Islamic University of Gaza
Faculty of Engineering
Computer Engineering Department
Embedded Systems Lab
Eng. Mohamed W. Awwad
Eng. Ibtesam M. Dheir

Lab 2

Basic I/O

Feb. 2016

Objective

1. To know how to use keys, LEDs and 7-segments.
2. Introducing 7-segment multiplexing.

Introduction

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:

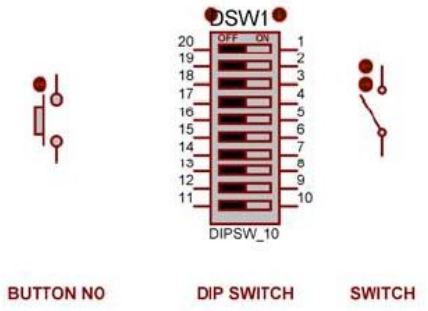
```
IO0DIR |= (1<<19); //Config P0.19 as output, other pins not affected
IO0SET = (1<<19); // Make output High for P0.19
```

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

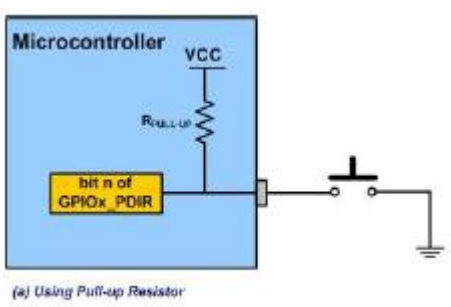
Switches and buttons are the basic input devices used to supply certain voltage level for microcontrollers. Button/switch is a mechanical component which connects or disconnects two points A and B over its contacts. By function, button/switch contacts can be normally open or normally closed



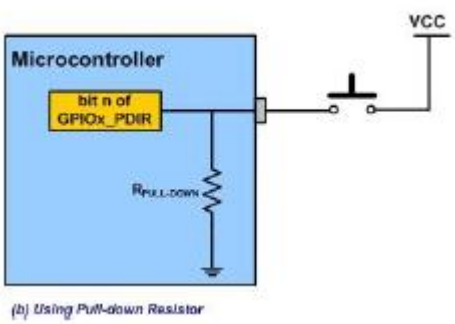
Buttons can be connected to the microcontroller in one of two ways:

- **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 1

You are going to use this keywords when you search for parts in proteus:

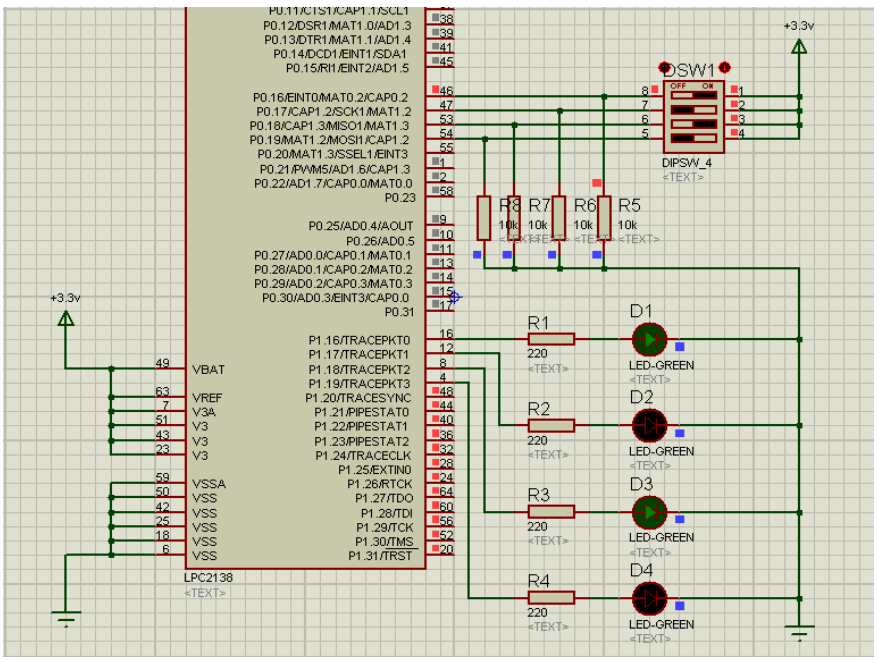
Part	Keyword
Microcontroller	LPC2138
Resistor	res
Switch	dipsw
LEDs	led-
Button	Push

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.

Note:

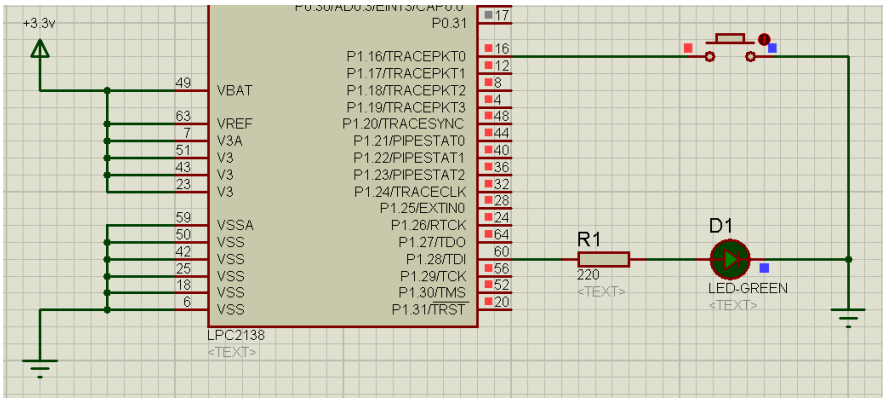
The needed time for the user to press the push button is approximately 200 ms, the speed of the LPC's work is very high, so 'while true' loop will be executed a lot of times through this period (200 ms), which means that the button has been pressed many times, but in fact the user did it just once!.

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*/
    }
  }
```

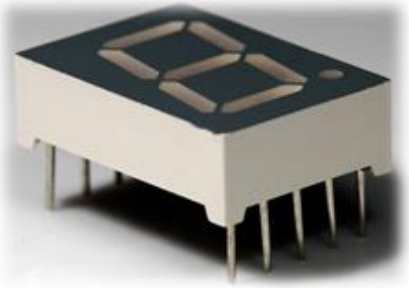
- **Protues**

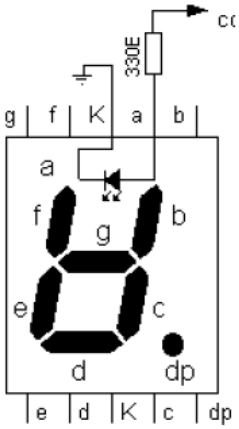


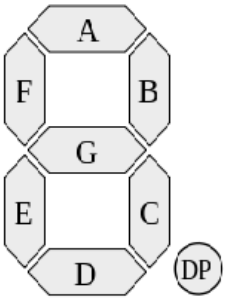
7-Segment Display

Seven-segment displays are commonly used in electronics as a method of displaying decimal numeric feedback on the internal operations of devices. A seven segment display, as its name indicates, is composed of seven elements. Individually on or off, they can be combined to produce simplified representations of the Hindu-Arabic numerals. The most commonly used is so called 7- segment display.

It is composed of 8 LEDs- 7 segments are arranged as a rectangle for symbol displaying and there is an additional segment for decimal point displaying. The segments of a 7 segment display are referred to by the letters A to G, as follows:

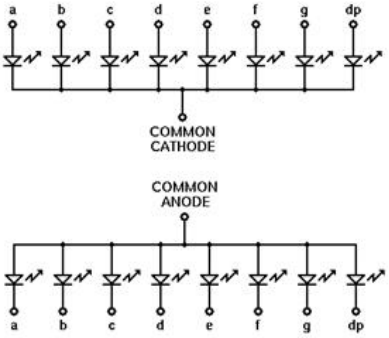


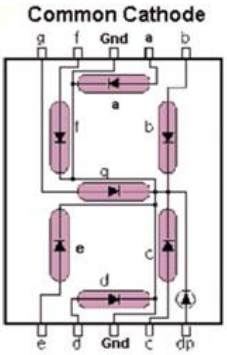


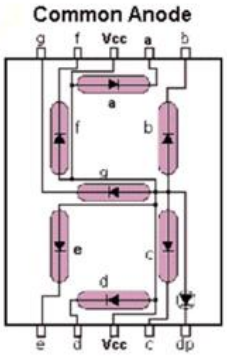


Seven segment display can be divided into 2 types of connection:

- One is called common anode of which all the anodes of the LEDs are connected together, leaving the cathodes open for connection.
- The other one is called common cathode of which all the cathodes of the LEDs are connected together, leaving the anodes open for connection.







Lab Work 2

You are going to use this keywords when you are search for parts in proteus:

Part	Keyword
Microcontroller	LPC2138
7 segments	7seg
Button	Push

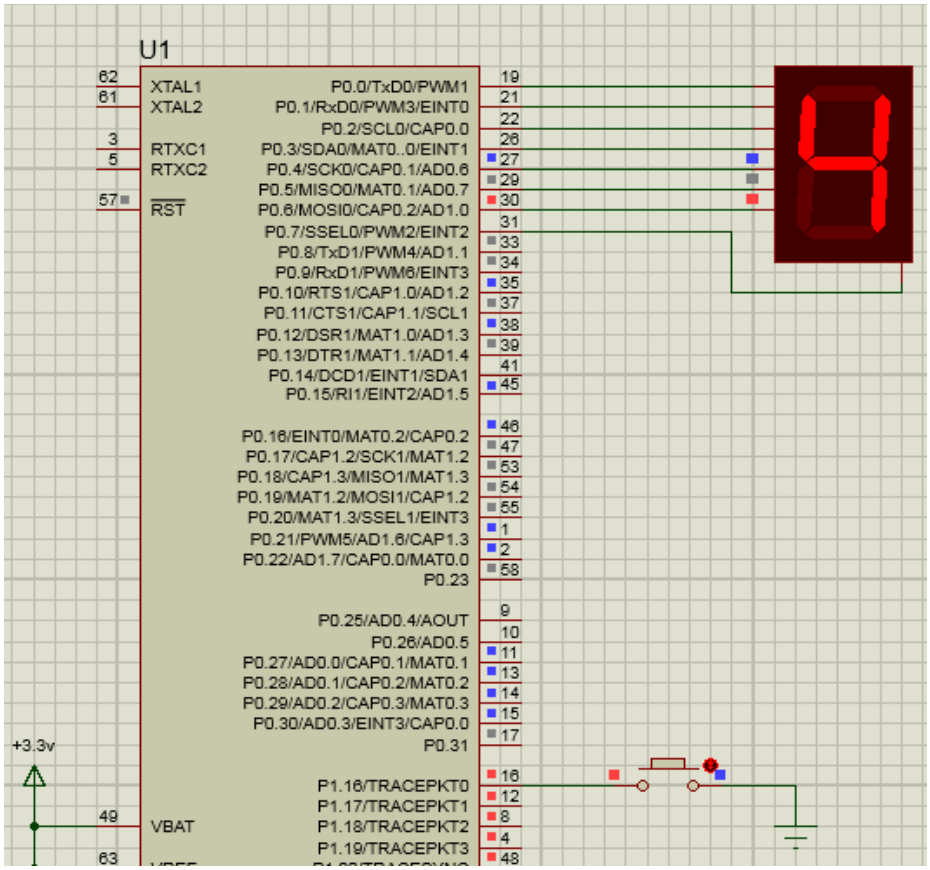
- Write and simulate a program that increments a 7-segment by one with every click (range 0 to 9).

  - **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 this keywords when you search for parts in proteus:

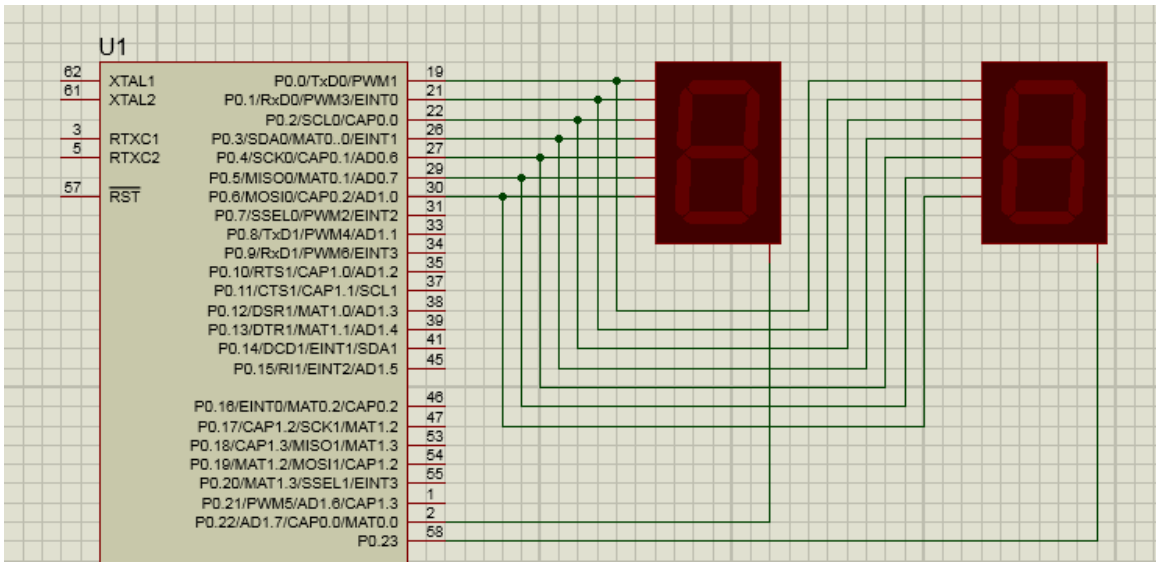
Part	Keyword
Microcontroller	LPC2138
7-segments	7seg

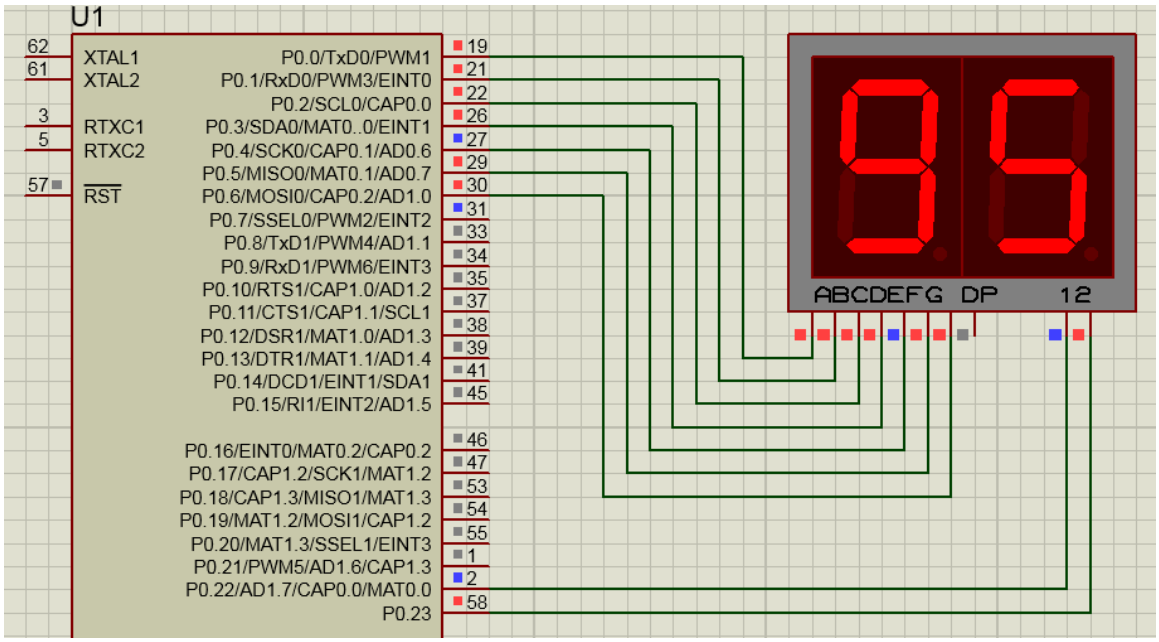
- Write and simulate a program that displays 95 on two 7-segments.
  - **Keil**

```
#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**


U1
62 XTAL1
61 XTAL2
3 RTXC1
5 RTXC2
57 RST
P0.0/TxD0/PWM1
P0.1/RxD0/PWM3/EINT0
P0.2/SCL0/CAP0.0
P0.3/SDA0/MAT0..0/EINT1
P0.4/SCK0/CAP0.1/AD0.6
P0.5/MISO0/MAT0.1/AD0.7
P0.6/MOSI0/CAP0.2/AD1.0
P0.7/SSEL0/PWM2/EINT2
P0.8/TxD1/PWM4/AD1.1
P0.9/RxD1/PWM6/EINT3
P0.10/RTS1/CAP1.0/AD1.2
P0.11/CTS1/CAP1.1/SCL1
P0.12/DSR1/MAT1.0/AD1.3
P0.13/DTR1/MAT1.1/AD1.4
P0.14/DCD1/EINT1/SDA1
P0.15/RI1/EINT2/AD1.5
P0.16/EINT0/MAT0.2/CAP0.2
P0.17/CAP1.2/SCK1/MAT1.2
P0.18/CAP1.3/MISO1/MAT1.3
P0.19/MAT1.2/MOSI1/CAP1.2
P0.20/MAT1.3/SSEL1/EINT3
P0.21/PWM5/AD1.6/CAP1.3
P0.22/AD1.7/CAP0.0/MAT0.0
P0.23
19 21 22 26 27 29 30 31 33 34 35 37 38 39 41 45 46 47 53 54 55 1 2 58


U1
62 XTAL1
61 XTAL2
3 RTXC1
5 RTXC2
57 RST
P0.0/TxD0/PWM1
P0.1/RxD0/PWM3/EINT0
P0.2/SCL0/CAP0.0
P0.3/SDA0/MAT0..0/EINT1
P0.4/SCK0/CAP0.1/AD0.6
P0.5/MISO0/MAT0.1/AD0.7
P0.6/MOSI0/CAP0.2/AD1.0
P0.7/SSEL0/PWM2/EINT2
P0.8/TxD1/PWM4/AD1.1
P0.9/RxD1/PWM6/EINT3
P0.10/RTS1/CAP1.0/AD1.2
P0.11/CTS1/CAP1.1/SCL1
P0.12/DSR1/MAT1.0/AD1.3
P0.13/DTR1/MAT1.1/AD1.4
P0.14/DCD1/EINT1/SDA1
P0.15/RI1/EINT2/AD1.5
P0.16/EINT0/MAT0.2/CAP0.2
P0.17/CAP1.2/SCK1/MAT1.2
P0.18/CAP1.3/MISO1/MAT1.3
P0.19/MAT1.2/MOSI1/CAP1.2
P0.20/MAT1.3/SSEL1/EINT3
P0.21/PWM5/AD1.6/CAP1.3
P0.22/AD1.7/CAP0.0/MAT0.0
P0.23
19 21 22 26 27 29 30 31 33 34 35 37 38 39 41 45 46 47 53 54 55 1 2 58
ABCDEFG DP
12

Homework

- **Include all your work during the lab.**
- **Implement a system that toggles 2 LEDs when controlling that with clicking on 2 push buttons.**
- **Implement a system that counts from 00 to 99 and increments every one click on a push button.**
- **Simulate your work on Proteus and attach it.**

Good Luck