PORT, pin Number , ON time, Active state);***

***&PORT*** is used to select the port on which the button is connected

***pin Number*** points on the particular pin where the button is connected. It is number whose value varie from 0 to 7, depending on the port.

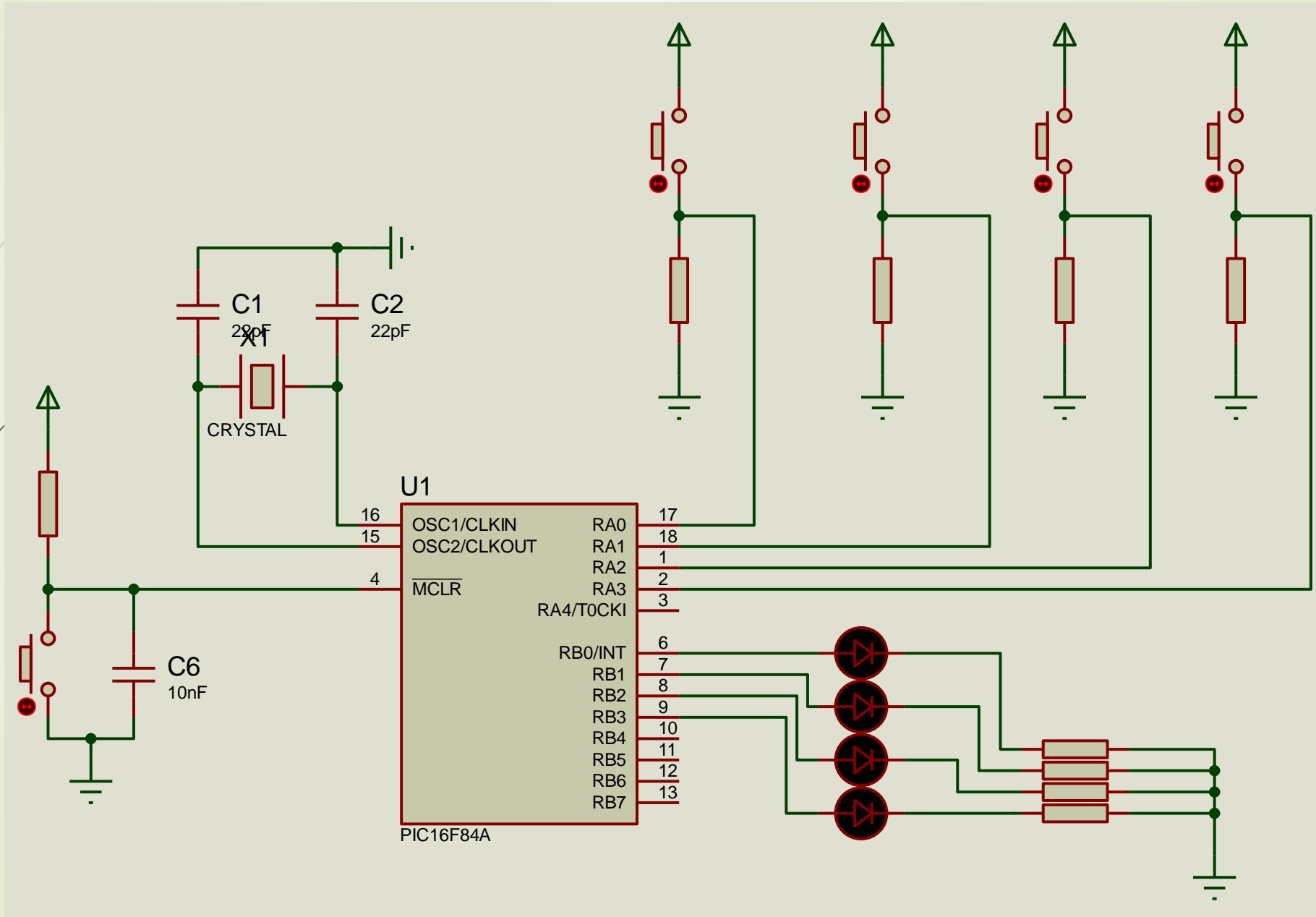
***ON time*** is a short integer used to indicate the maximum duration of the action on the button. It is given in milliseconds

***Active state*** determines the logic state of the targeted pin when the button is pressed.

Example

```
if (Button (&PORTA, 1, 100, 1))  
{  
    PORTB.F1 = 1; // RB1 is high or on  
}
```

# The diagram





# The program

*////////// Declaration of global variables //////////*

*////////// Declaration of functions if they are //////////*

*////////// Main Program //////////*

*void main()*

*{*

*//////// Initializations //////////*

*TRISB = 0x00; // all PORTB set as output*

*TRISA = 0x1F; // all PORTA set as input*

*PORTB = 0x00; // initialization of PORTB to 0*

*//////// Definition of the endless loop in which the program will be permanently running ////*

*do*

*{*

*//Program to be executed*

*if (Button(&PORTA, 0, 100, 1))*

*{*

*PORTB.F0 = 1; // RB0 is high or on*

*}*

*if (Button(&PORTA, 1, 100, 1))*

*{*

*PORTB.F1 = 1; // RB1 is high or on*

*}*

*if (Button(&PORTA, 2, 100, 1))*

*{*

*PORTB.F2 = 1; // RB2 is high or on*

*}*

*if (Button(&PORTA, 3, 100, 1))*

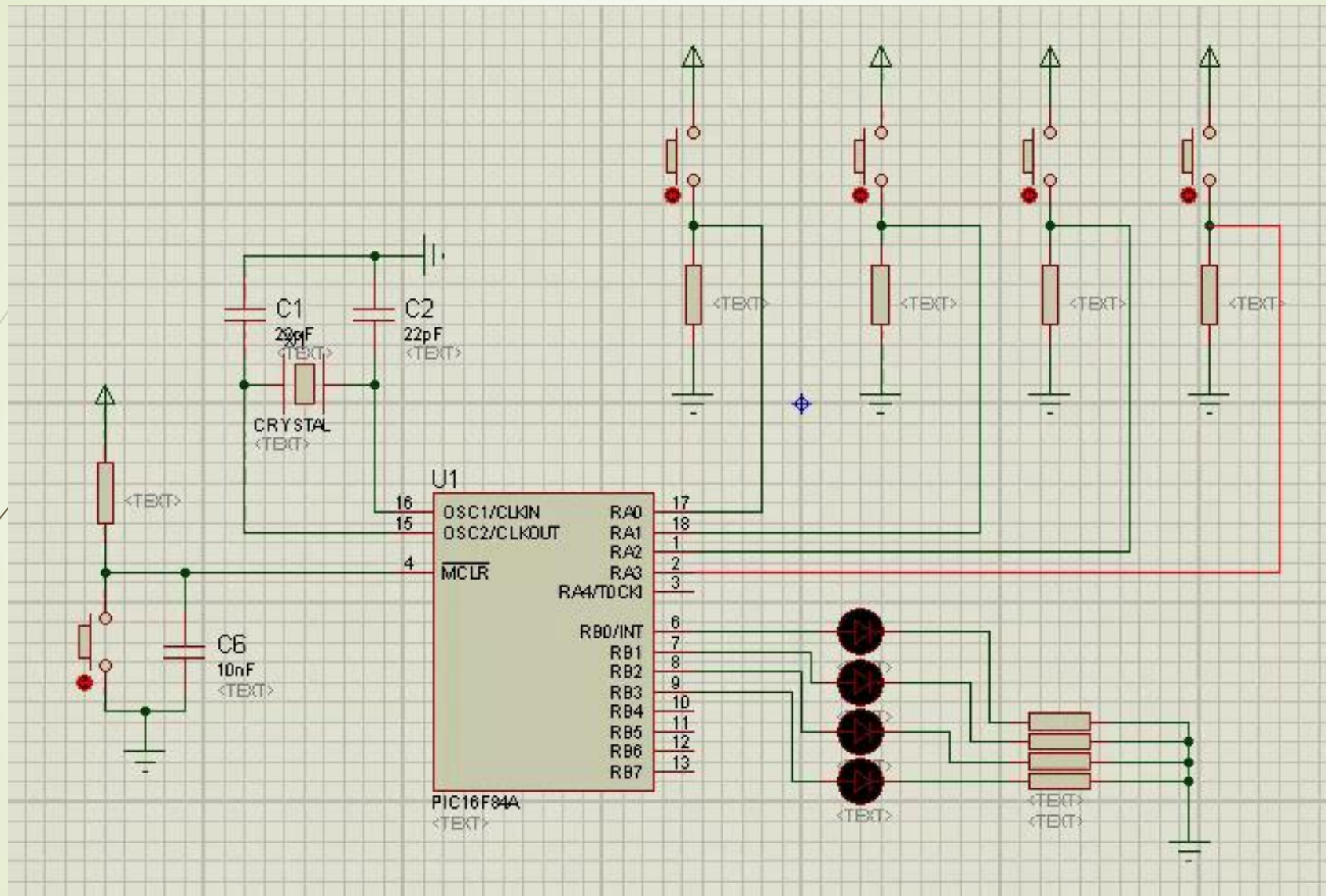
*{*

*PORTB.F3 = 1; // RB3 is high or on*

*}*

*} while(1); // end of the endless loop*

*} // end of the program*



Play the video  
and observe

# Project 6: Create and use functions

A function is a sub-program written out of the main function that we can call and execute at any time in the main program. MikroC compiler provides many functions, and we have been using some of them (**Delay\_ms()**, **Button()**...), but you can create personalized functions.

The syntax to create a function without parameters is as follows:

```
void name_of_the_function(parameters)  
    {  
        // statements of the function  
    }
```

*NB : all the other forms of functions can still be created in MikroC*

***Example of execution***

*Nom\_de\_la\_fonction()*

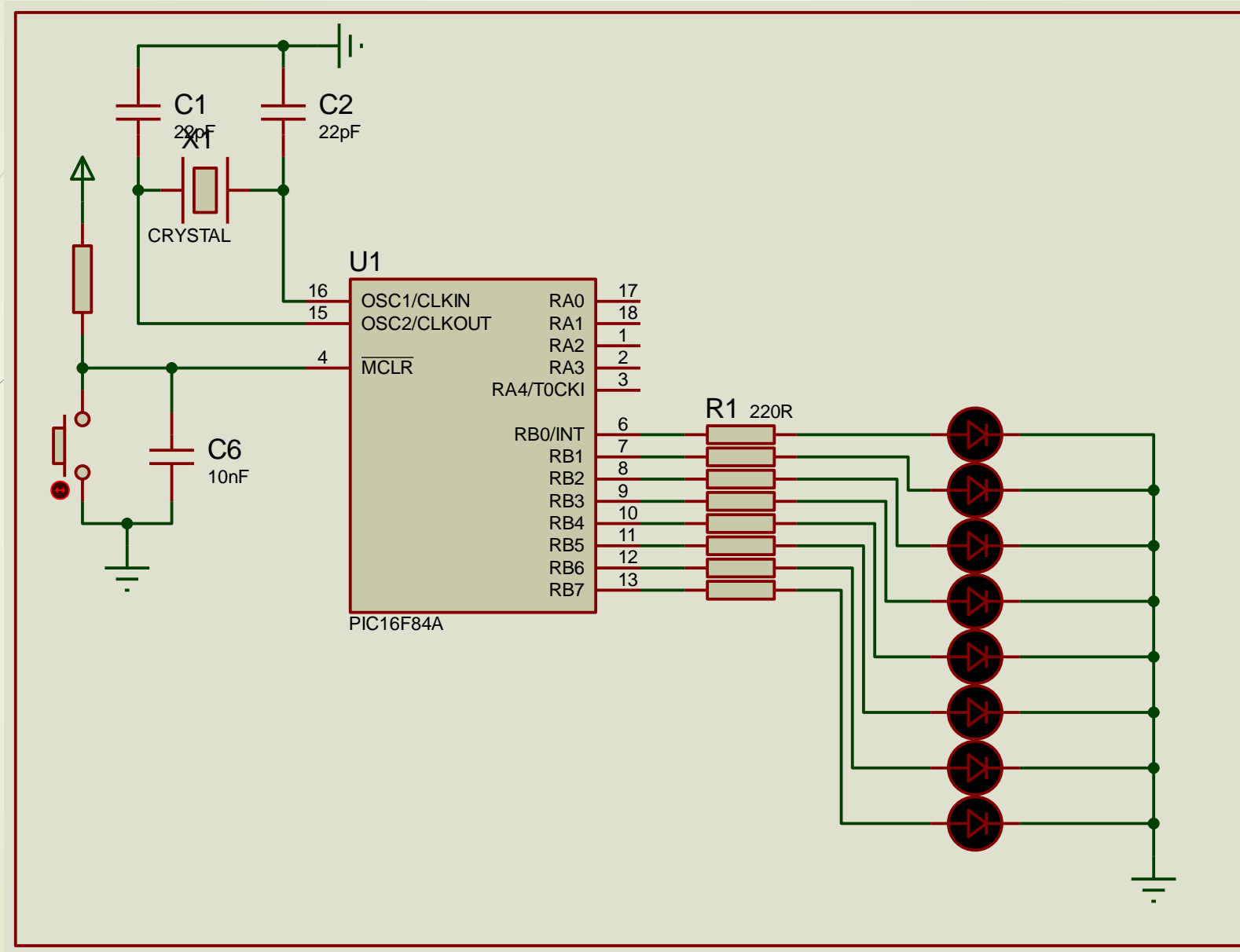


For a better use of functions, we propose to realize a project whose log book includes creation of functions as follows:

- Initialise the PIC
- Gradually the LEDs connected on PORTB
- Gradually put OFF LEDs on PORTB, starting on the last one
- Blink all the LEDs on PORTB twice.

We observe that this project integrate almost all the functionalities developed in the previous projects. For each step of the project, we will define functions that we will call when need be.

# The diagram





# The program and Result of the simulation

Open the program with MikroC



