جامعة بيرزيت

BIRZEIT UNIVERSITY

Electrical and Computer Engineering

Computer Design Lab – ENCS4110

ARM Basic I/O

Introduction

A microcontroller (or MCU) is a computer-on-a-chip. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor (the kind used in a PC).

A microcontroller is a single integrated circuit, commonly with the following features:

- Central processing unit (CPU) - ranging from small and simple 4-bit processors to sophisticated 32- or 64-bit processors
- Memory (SRAM and Flash memory)
- Input/output interfaces such as serial ports (UARTs)
- Peripherals such as timers and watchdog
- Many include analog-to-digital converters
- Clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit

Microcontroller versus Microprocessor

A microcontroller differs from a microprocessor in many ways. The first and most important difference is its functionality. In order that the microprocessor may be used, other components such as memory must be added to it. Even though the microprocessors are considered to be powerful computing machines, their weak point is that they are not adjusted to communicating to peripheral equipment. On the other hand, the microcontroller is designed to be all of that in one. No other specialized external components are needed for its application because all necessary circuits are already built into it.

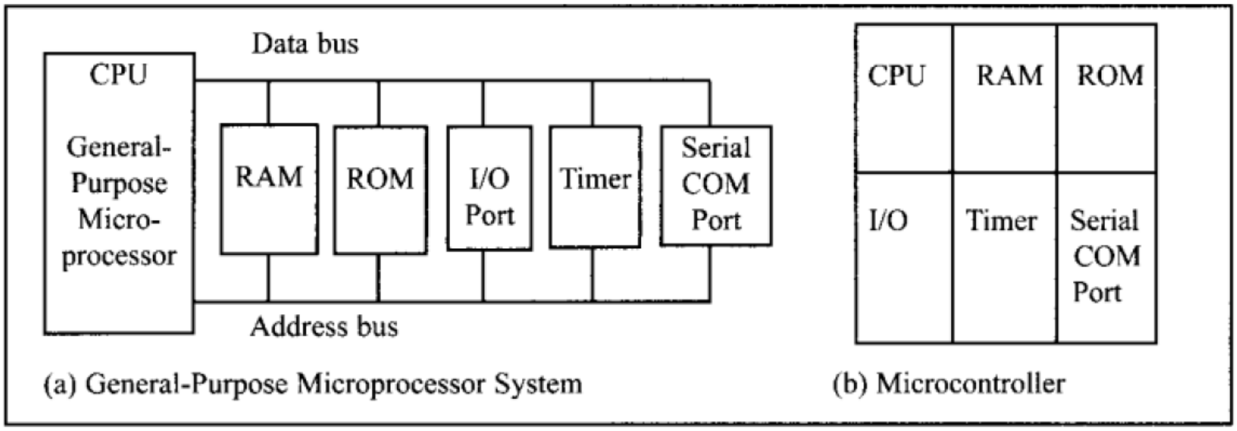


Figure1-1. Microcontroller versus Microprocessor

ARM Processor

ARM-Advanced RISC Machine, is one of a family CPUs based on a reduced instruction set computer (RISC) architecture developed by British company ARM Holdings. A RISC-based computer design approach means ARM processors require significantly fewer transistors than typical processors in average computers. This approach reduces costs, heat and power use. These are desirable traits for light, portable, battery-powered devices—including smart phones, laptops, tablet and notepad computers), and other embedded systems.

Memory Unit

□ **Flash memory**

The contents of this memory can be written and cleared practically an unlimited number of times, the microcontrollers with Flash ROM are ideal for learning, experimentation and small-scale manufacture.

□ **Random Access Memory(RAM)**

Once the power supply is off the contents of RAM (Random Access Memory) is cleared. It is used for temporary storing data and intermediate results created and used during the operation of the microcontroller.

□ **Electrically Erasable Programmable ROM (EEPROM)**

The contents of the EEPROM may be changed during operation (similar to RAM), but remains permanently saved even upon the power supply goes off (similar to ROM). Accordingly, an EEPROM is often used to store values, created during operation, which must be permanently saved.

LPC21xx Microcontrollers

The LPC2131/2132/2134/2136/2138 microcontrollers are based on a 32/16 bit ARM7TDMI-S CPU with real-time emulation and embedded trace support, that combines the microcontroller with 32 kB, 64 kB, 128 kB, 256 kB and 512 kB of embedded high speed Flash memory. A 128-bit wide memory interface and a unique accelerator architecture enable 32-bit code execution at maximum clock rate.

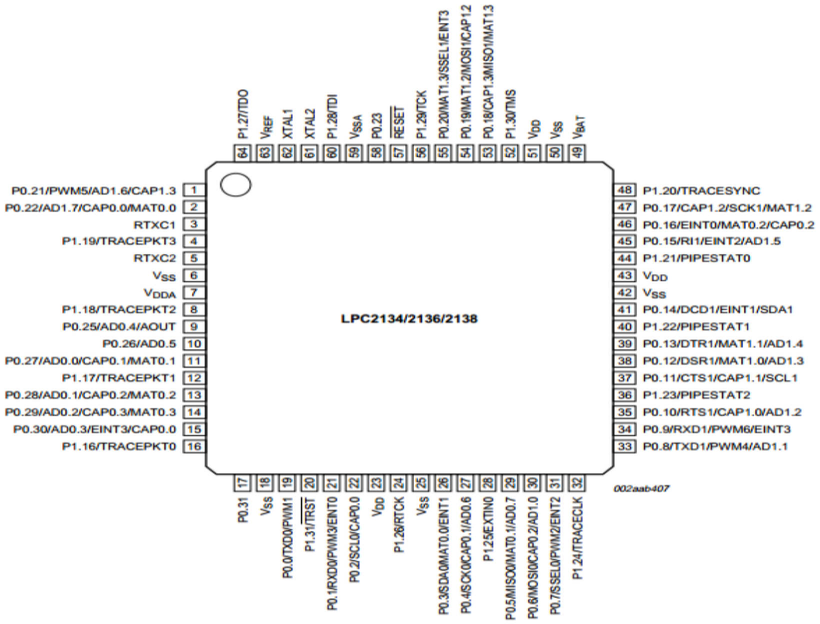
Due to their tiny size and low power consumption, these microcontrollers are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. With a wide range of serial communications interfaces and on-chip SRAM options of 8/16/32 kB, they are very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, single or dual 10-bit 8 channel ADC(s), 10-bit DAC, PWM channels and 47 GPIO lines with up to nine edge or level sensitive external interrupt pins make these microcontrollers particularly suitable for industrial control and medical systems.

LPC2138 Microcontroller

This is the microcontroller we are going to use during our lab. This microcontroller has the following main features :

- an ARM7TDMI-S based high-performance 32-bit RISC Microcontroller
- 512KB on-chip Flash ROM
- 32KB RAM
- Two 8-ch 10bit ADC
- Two UARTs
- Single 10-bit DAC
- Two I2C serial interfaces
- Two SPI serial interfaces
- Three 32-bit timers
- Watchdog Timer
- Vectored Interrupt Controller
- In-System Programming (ISP)
- In-Application Programming (IAP)
- Up to 47 general purpose I/O pins.
- CPU clock up to 60 MHz
- On-chip crystal oscillator and On-chip PLL.

Pin Diagram



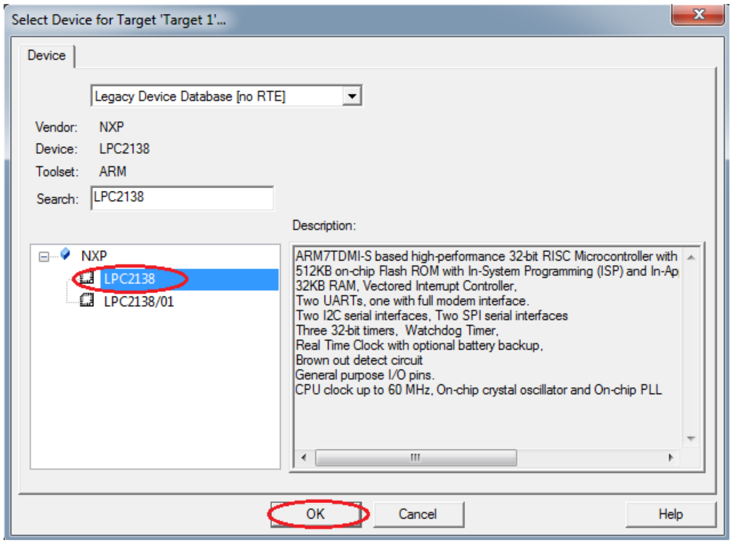
Software Tools

These programs are the backbone of the microprocessor and microcontroller based systems; since using Keil we can build the software of the project using C, and then we can simulate the project virtually using Proteus, finally we can download the program on the microcontroller and see the practical results.

Keil Legacy Device Database

In order to have NXP LPC2138 controller in the device selection when creating new project with the Keil uversion, you need to download and install Keil.LPC1700_DFP.2.7.0.pack

https://armkeil.blob.core.windows.net/legacy/MDK79525.EXE



Legacy Device Database [no RTE]

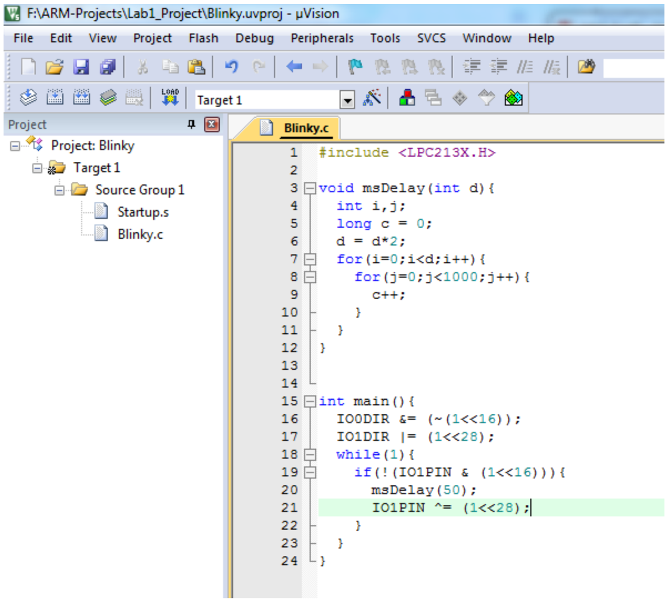
Vendor: NXP
Device: LPC2138
Toolset: ARM
Search: LPC2138

NXP
LPC2138
LPC2138/01

Description:
ARM7TDMI-S based high-performance 32-bit RISC Microcontroller with 512KB on-chip Flash ROM with In-System Programming (ISP) and In-Ap
32KB RAM, Vectored Interrupt Controller,
Two UARTs, one with full modem interface.
Two I2C serial interfaces, Two SPI serial interfaces
Three 32-bit timers, Watchdog Timer,
Real Time Clock with optional battery backup,
Brown out detect circuit
General purpose I/O pins.
CPU clock up to 60 MHz, On-chip crystal oscillator and On-chip PLL

Create a new project with Keil MDK

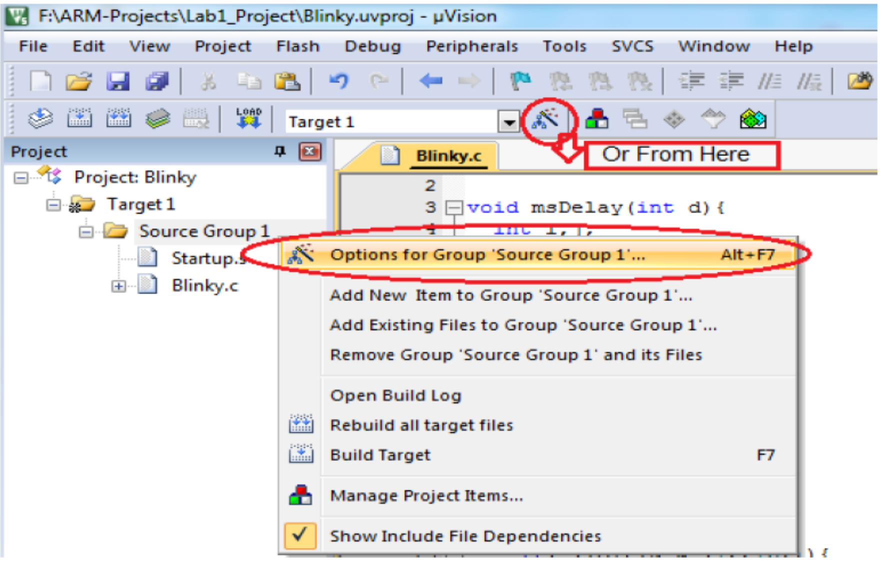
Create new project using **LPC2138** controller and copy the following c code into a new c file. This is a simple program to toggle the LED on the LPC2138 microcontroller


F:\ARM-Projects\Lab1_Project\Blinky.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: Blinky
Target 1
Source Group 1
Startup.s
Blinky.c
Blinky.c

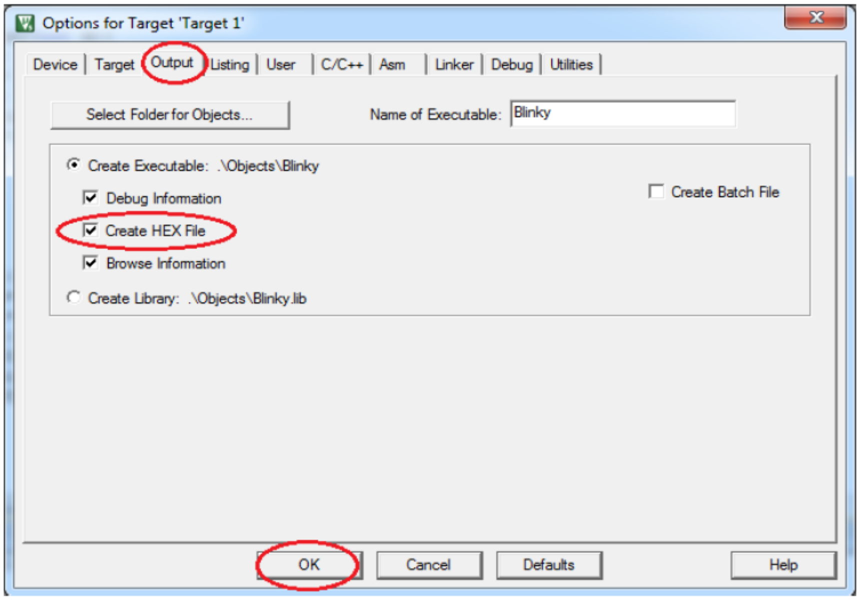
```c
#include <LPC213X.H>

void msDelay(int d){
  int i,j;
  long c = 0;
  d = d*2;
  for(i=0;i<d;i++){
    for(j=0;j<1000;j++){
      c++;
    }
  }
}


int main(){
  IO0DIR &= (~(1<<16));
  IO1DIR |= (1<<28);
  while(1){
    if(!(IO1PIN & (1<<16))){
      msDelay(50);
      IO1PIN ^= (1<<28);
    }
  }
}
```

Generate the Hex file machine code

1- Right click on the “Source Group1” and choose “Options for Target “Target 1”. Or simply choose “Option for Target” icon as shown.


F:\ARM-Projects\Lab1_Project\Blinky.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Or From Here
Project
Project: Blinky
Target 1
Source Group 1
Startup.s
Blinky.c
Blinky.c
void msDelay(int d){
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Open Build Log
Rebuild all target files
Build Target F7
Manage Project Items...
Show Include File Dependencies

2- Next, from the ”Options for Target “Target 1”” dialog, click on “Output” from the upper menu bar and check on “create HEX file” and press “Ok” button.



Name of Executable: Blinky

● Create Executable: .\Objects\Blinky

☑ Debug Information

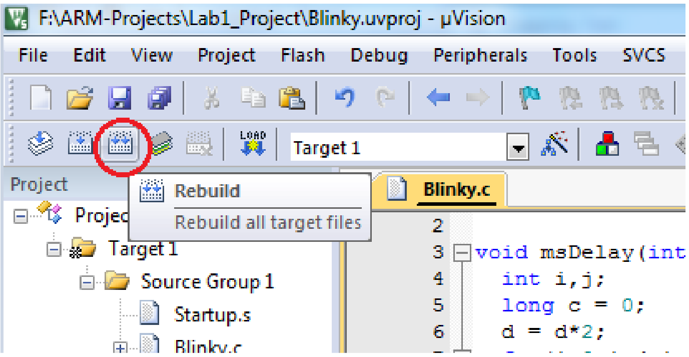
☑ Create HEX File

☑ Browse Information

○ Create Library: .\Objects\Blinky.lib

□ Create Batch File

3- Click on “Rebuild” to rebuild the file to create a hex file.



```
void msDelay(int
  int i,j;
  long c = 0;
  d = d*2;
```

Proteus Program

Proteus is a simulation program used to simulate hardware connections and check that they are error-free before connecting them.

Installation of Proteus

1- Run the “Proteus 8 professionals sp0 build 15417” as an administrator.

2- Press next until you reach the window which asks for the key.

3- From browse for key; browse until you find the file where the key exist.

4- Then click on Grassington North Yorkshire.lxk and then press install.

5- Then choose yes and close the window, after that press Next and choose Typical install to install the program.

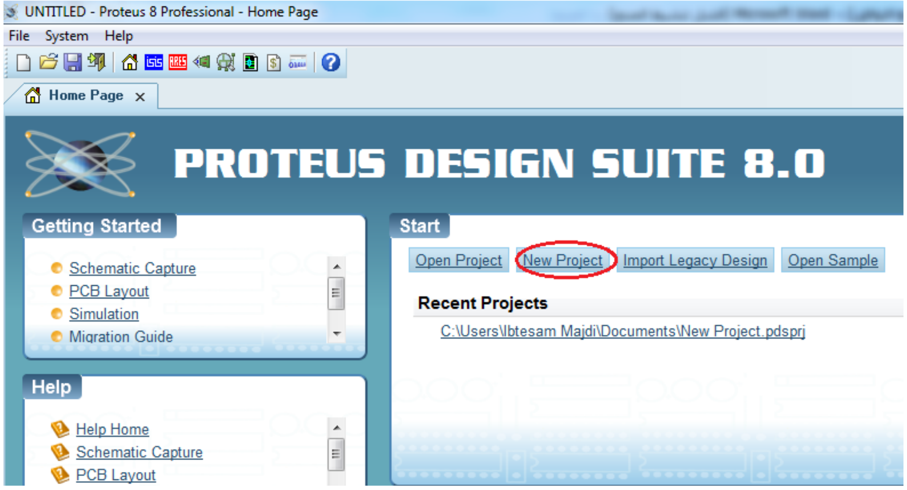
6- After installation, for Crack, copy (“PIN”,”Models”,”Help”,”LICENCE”) and past them on the folder where the Protues installed in your PC.

7- Run the LICENCE and press install.

8- Finally, Choose “Ok” and then “Close”

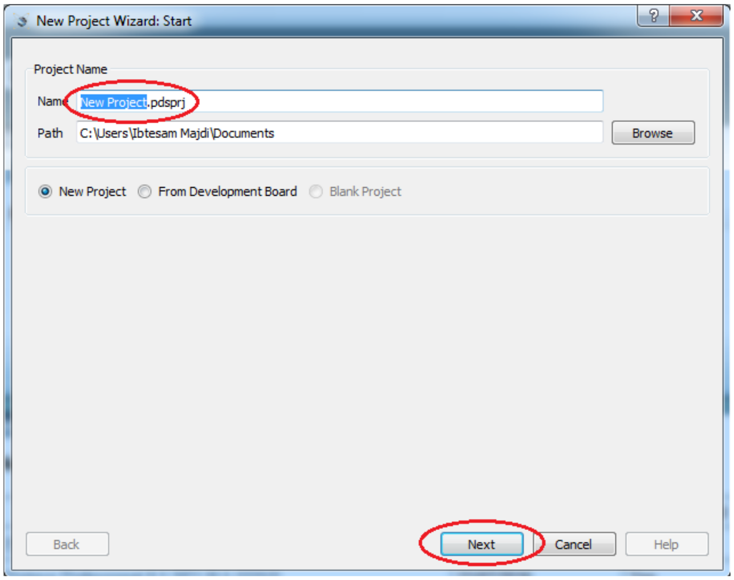
Create a new project with Proteus

1- Open Proteus and click on “New Project” button.



C:\Users\Ibtesam Majdi\Documents\New Project.pdsprj

2- Name your project with suitable name and click “Next”.

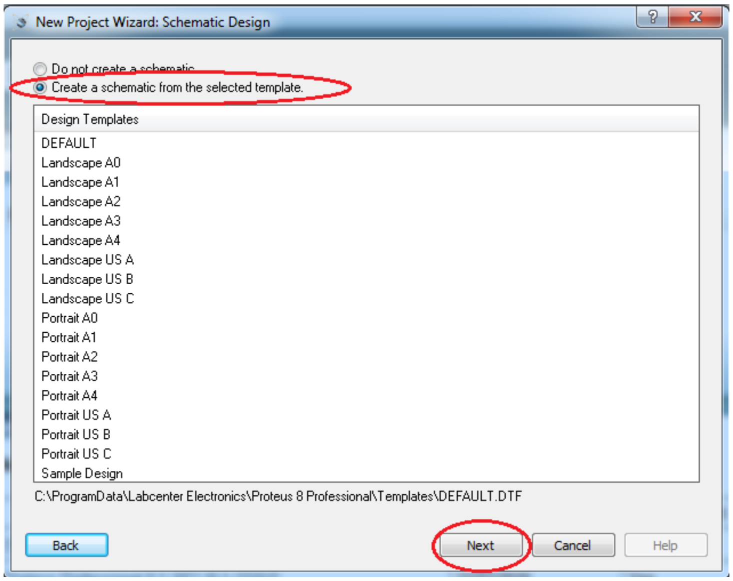


Name: New Project.pdsprj

Path: C:\Users\Ibtesam Majdi\Documents

● New Project ○ From Development Board ○ Blank Project

3- Choose “Creat a schematic from the selected template” and click “Next”.



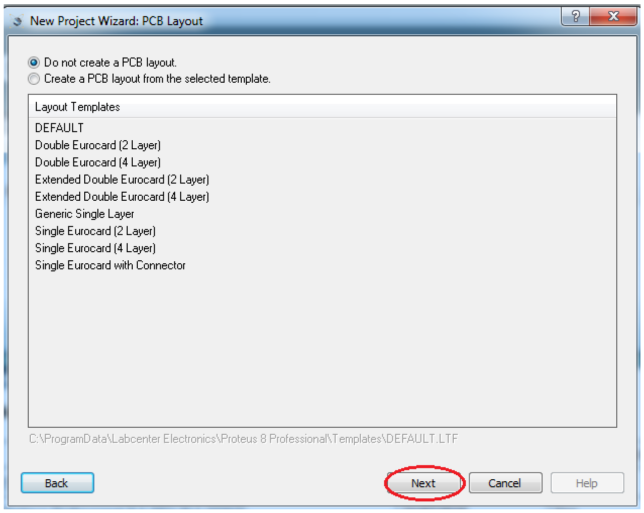
○ Do not create a schematic.

● Create a schematic from the selected template.

Design Templates
DEFAULT
Landscape A0
Landscape A1
Landscape A2
Landscape A3
Landscape A4
Landscape US A
Landscape US B
Landscape US C
Portrait A0
Portrait A1
Portrait A2
Portrait A3
Portrait A4
Portrait US A
Portrait US B
Portrait US C
Sample Design

C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\Templates\DEFAULT.DTF

4- Click “ Next”.



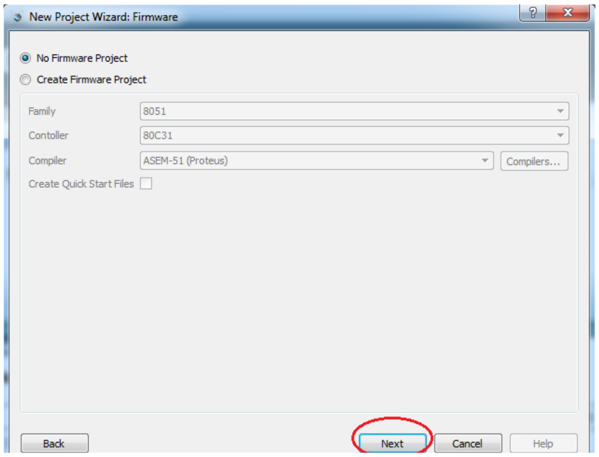
● Do not create a PCB layout.

○ Create a PCB layout from the selected template.

Layout Templates
DEFAULT
Double Eurocard (2 Layer)
Double Eurocard (4 Layer)
Extended Double Eurocard (2 Layer)
Extended Double Eurocard (4 Layer)
Generic Single Layer
Single Eurocard (2 Layer)
Single Eurocard (4 Layer)
Single Eurocard with Connector

C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\Templates\DEFAULT.LTF

5- Click “Next”.



● No Firmware Project

○ Create Firmware Project

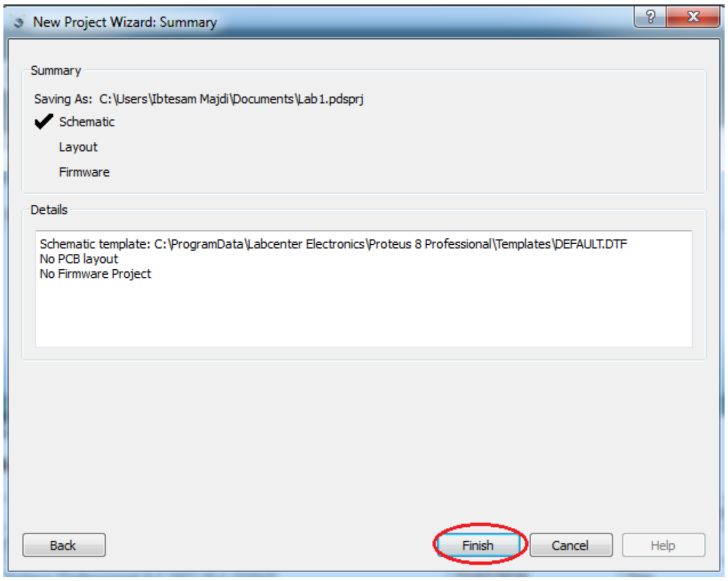
Family 8051

Contoller 80C31

Compiler ASEM-51 (Proteus)

Create Quick Start Files □

6- Click “Finish”. And now your workspace is ready.


New Project Wizard: Summary

Summary

Saving As: C:\Users\Ibtesam Majdi\Documents\Lab1.pdsprj

✓ Schematic

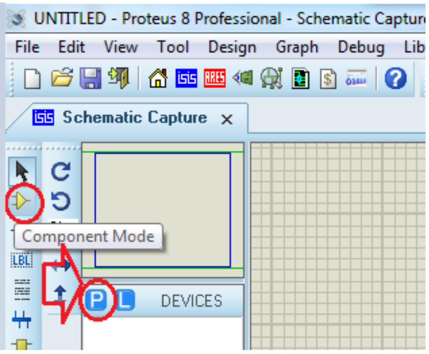
Layout

Firmware

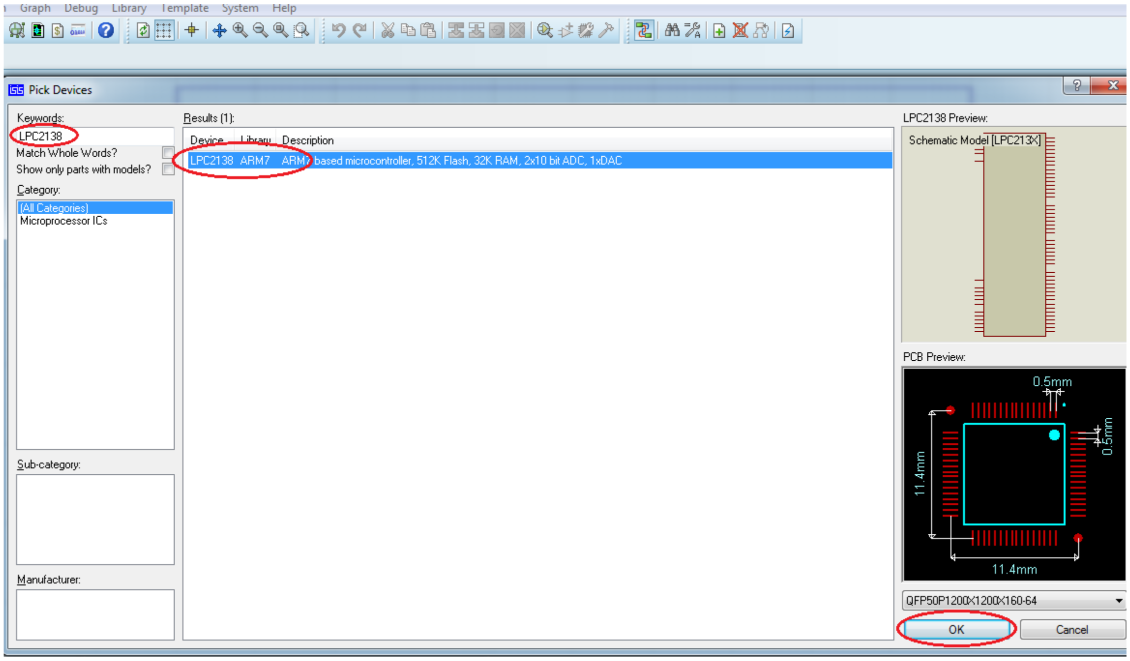
Details

Schematic template: C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\Templates\DEFAULT.DTF
No PCB layout
No Firmware Project

7- Press the “Component mode” to choose the components from “P” icon



8- As an Example, we deal with “LPC2138 Microcontroller” in this lab, from the Keyword field we write the keyword “LPC2138” and choose it then press “Ok” button.

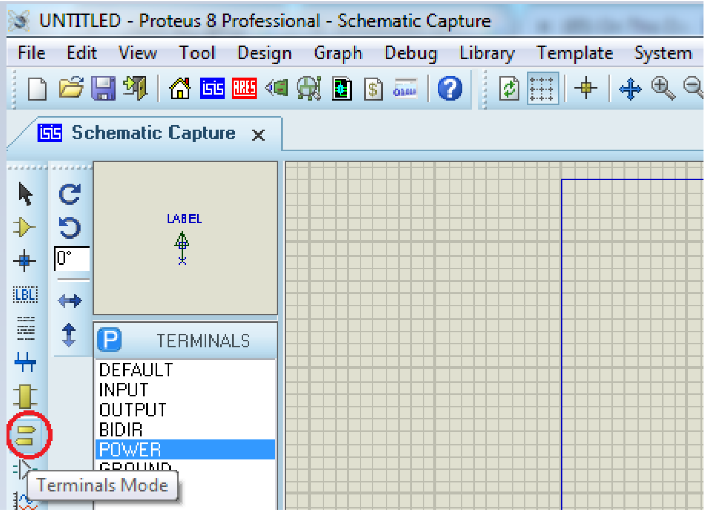


Keywords: LPC2138

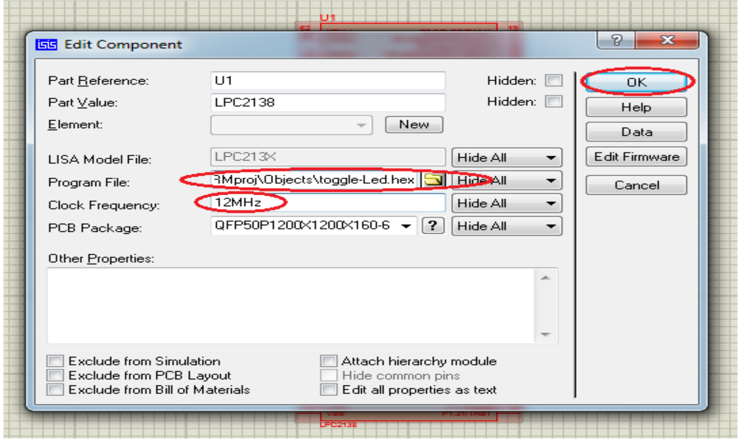
Device	Library	Description
LPC2138	ARM7	ARM7 based microcontroller, 512K Flash, 32K RAM, 2x10 bit ADC, 1xDAC

QFP50P1200X1200X160-64

9- From “Terminal Mode” we can find the “Power” and the “Ground”.


UNTITLED - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System
Schematic Capture
0°
LABEL
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
Terminals Mode

10- To install your program to the microcontroller, double click on the “LPC2138” , then from the “Edit Component” dialog click on “search file” field and browse for your .hex file and then press “OK”.



Part Reference: U1 Hidden: □

Part Value: LPC2138 Hidden: □

Element:

LISA Model File: LPC213X

Program File: RMproj\Objects\toggle-Led.hex

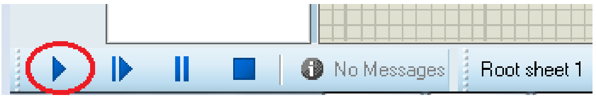
Clock Frequency: 12MHz

PCB Package: QFP50P1200X1200X160-6

□ Exclude from Simulation
□ Exclude from PCB Layout
□ Exclude from Bill of Materials

□ Attach hierarchy module
□ Hide common pins
□ Edit all properties as text

11- You can run the simulation as shown.



Basic I/O

Generally, every microcontroller has at minimum two registers associated with each of I/O port. They are Data Register and Direction Register. The Direction register is used to make the pin either input or output. After the Direction register is properly configured, then we use the Data register to actually write to the pin or read data from the pin. For example, suppose you want your device to turn three signal LEDs and simultaneously monitor the logic state of five sensors or push buttons. Some of ports need to be configured so that there are three outputs (connected to the LEDs) and five inputs (connected to sensors).

LPC2138 GPIO Programming

LPC2138 MCU has 2 ports, port0 and port1.

□ **Port 0** is a 32-bit I/O port with individual direction controls for each bit.
Total of 31 pins of the Port 0 can be used as a general purpose bi-directional digital I/Os while P0.31 is output only pin. The operation of port 0 pins depends upon the pin function selected via the pin connect block.Pin P0.24 is not available.

□ **Port1** is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 1 pins depends upon the pin function selected via the pin connect block. Pins 0 through 15 of port 1 are not available.

In LPC2138 MCU most of the pins are Multiplexed i.e. these pins can be configured to provide different functions. we'll explain this in upcoming labs. For now, just keep in mind that by default all functional pins are set as GPIO so we can direclty use them when learning GPIO usage. Also note, All GPIO pins are configured as Input after Reset by default

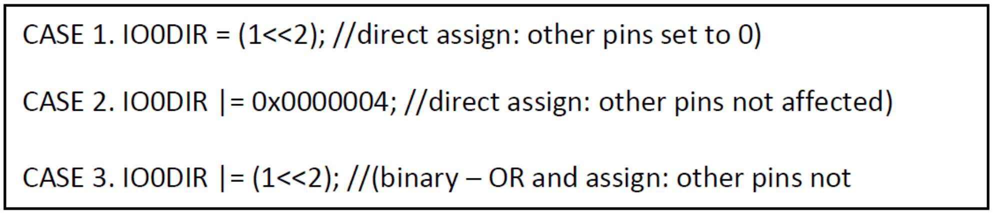
There are 4 registers we need to understand to program these ports.
1. **IOxPIN** (x=port number) : This register can be used to Read or Write values directly to the pins. Regardless of the direction set for the particular pins it gives the current state of the GPIO pin when read.

2. **IOxDIR** : This is the GPIO direction control register. Setting a bit to 0 in this register will configure the corresponding pin to be used as an Input while setting it to 1 will configure it as Output.

3**. IOxSET**: This register can be used to drive an output configured pin to logic 1. Writing Zero does NOT have any effect and hence it cannot be used to drive a pin to Logic 0.

4. **IOxCLR**: This register can be used to drive an output configured pin to logic 0. Writing Zero does NOT have any effect and hence it cannot be used to drive a pin to Logic 1.

Now setting Pin 2 of Port 0 as output can be done in various ways as show :



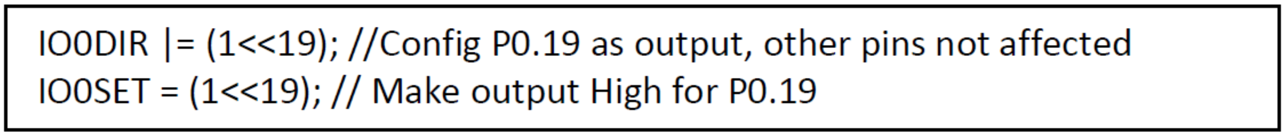
```
CASE 1. IO0DIR = (1<<2); //direct assign: other pins set to 0)

CASE 2. IO0DIR |= 0x0000004; //direct assign: other pins not affected)

CASE 3. IO0DIR |= (1<<2); //(binary – OR and assign: other pins not
```

□ **Case 1** must be avoided since we are directly assigning a value to the register. So while we are making P0.2 ‘1’ others are forced to be assigned a ‘0’ which can be avoided by ORing and then assigning Value.

□ **Case 2** can be used when bits need to be changed in bulk.

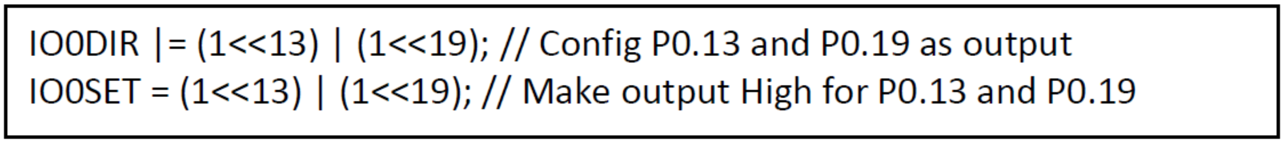
□ **Case 3** when some or single bit needs to be changed.

For example, Consider that we want to configure Pin 19 of Port 0 as output and want to drive it a high logic, this can be done as:



```
IO0DIR |= (1<<19); //Config P0.19 as output, other pins not affected
IO0SET = (1<<19); // Make output High for P0.19
```

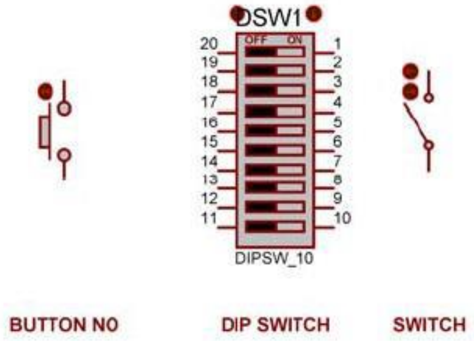
Also Consider that we want to configuring P0.13 and P0.19 as output and setting them high:



```
IO0DIR |= (1<<13) | (1<<19); // Config P0.13 and P0.19 as output
IO0SET = (1<<13) | (1<<19); // Make output High for P0.13 and P0.19
```

Switches and Buttons

Switches and buttons are the basic input devices used to supply certain voltage level for microcontrollers. Button/switch is a mechanical component, which connects or disconnects two points A and B over its contacts. By function, button/switch contacts can be normally open or normally closed



Buttons can be connected to the microcontroller in one of two ways:

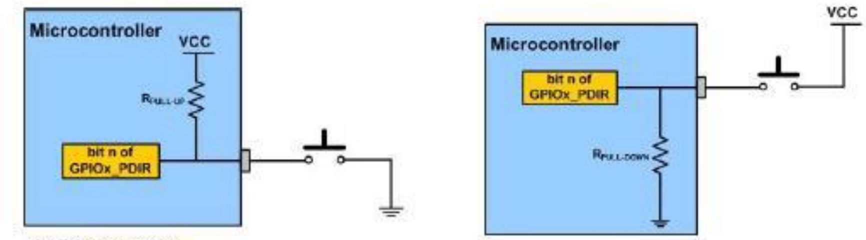
□ **Button with pull down resistor**

- When the button is presses, it will supply logical one (+5V) to MCU.
- When the button is not pressed, the pull down resistor will supply logical zero(0v) to the MCU.

☐ **Button with pull up resistor**

- When the button is presses, it will supply logical zero(0v) to the MCU.
- When the button is not pressed, the pull up resistor will supply logical one (+5V) to MCU.

These ways are used in order to avoid unknown case (Not Zero & Not 5V), so one of these ways are used to give ‘0’ or ‘1’ state normally. In LPC2138 the internal Pull-ups are enabled and implemented in port1 from pin 16 to 25, the default state of the pins configured as Input will be always high unless it is explicitly made low by connecting it to Ground



(a) Using Pull-up Resistor

(b) Using Pull-down Resistor

Lab Work:

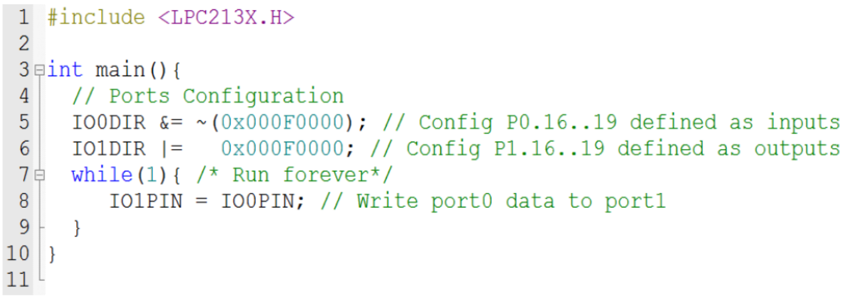
➢ Lab Work 1

You are going to use these keywords when you search for parts in Proteus:

Part	Keyword
Microcontroller	LPC2138
Resistor	res
Switch	dipsw
LEDs	led-
Button	Push

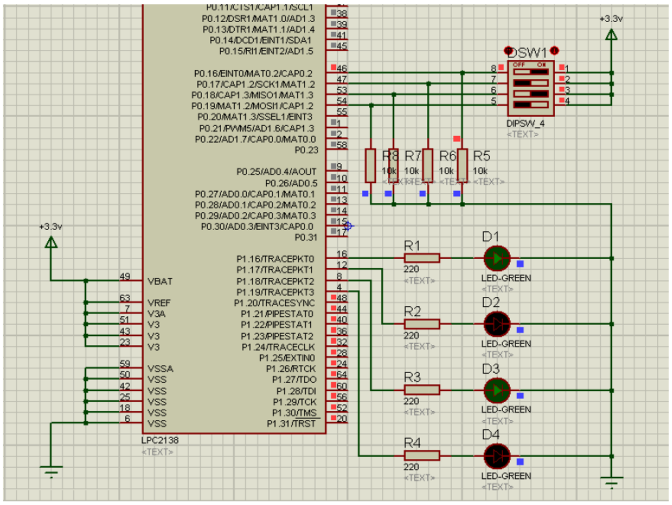
A. Write and simulate a program that reads data from port0 and send it to port1.

- ○ **Keil**



```
#include <LPC213X.H>

int main(){
  // Ports Configuration
  IO0DIR &= ~(0x000F0000); // Config P0.16..19 defined as inputs
  IO1DIR |=   0x000F0000; // Config P1.16..19 defined as outputs
  while(1){ /* Run forever*/
      IO1PIN = IO0PIN; // Write port0 data to port1
  }
}

```

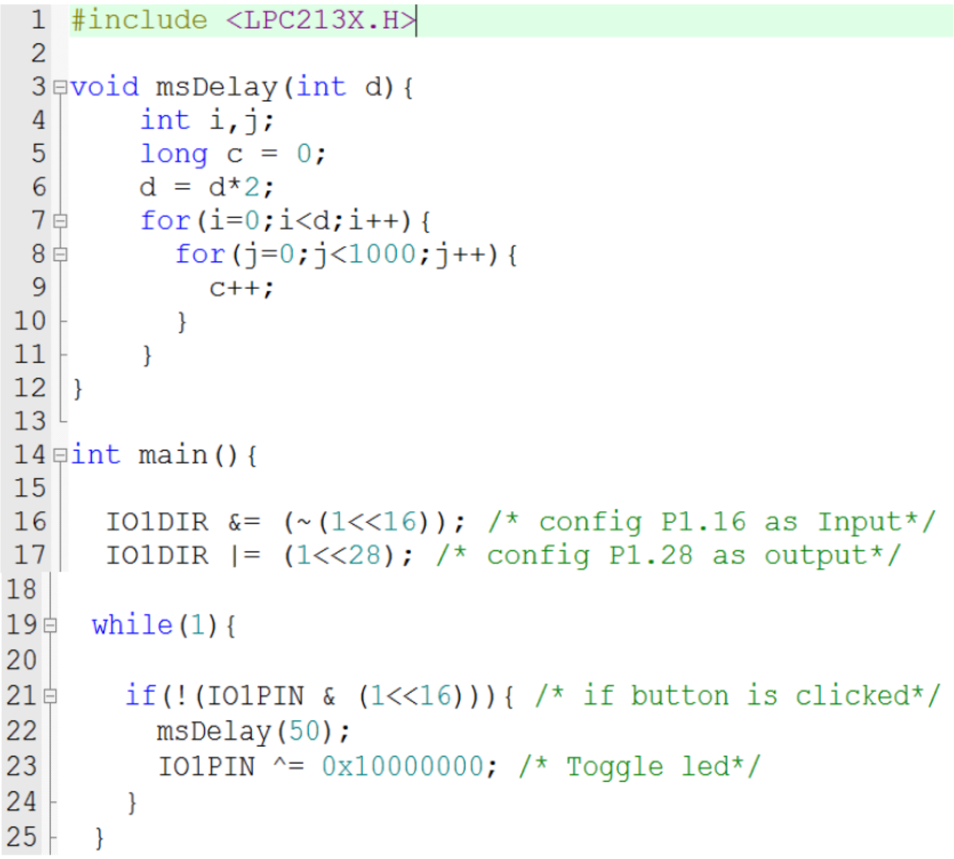
- Protues



B. Write a program that toggles a LED when clicking on a push button.

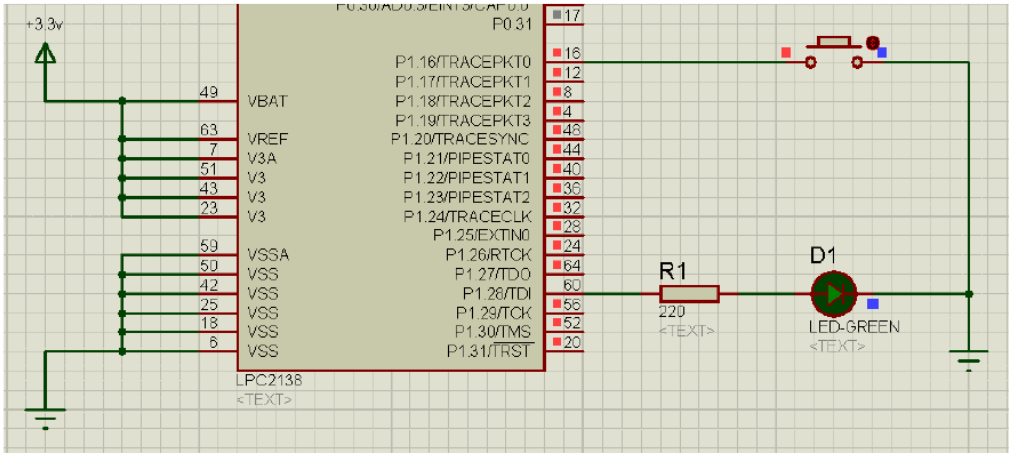
<u>Note</u>:
The needed time for the user to press the push button is approximately 200 ms, the speed of the LPC's work is very high, so 'while true' loop will be executed a lot of times through this period (200 ms), which means that the button has been pressed many times, but in fact the user did it just once!.
So we need some delay after button clicking detection.

- ## Keil



```
#include <LPC213X.H>

void msDelay(int d){
      int i,j;
      long c = 0;
      d = d*2;
      for(i=0;i<d;i++){
         for(j=0;j<1000;j++){
            c++;
         }
      }
}

int main(){

   IO1DIR &= (~(1<<16)); /* config P1.16 as Input*/
   IO1DIR |= (1<<28); /* config P1.28 as output*/

  while(1){

     if(!(IO1PIN & (1<<16))){ /* if button is clicked*/
       msDelay(50);
       IO1PIN ^= 0x10000000; /* Toggle led*/
     }
  }
```

- ## Protues



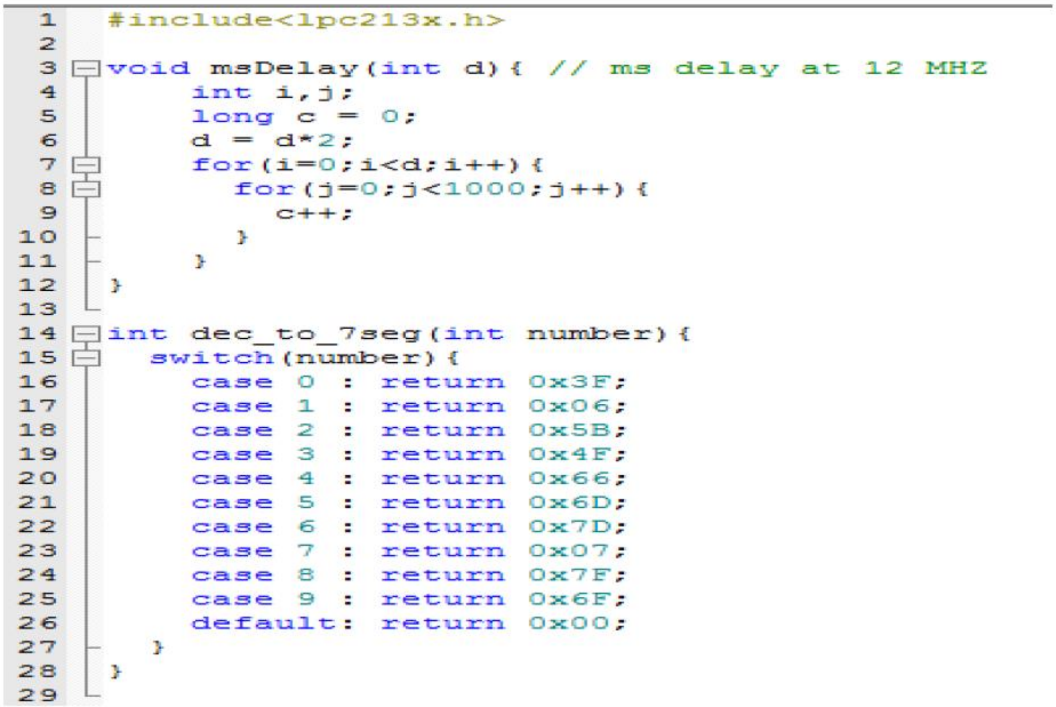
➢ Lab Work 2

You are going to use these keywords when you are search for parts in Proteus:

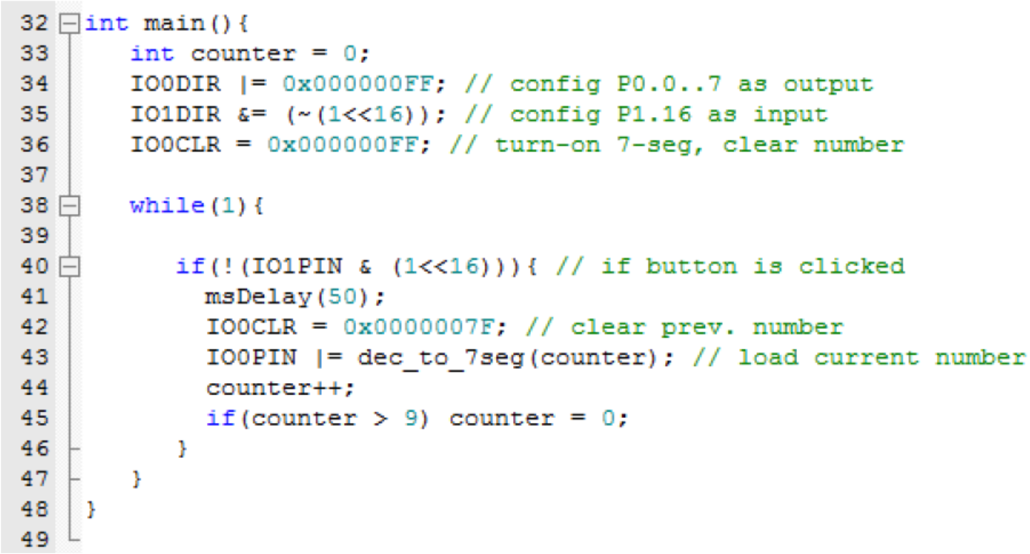
Part	Keyword
Microcontroller	LPC2138
7 segments	7seg
Button	Push

Write and simulate a program that increments a 7-segment by one with every click (range 0 to 9).

○ Keil

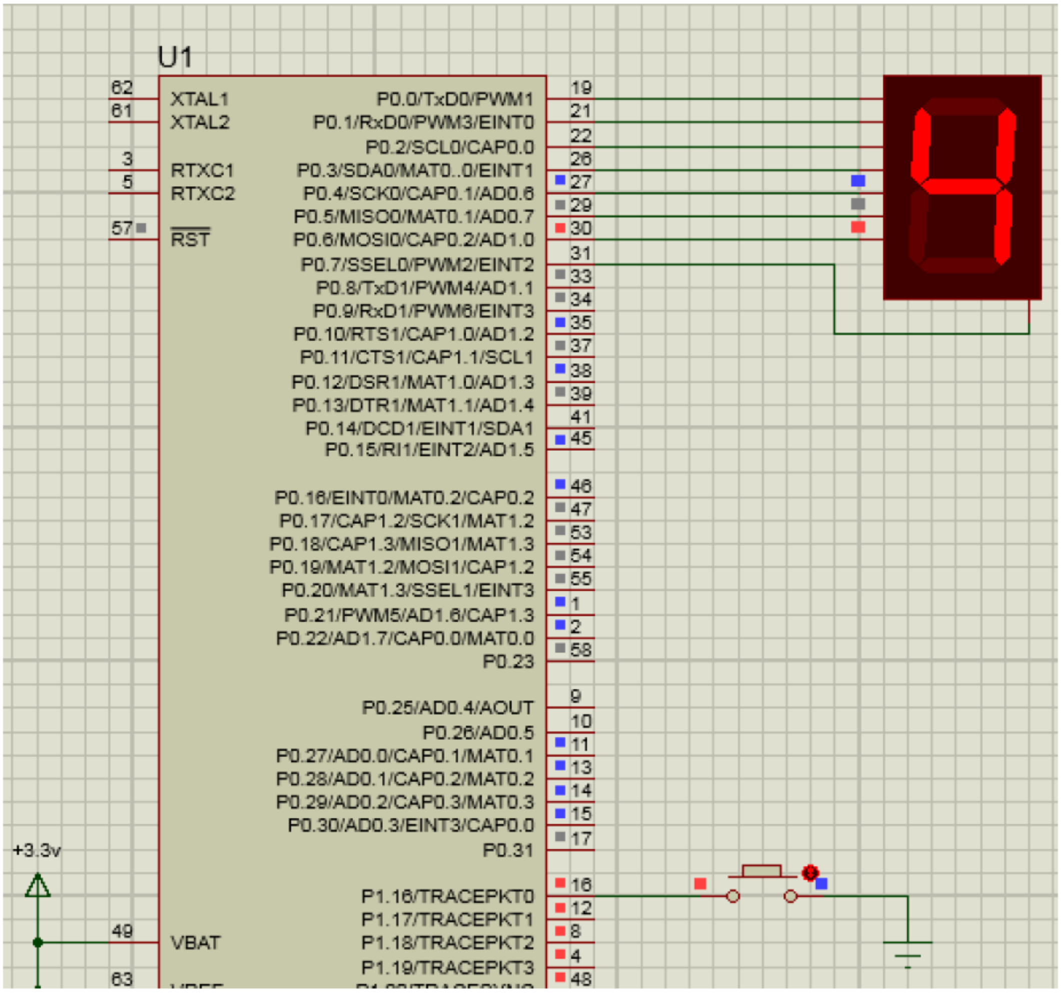


```
#include<lpc213x.h>

void msDelay(int d){ // ms delay at 12 MHZ
        int i,j;
        long c = 0;
        d = d*2;
        for(i=0;i<d;i++){
           for(j=0;j<1000;j++){
              c++;
           }
        }
}

int dec_to_7seg(int number){
    switch(number){
        case 0 : return 0x3F;
        case 1 : return 0x06;
        case 2 : return 0x5B;
        case 3 : return 0x4F;
        case 4 : return 0x66;
        case 5 : return 0x6D;
        case 6 : return 0x7D;
        case 7 : return 0x07;
        case 8 : return 0x7F;
        case 9 : return 0x6F;
        default: return 0x00;
    }
}
```



```
int main(){
    int counter = 0;
    IO0DIR |= 0x000000FF; // config P0.0..7 as output
    IO1DIR &= (~(1<<16)); // config P1.16 as input
    IO0CLR = 0x000000FF; // turn-on 7-seg, clear number

    while(1){

        if(!(IO1PIN & (1<<16))){ // if button is clicked
          msDelay(50);
          IO0CLR = 0x0000007F; // clear prev. number
          IO0PIN |= dec_to_7seg(counter); // load current number
          counter++;
          if(counter > 9) counter = 0;
        }
    }
}
```

- **Proteus**



7-Segment Multiplexing

The simplest way to drive a display is via a display driver. These are available for up to 4 displays. Alternatively displays can be driven by a microcontroller and if more than one display is required, the method of driving them is called "multiplexing".

If a single display is to be driven from a microcontroller, 7 lines will be needed plus one for the decimal point. For each additional display, only one extra line is needed.

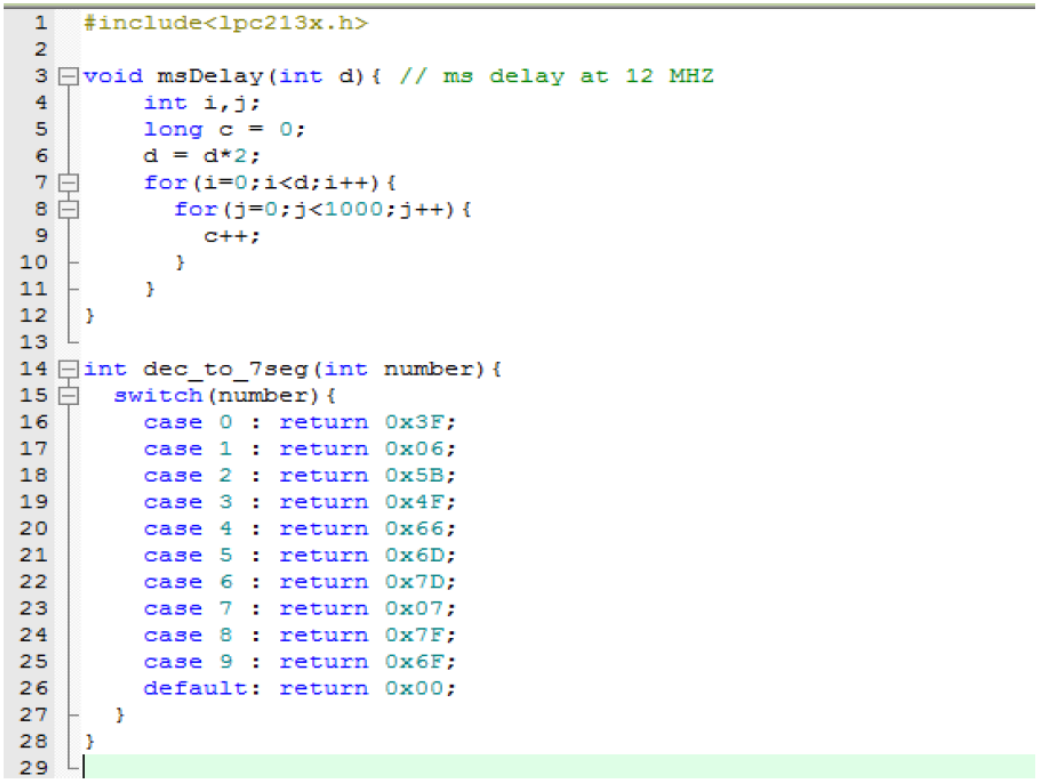
To produce a 4, 5 or 6 digit display, all the 7- segment displays are connected in parallel. The common line (the common-cathode line) is taken out separately and this line is taken low for a short period of time to turn on the display. Each display is turned on at a rate above 100 times per second, and it will appear that all the displays are turned on at the same time. As each display is turned on, the appropriate information must be delivered to it so that it will give the correct reading. Up to 6 displays can be accessed like this without the brightness of each display being affected.

➢ Lab Work 3

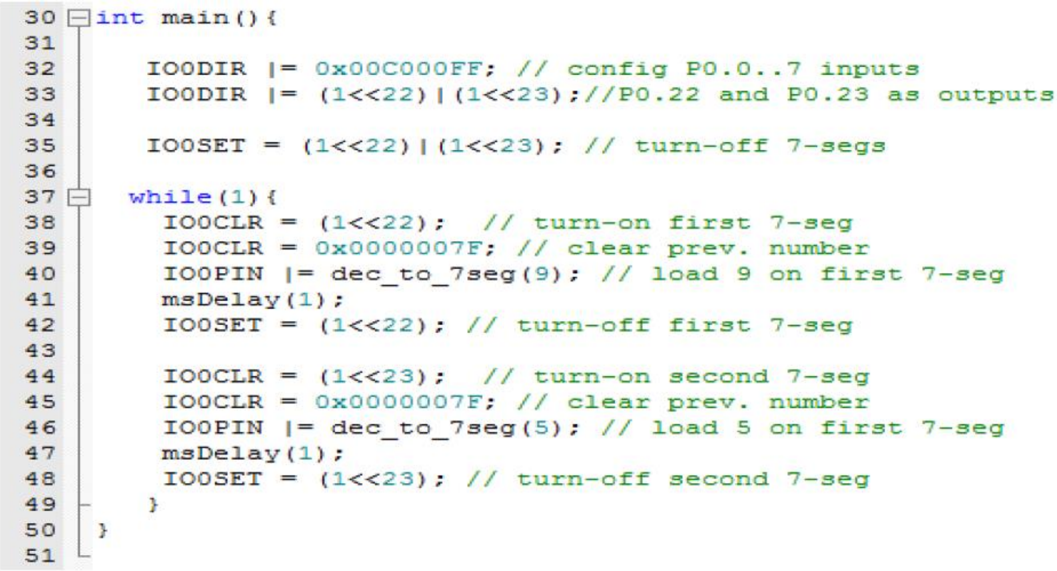
You are going to use these keywords when you search for parts in Proteus:

Part	Keyword
Microcontroller	LPC2138
7-segments	7seg

○ Keil

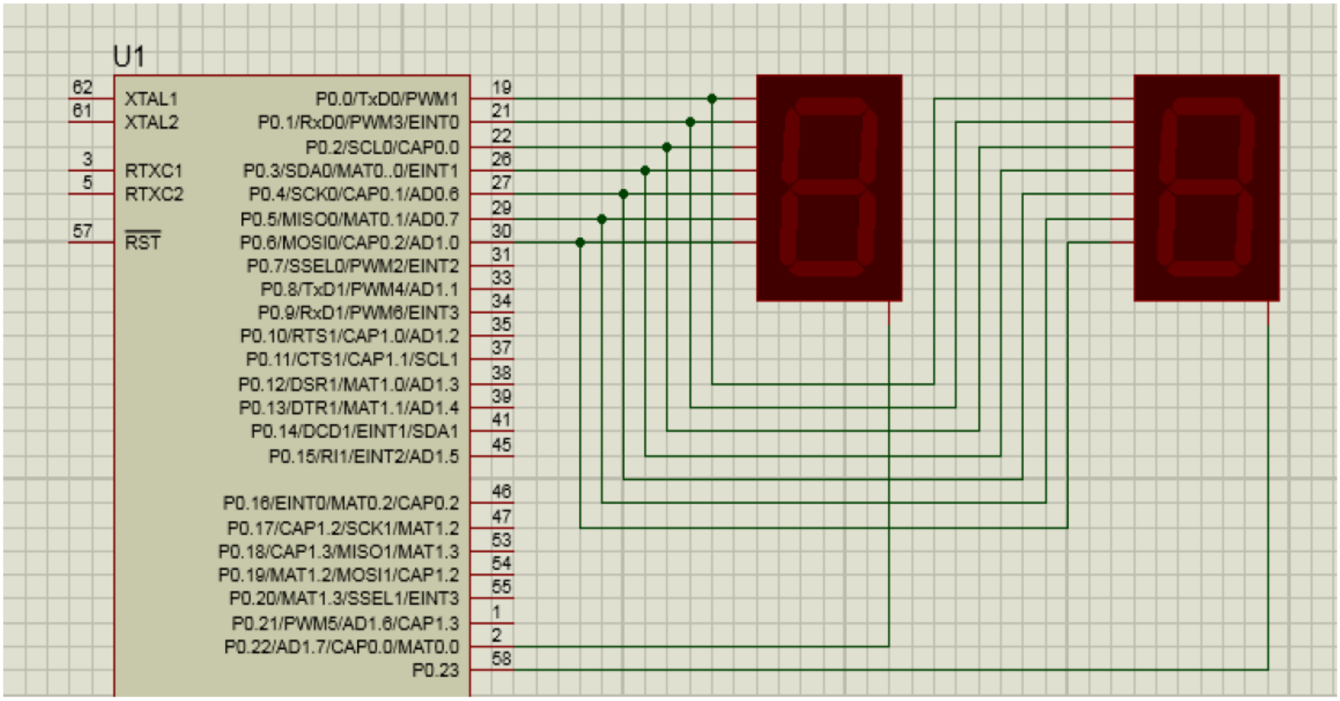


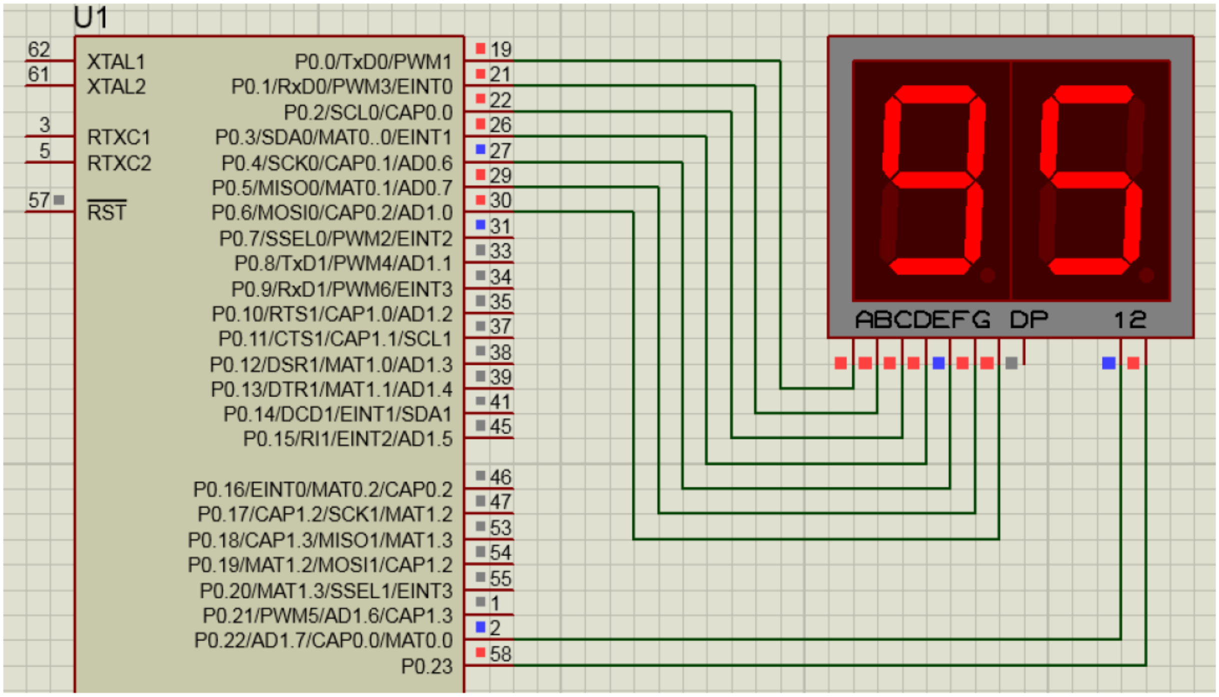
```
#include<lpc213x.h>

void msDelay(int d){ // ms delay at 12 MHZ
    int i,j;
    long c = 0;
    d = d*2;
    for(i=0;i<d;i++){
      for(j=0;j<1000;j++){
        c++;
      }
    }
}

int dec_to_7seg(int number){
  switch(number){
    case 0 : return 0x3F;
    case 1 : return 0x06;
    case 2 : return 0x5B;
    case 3 : return 0x4F;
    case 4 : return 0x66;
    case 5 : return 0x6D;
    case 6 : return 0x7D;
    case 7 : return 0x07;
    case 8 : return 0x7F;
    case 9 : return 0x6F;
    default: return 0x00;
  }
}
```



```
int main(){

   IO0DIR |= 0x00C000FF; // config P0.0..7 inputs
   IO0DIR |= (1<<22)|(1<<23);//P0.22 and P0.23 as outputs

   IO0SET = (1<<22)|(1<<23); // turn-off 7-segs

  while(1){
    IO0CLR = (1<<22);  // turn-on first 7-seg
    IO0CLR = 0x0000007F; // clear prev. number
    IO0PIN |= dec_to_7seg(9); // load 9 on first 7-seg
    msDelay(1);
    IO0SET = (1<<22); // turn-off first 7-seg

    IO0CLR = (1<<23);  // turn-on second 7-seg
    IO0CLR = 0x0000007F; // clear prev. number
    IO0PIN |= dec_to_7seg(5); // load 5 on first 7-seg
    msDelay(1);
    IO0SET = (1<<23); // turn-off second 7-seg
   }
}
```

- ○ **Protues**





Post Lab:

1. Include all your work during the lab.

2. Implement a system that toggles two LEDs when controlling that with clicking on 2 push buttons.

3. Implement a system that counts from 00 to 99 and increments every one click on a push button.

4. Simulate your work on Proteus and attach it.