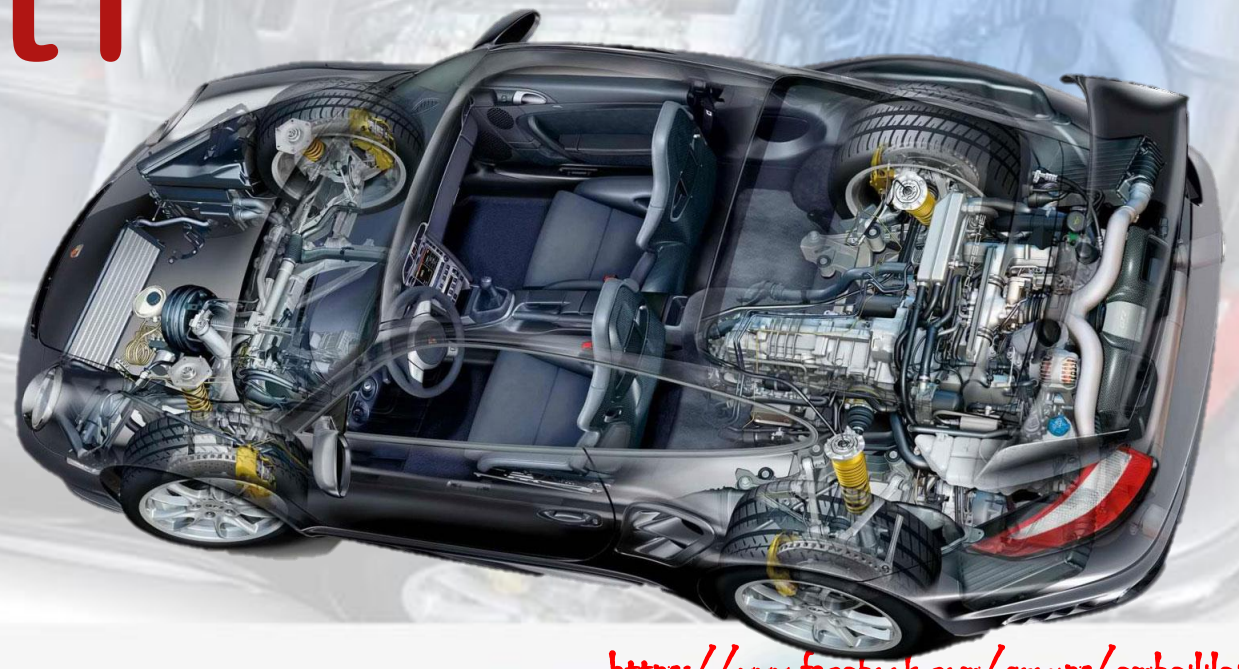
Unit7 Lesson 2
Section Part1



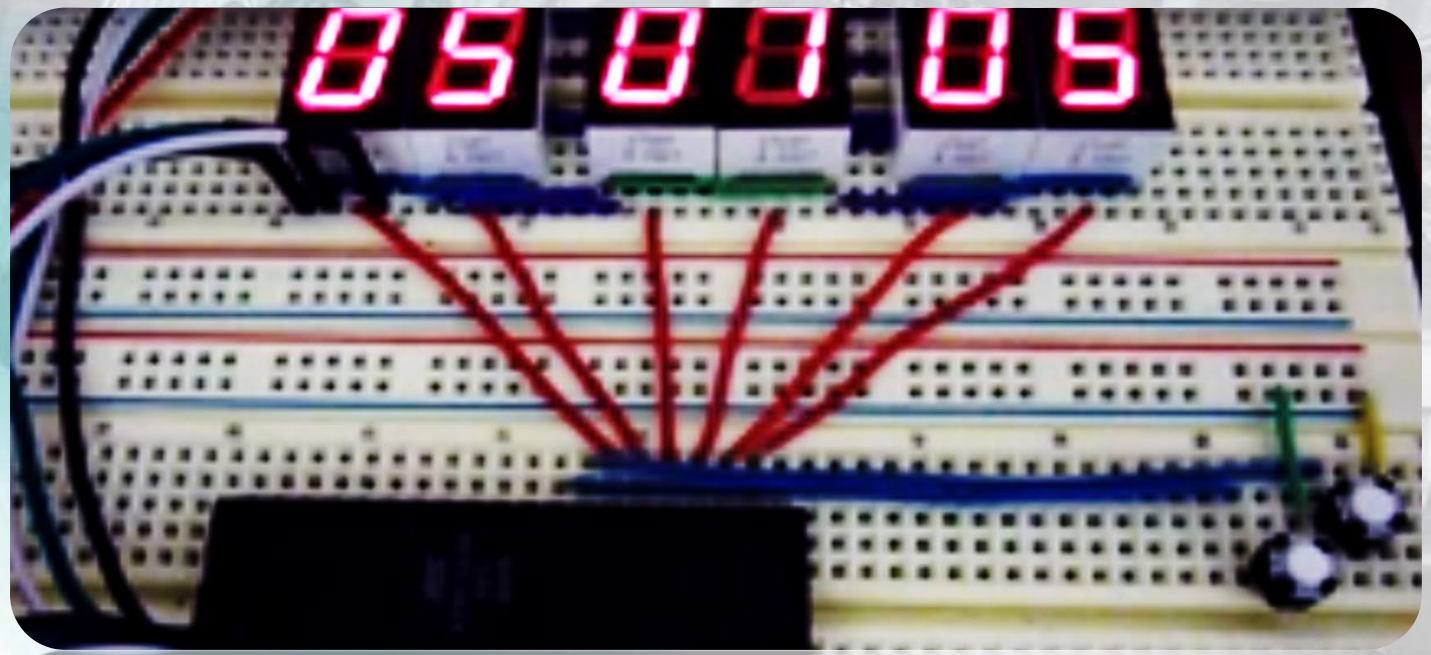
Index (2)

- Previous Projects Solution
- LCD
  - LCD Driver
- KeyPAD
  - KeyPad Driver

https://www.facebook.com/groups/embedded.system.KS/

Previous Projects Solution

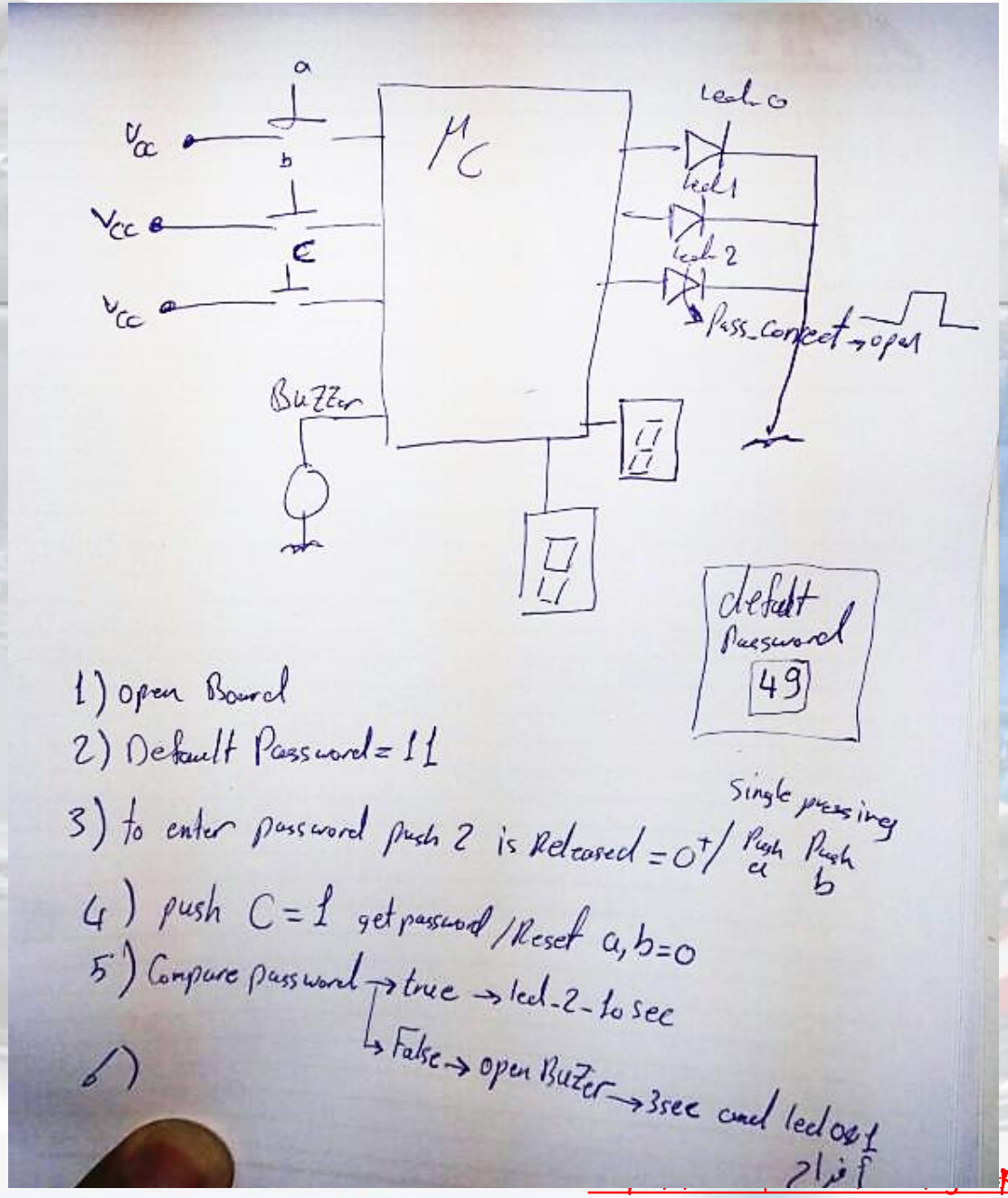
DIO_Project2
Seven Segment Digital Clock without Timer Module



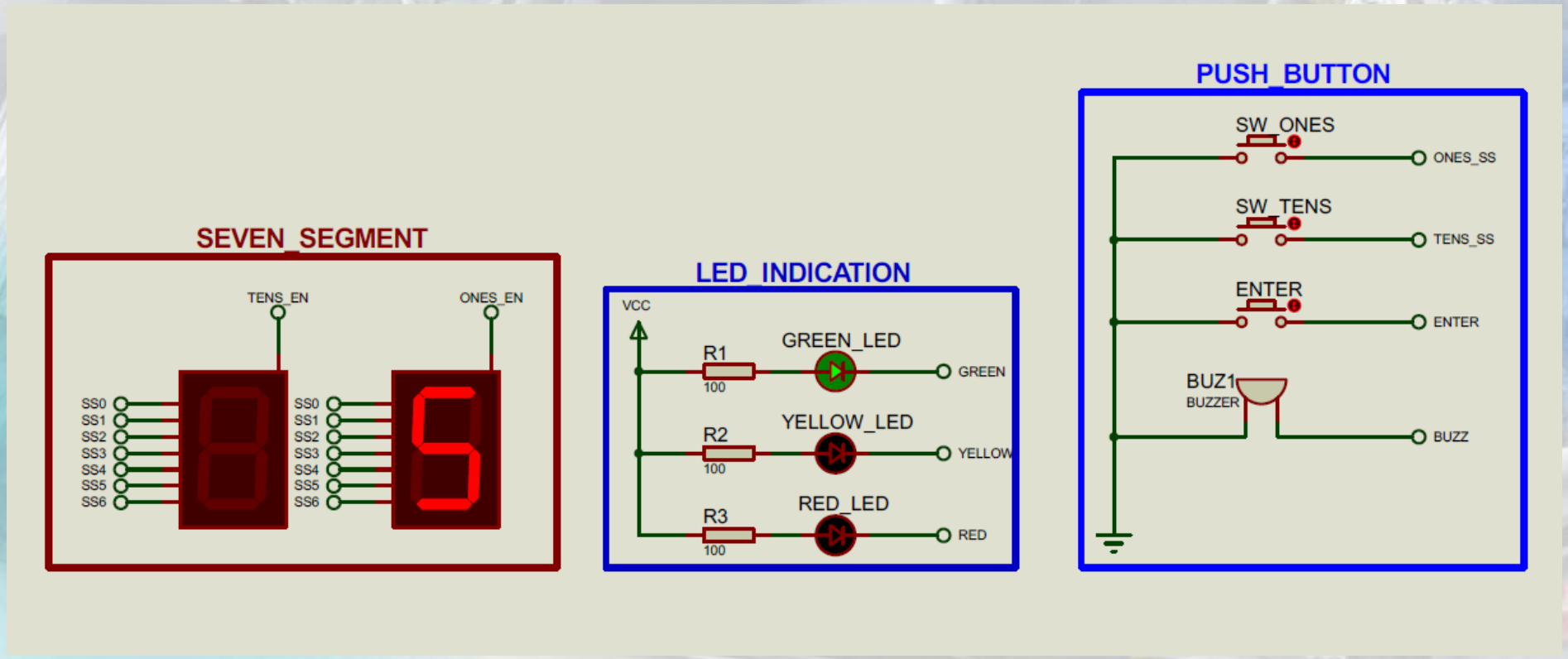
https://github.com/keroles/Embedded_System/tree/master/Embedded_System/Projects_solutiones/new_diploma/clock_6_digits_without_Timer_Module

GPIO Project 3 Assignment Microcontroller session

Write code to make the following functionality:



https://github.com/keroles/Embedded_System/tree/master/Embedded_System/Projects_solutiones/new_diploma/security%20system



New Content

LCD (Liquid Crystal Display)

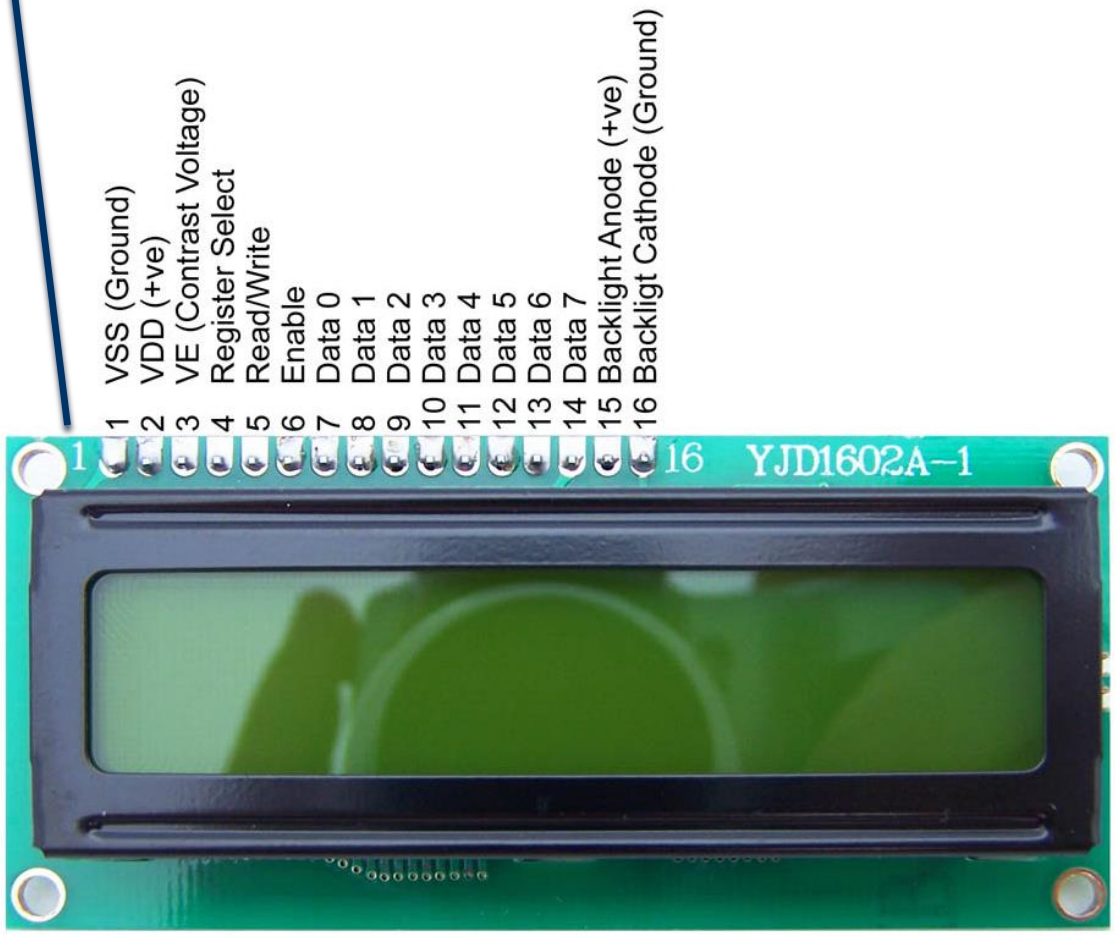
Interfacing an 16×2 character LCD (Liquid Crystal Display)

- LCD stands for liquid crystal display. Now earlier we used to use 7 segment displays for display purposes, but now LCD's are preferred. The main reason is we need less number of data bus lines for interfacing LCD's as compared **to 7 segment displays**.
- Other reason is we can print various characters on the screen.
- Now the basic characters are already saved inside **CGROM(Character Generator ROM).** **So you need to send only the ASCII values in order to display the character on screen**
- 16×2 LCD means It has 2 rows and 16 columns

LCD interfacing

6. Interface pin description

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

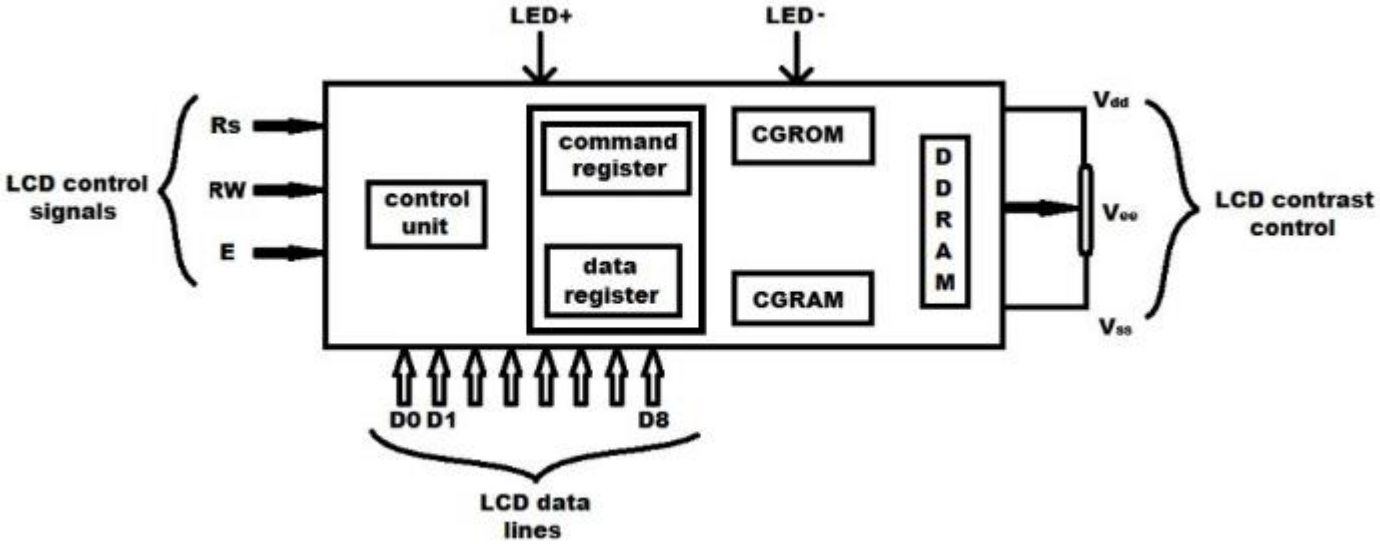


LCD interfacing

- a **VDD** pin for 5 volts
- a **VSS** pin for ground.
- **V0** pin for the adjustment of the LCD contrast.
- Some LCDs even have an LED backlight and are generally **the last two pins**.
- the LCD has a row of 8 pins to serve as its port. **D0, D1, D2, D3, D4, D5, D6** and **D7.** These pins are generally used to pass information into the LCD, but it can also be set to pass information back to the microcontroller
- **read/write** is microcontroller centric: the LCD "read" mode is the process of passing information from the LCD to the microcontroller (microcontroller port set as input or reading or listening).; the LCD "write" mode is passing information from the microcontroller to the LCD (microcontroller set to output or writing).
- The pin on the LCD that is responsible for whether the information sent is a character or a control, is the **RS pin (Register Select)**
- the pin that helps the LCD accept information is called the **EN pin (Enable**

Block Diagram

- There are three memories in LCD to manipulate display characters:
  - **CGROM** – character generating ROM which is responsible for stored **standard character** patter n.
  - **CGRAM** – character generating RAM which holds **custom character pattern** space (total 8 in 2×16 module).
  - **DDRAM** – data display RAM which stores ASCII codes.

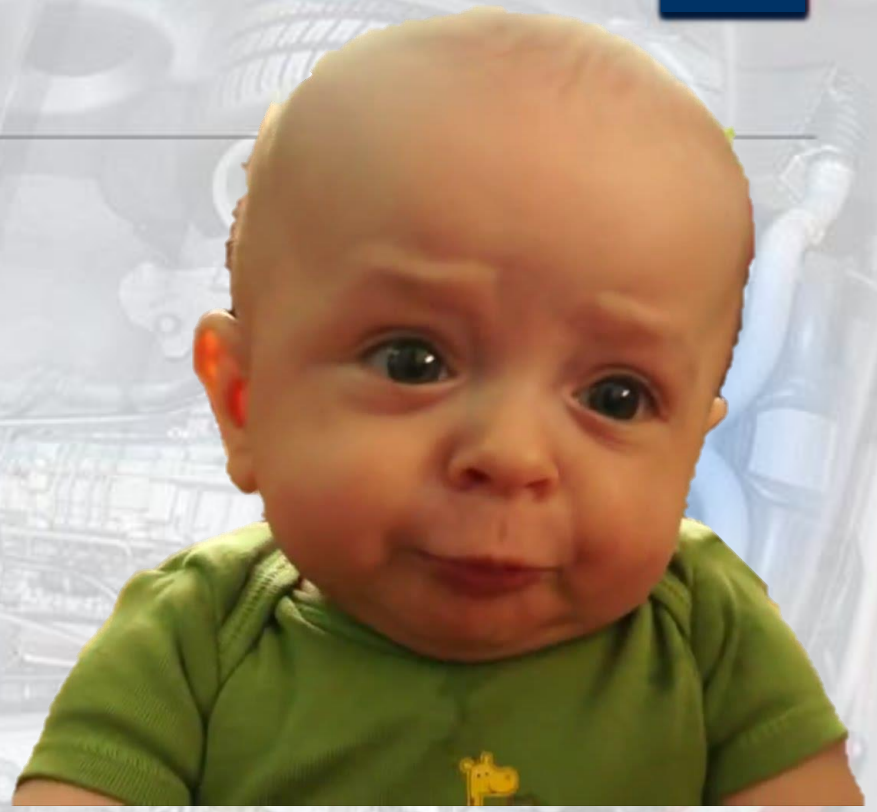


LCD Functionality

(1) to make sure the LCD is not busy

(2) Control the LCD's cursor, or display function

(3) Write a character to the LCD for it to display



Give me all your attention

Please

LCD Functionality

RW ON

RS OFF

Read Command

Microcontroller

If Data 7 is 1 (LCD Busy)
If zero (LCD not Busy)

(1) to make sure the LCD is not busy

a. set the port to receive data on the microcontroller (Data direction as input).

b. put the LCD in read mode (RW on).
c. put the LCD in command mode (RS off).

And the port now magically contains the data from the LCD (**D7 pin** will be ***ON*** if the LCD is ***busy*** and ***OFF*** if the LCD ***is not busy***).

(2) Control the LCD's cursor, or display function

(3) Write a character to the LCD for it to display

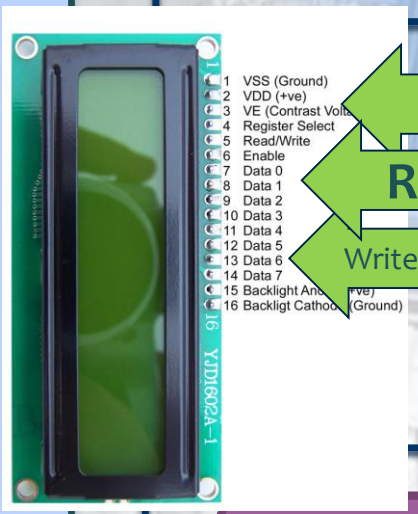
LCD Functionality

(1) to make sure the LCD is not busy

(2) Control the LCD's cursor, or display function (Send a command to the LCD)

- set the port direction as **output** so you can send information to the LCD.
- turn RW off so you can **write**.
- turn RS off for **command mode**.
- Write the **command** on D0… D7
- turn on the enable and then turn it off.(delay ~500ns)

The LCD will magically perform the command.



RW off

RS OFF

Write Command

Microcontroller

(3) Write a character to the LCD for it to display

LCD Functionality

(1) to make sure the LCD is not busy

(2) Control the LCD's cursor, or display function

- (3) Write a character to the LCD for it to display

  Send a character to the LCD: This is the same as sending a command except the **RS is on** and the **port will equal the character corresponding to the ASCII code.**

- **set the port direction as output so you can send information to the LCD.**
- **turn RW off so you can write.**
- ***turn RS ON for Data mode.***
- **Write the command on D0… D7**
- **turn on the enable and then turn it off.(delay ~500ns)**

RW off

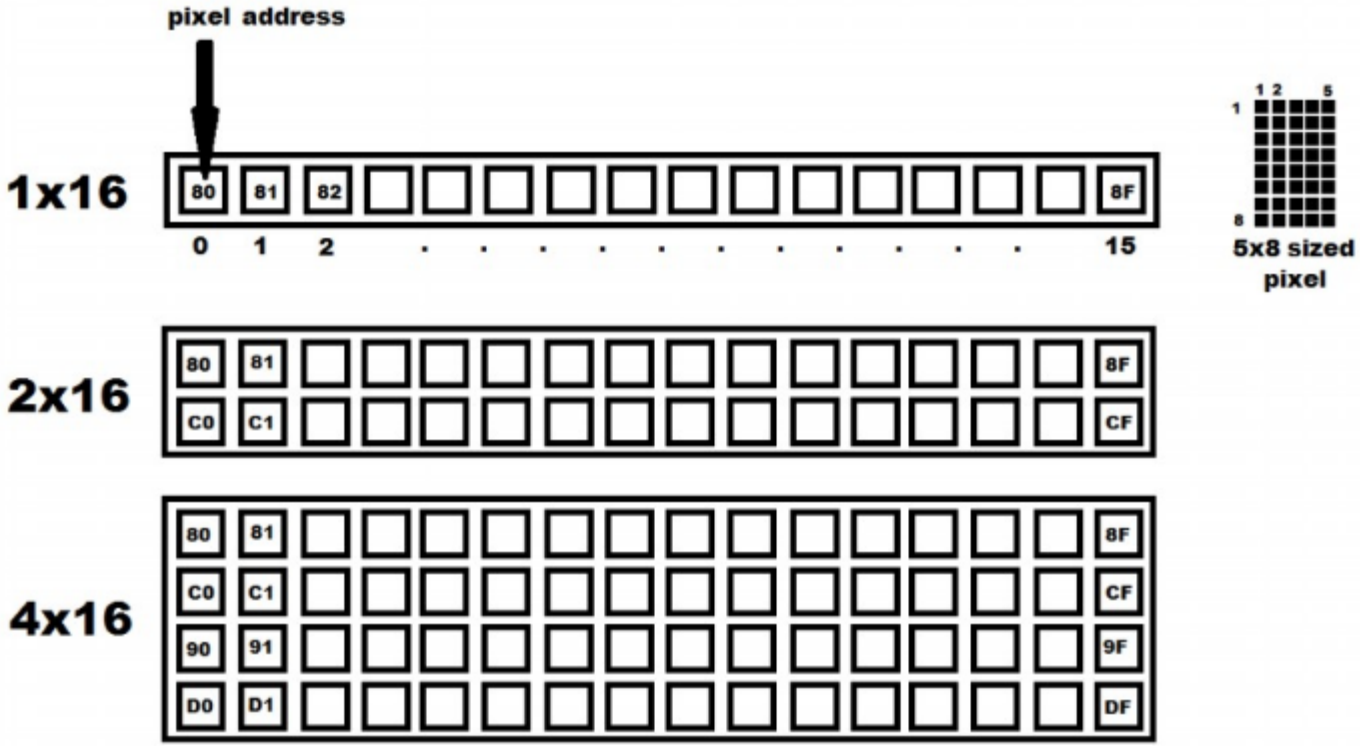
RS ON

Write **character**

Microcontroller

16×2 lcd commands

COMMAND	FUNCTION
0x80	Force cursor to beginning of the first line
0xC0	Force cursor to beginning of second line
0x38	Use two lines and 5x7 matrix
0x83	Cursor line 1 position 3
0x3C	Activate second line
0xC3	Jump to second line position 3
0xC1	Jump to second line position1

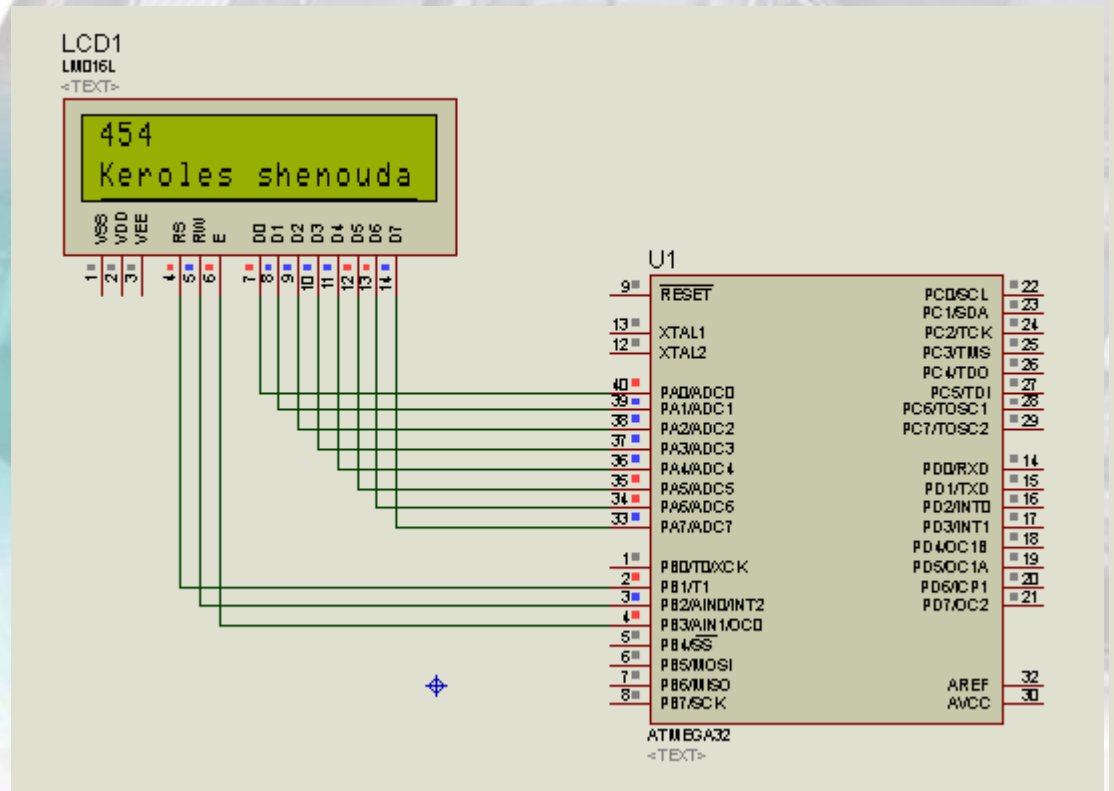


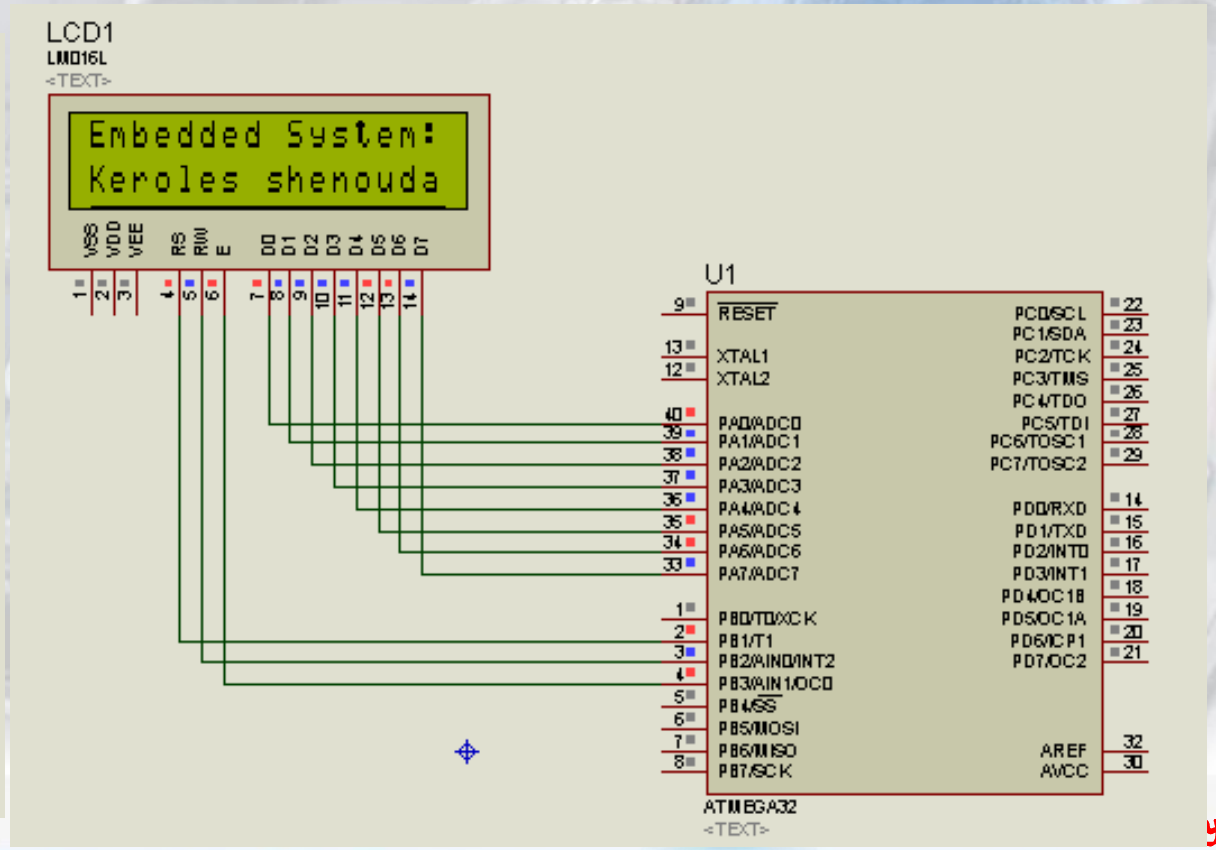
16×2 lcd commands

Instruction	Hex
Function Set: 8-bit, 1 Line, 5x7 Dots	0x30
Function Set: 8-bit, 2 Line, 5x7 Dots	0x38
Function Set: 4-bit, 1 Line, 5x7 Dots	0x20
Function Set: 4-bit, 2 Line, 5x7 Dots	0x28
Entry Mode	0x06
Display off Cursor off (clearing display without clearing DDRAM content)	0x08
Display on Cursor on	0x0E
Display on Cursor off	0x0C
Display on Cursor blinking	0x0F
Shift entire display left	0x18
Shift entire display right	0x1C
Move cursor left by one character	0x10
Move cursor right by one character	0x14
Clear Display (also clear DDRAM content)	0x01

LAB: write LCD Driver [8bit] Data mode

Test your driver by print your name and an integer number then double number





LCD Driver

focus
with me

https://github.com/keroles/Embedded_System/tree/master/Atmega32_Drivers/ATMEGA32_Drivers/lcd

Solution:LCD

LCD.h

lcd.h

```
/*
 * lcd.h
 *
 *  Created on: Jan 19, 2017
 *      Author: Keroles shenouda
 */

#ifndef DRIVERS_LCD_LCD_H_
#define DRIVERS_LCD_LCD_H_

#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include <stdlib.h>
#include <stdio.h>

#define LCD_port PORTA
#define DataDir_LCD_port DDRA
#define DATA_shift 0    //0:4bit data mode | 4:4bit data mode
#define LCD_control PORTB
#define DataDir_LCD_control DDRB
#define EN_switch 3
#define ReadWrite 2
#define RS_switch 1

//#define   EIGHT_BIT_MODE
#define FOUR_BIT_MODE


#define LCD_REGISTER_SELECT_PIN                 (0)
#define LCD_READ_WRITE_PIN                      (1)
#define LCD_ENABLE_PIN                          (2)
#define LCD_REGISTER_SELECT_ENABLE              (1)
#define LCD_REGISTER_SELECT_DISABLE             (0)
#define READ_FROM_LCD                           (1)
#define WRITE_TO_LCD                            (0)
#define LCD_ENABLE                              (1)
#define LCD_DISABLE                             (0)
#define LCD_FIRST_LINE                          (0)
```

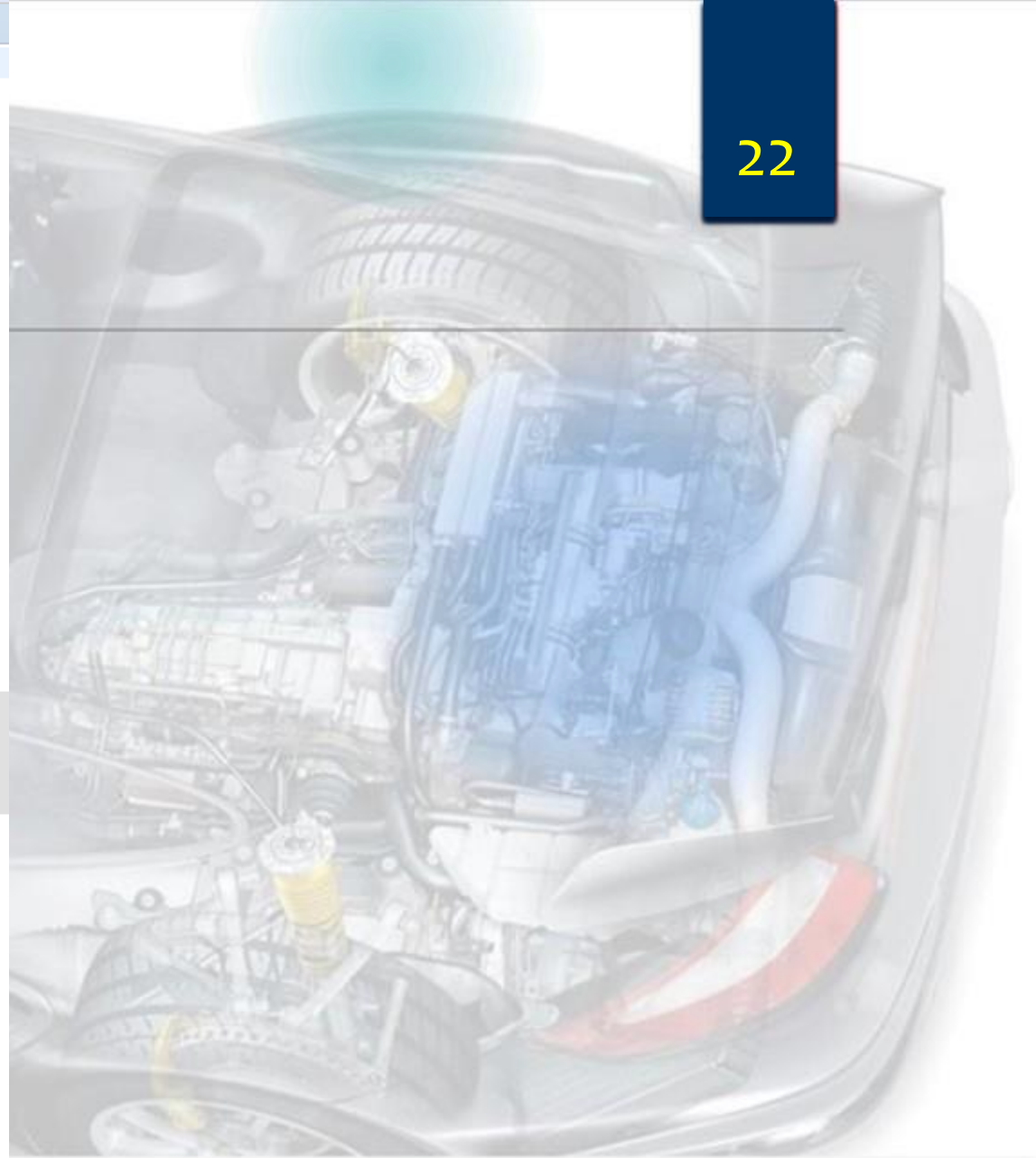
Configure
4Mode
Or 8Mode

```
#define LCD_SECOND_LINE                         (1)
#define LCD_FUNCTION_8BIT_2LINES                (0x38)
#define LCD_FUNCTION_4BIT_2LINES                (0x28)
#define LCD_MOVE_DISP_RIGHT                     (0x1C)
#define LCD_MOVE_DISP_LEFT                      (0x18)
#define LCD_MOVE_CURSOR_RIGHT                   (0x14)
#define LCD_MOVE_CURSOR_LEFT                    (0x10)
#define LCD_DISP_OFF                            (0x08)
#define LCD_DISP_ON_CURSOR                      (0x0E)
#define LCD_DISP_ON_CURSOR_BLINK                (0x0F)
#define LCD_DISP_ON_BLINK                       (0x0D)
#define LCD_DISP_ON                             (0x0C)
#define LCD_ENTRY_DEC                           (0x04)
#define LCD_ENTRY_DEC_SHIFT                     (0x05)
#define LCD_ENTRY_INC_                          (0x06)
#define LCD_ENTRY_INC_SHIFT                     (0x07)
#define LCD_BEGIN_AT_FIRST_RAW                  (0x80)
#define LCD_BEGIN_AT_SECOND_RAW                 (0xC0)
#define LCD_CLEAR_SCREEN                        (0x01)
#define LCD_ENTRY_MODE                          (0x06)


void LCD_check_lcd_isbusy(void);
void LCD_lcd_kick (void);
void LCD_Send_A_Command(unsigned char command);
void LCD_Send_A_Character(unsigned char character);
void LCD_Send_A_String(char *string);
void LCD_lcd_init(void);
void LCD_clear_screen ();
void LCD_GotoXY(unsigned char line, unsigned char position );
void LCD_display_number (int Number );
void LCD_display_real_number (double  Number );

#endif /* DRIVERS_LCD_LCD_H_ */
```

LCD.C

lcd.c

```c
/*
 * lcd.c
 *
 *  Created on: Jan 19, 2017
 *      Author: Keroles shenouda
 */

#include "lcd.h"

void LCD_lcd_init(void)
{

    _delay_ms(20);
    DataDir_LCD_control |= (1<<EN_switch | 1<<ReadWrite | 1<<RS_switch) ;
    LCD_control &= ~(1<<EN_switch | 1<<ReadWrite | 1<<RS_switch) ;

    DataDir_LCD_port = 0xff ;
    _delay_ms(15);

    LCD_clear_screen ();

#ifdef EIGHT_BIT_MODE
    LCD_Send_A_Command(LCD_FUNCTION_8BIT_2LINES);

#endif
#ifdef FOUR_BIT_MODE
    LCD_Send_A_Command(0x02);

    LCD_Send_A_Command(LCD_FUNCTION_4BIT_2LINES);

#endif

    LCD_Send_A_Command(LCD_ENTRY_MODE);
    LCD_Send_A_Command(LCD_BEGIN_AT_FIRST_RAW);
    LCD_Send_A_Command(LCD_DISP_ON_CURSOR_BLINK);

}

```



LCD.C


```c
void LCD_GotoXY(unsigned char line, unsigned char position )
{

    if(line==0)
    {
        if (position < 16 && position >=0)
            Send_A_Command(0x80+position);
    }
    else if (line==1)
    {
        if (position < 16 && position >=0)
            Send_A_Command(0xc0 + position);
    }
}




void check_lcd_isbusy()
{
    DataDir_LCD_port &= ~(0xff<<DATA_shift) ;
    LCD_control |= (1<<ReadWrite); // read
    LCD_control &= ~ (1<<RS_switch);

    //  while (LCD_port >= 0x80)
    //  {
    lcd_kick ();
    //  }

    //  _delay_ms (500) ;
    DataDir_LCD_port = 0xFF; //0xFF means 0b11111111
    LCD_control &= ~ (1<<ReadWrite); //write
}
```

```c
void LCD_clear_screen ()
{
    LCD_Send_A_Command(LCD_CLEAR_SCREEN); // clear the screen

}

void LCD_GotoXY(unsigned char line, unsigned char position )
{

    if(line==0)
    {
        if (position < 16 && position >=0)
            LCD_Send_A_Command(0x80+position);
    }
    else if (line==1)
    {
        if (position < 16 && position >=0)
            LCD_Send_A_Command(0xc0 + position);
    }
}
```

LCD.C

```
64  void LCD_check_lcd_isbusy()
65  {
66      DataDir_LCD_port &= ~(0xff<<DATA_shift) ;
67      LCD_control |= (1<<ReadWrite); // read
68      LCD_control &= ~ (1<<RS_switch);
69
70      //  while (LCD_port >= 0x80)
71      //  {
72      LCD_lcd_kick ();
73      //  }
74
75      //  _delay_ms (500) ;
76      DataDir_LCD_port = 0xFF; //0xFF means 0b11111111
77      LCD_control &= ~ (1<<ReadWrite); //write
78  }
79
80
81
82
83  void LCD_lcd_kick ()
84  {
85      LCD_control &= ~ (1<<EN_switch);
86      asm volatile ("nop");
87      asm volatile ("nop");
88      _delay_ms (50) ;
89      LCD_control |= 1<<EN_switch;
90  }
91
```

```
93  void LCD_Send_A_Command(unsigned char command)
94  {
95  #ifdef EIGHT_BIT_MODE
96      LCD_check_lcd_isbusy();
97
98      LCD_port = command  ;
99      LCD_control &= ~ ((1<<ReadWrite)|(1<<RS_switch));
100     LCD_lcd_kick ();
101     //LCD_port = 0;
102 #endif
103 #ifdef FOUR_BIT_MODE
104     LCD_check_lcd_isbusy();
105     LCD_port = (LCD_port & 0x0F) | (command & 0xF0);
106     //LCD_VoidLcd_waitIfBusy();
107     //SET_DATA_DIRECTION_REGISTER(LCD_DATA_DIRECTION_PORT, PORT_OUTPUT_DIRECTION);
108     //LCD_SET_REGISTER_SELECT(LCD_REGISTER_SELECT_DISABLE);
109     //  LCD_SET_READ_WRITE(WRITE_TO_LCD);
110     LCD_control &= ~ ((1<<ReadWrite)|(1<<RS_switch));
111     //LCD_VoidLcd_referish();
112     LCD_lcd_kick ();
113     LCD_port = (LCD_port & 0x0F) | (command << 4);
114     //LCD_VoidLcd_waitIfBusy();
115     LCD_control &= ~ ((1<<ReadWrite)|(1<<RS_switch));
116     //LCD_VoidLcd_referish();
117     LCD_lcd_kick ();
118 #endif
119
120 }
```

LCD.C

```c
void LCD_Send_A_Command_4mode(unsigned char command)
{
    LCD_check_lcd_isbusy();
    LCD_port&= 0x0F;
    LCD_port |= (command&0xF0);
    LCD_control &= ~ ((1<<ReadWrite)|(1<<RS_switch));
    LCD_lcd_kick ();
    LCD_control &= 0x0F;  // Make Data Nibble as 0000
    LCD_control |= ((command<<4)&0xF0);
    LCD_lcd_kick ();
    //LCD_port = 0;
}
```

```c
void LCD_Send_A_Character(unsigned char character)
{

#ifdef EIGHT_BIT_MODE
    LCD_check_lcd_isbusy();

    LCD_control |= 1<<RS_switch; //turn RS ON for Data mode.
    LCD_port = ( ( (character  ) << DATA_shift)   )  ;
    LCD_control |= 1<<RS_switch; //turn RS ON for Data mode.
    LCD_control &= ~ (1<<ReadWrite);//turn RW off so you can write.


    LCD_lcd_kick ();
    //LCD_port = 0;
#endif

#ifdef FOUR_BIT_MODE
    LCD_port = (LCD_port & 0x0F) | (character & 0xF0);
    LCD_control |= 1<<RS_switch; //turn RS ON for Data mode.
    LCD_control &= ~ (1<<ReadWrite);//turn RW off so you can write.
    LCD_lcd_kick ();
    LCD_port = (LCD_port & 0x0F) | (character << 4);
    LCD_control |= 1<<RS_switch; //turn RS ON for Data mode.
    LCD_control &= ~ (1<<ReadWrite);//turn RW off so you can write.
    LCD_lcd_kick ();

#endif


}
```

LCD.C


```c
void LCD_Send_A_String(char *StringOfCharacters)
{


    int count=0 ;//to count how much char on the line (it should be 16 char only)
    while(*StringOfCharacters > 0)
    {
        count++;
        LCD_Send_A_Character(*StringOfCharacters++);
        if (count == 16  ) // go to the second line
        {
            LCD_GotoXY(1,0); //line 1 position zero
        }
        else if (count == 32 || count==33) // clear screen and show again
        {
            LCD_clear_screen();
            LCD_GotoXY(0,0); //line 0 position zero
            count = 0 ;
        }
    }
}


void LCD_display_number (int Number )
{

    char  str[7];

    sprintf(str,"%d",Number);   // Adjust the formatting to your liking.

    LCD_Send_A_String (str) ;

}
```

LCD.C

```
203 void LCD_display_real_number (double  Number )
204 {
205
206     char  str[16];
207
208
209     char *tmpSign = (Number < 0) ? "-" : "";
210     float tmpVal = (Number < 0) ? -Number : Number;
211
212     int tmpInt1 = tmpVal;                  // Get the integer (678).
213     float tmpFrac = tmpVal - tmpInt1;      // Get fraction (0.0123).
214     int tmpInt2 = tmpFrac * 10000;  // Turn into integer (123).
215
216     // Print as parts, note that you need 0-padding for fractional bit.
217
218     sprintf (str, "%s%d.%04d", tmpSign, tmpInt1, tmpInt2);
219
220
221
222     LCD_Send_A_String (str) ;
223
224 }
225
```

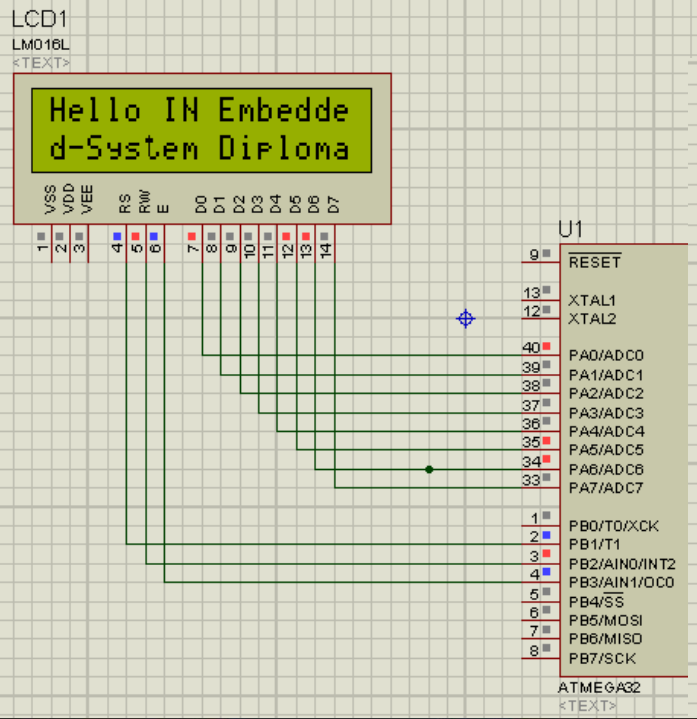
Interview question

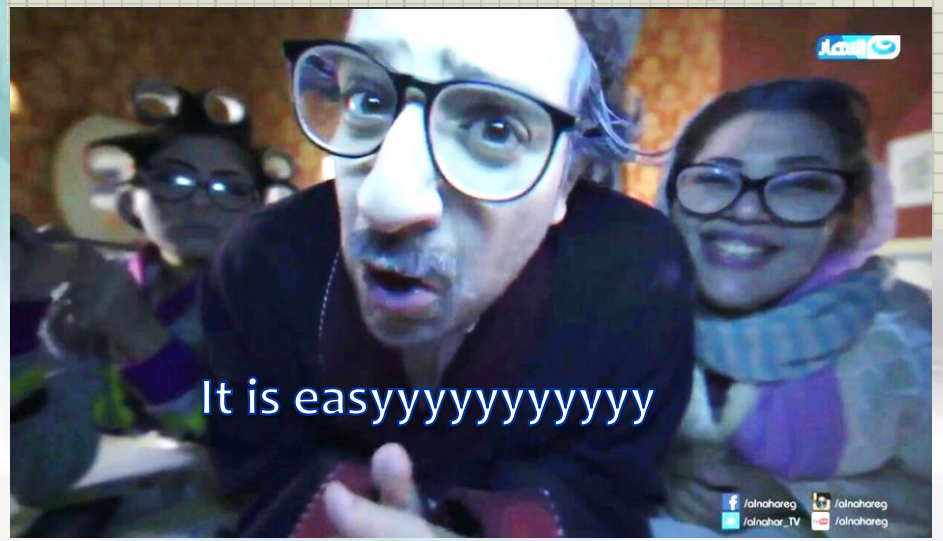
How to display in lcd real number got by ADC?

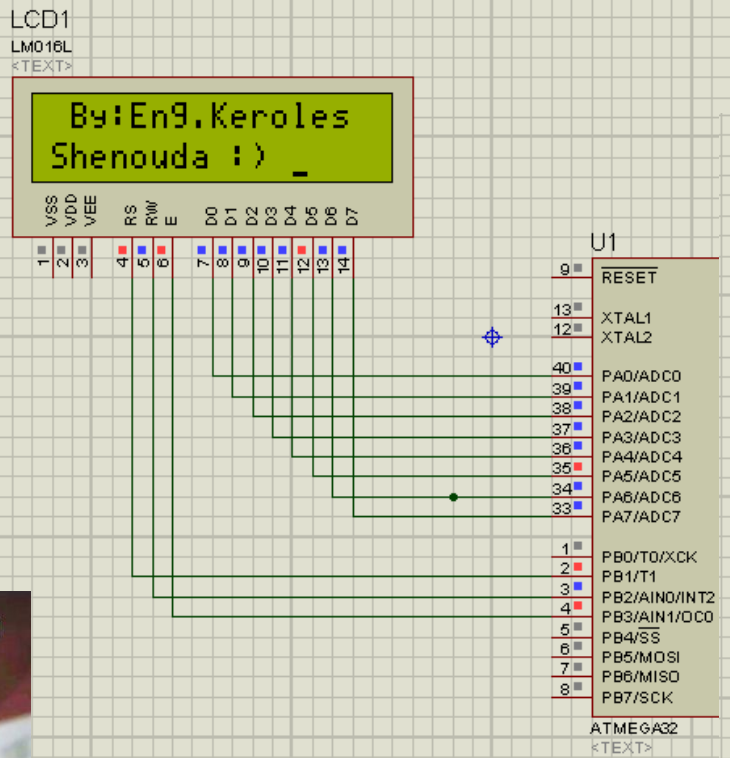
*main.c | lcd.c | lcd.h | mydelay.h

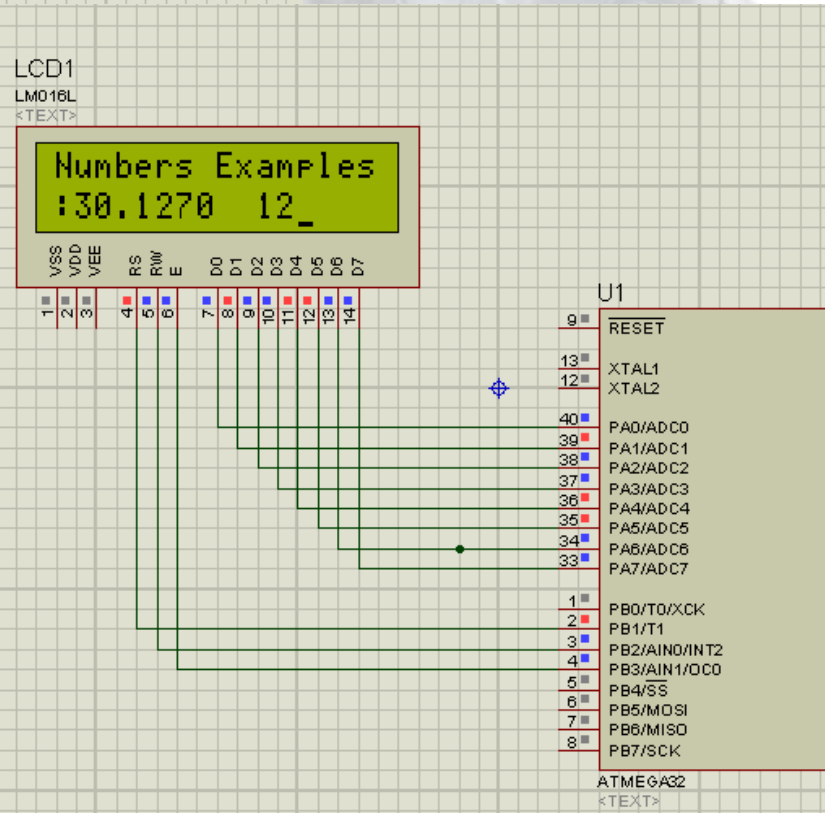
main.c

```
 *  Created on: Aug 4, 2017
 *      Author: Keroles Shenouda
 *      https://www.facebook.com/groups/embedded.system.KS/
 */

#include "GPIO/GPIO.h"
#include "lcd/lcd.h"
#include "mydelay.h"
#define output 1
#define input 0

int main ()
{

    LCD_lcd_init();


    while (1)
    {
        LCD_Send_A_String("Hello IN Embedded-System Diploma By:Eng.Keroles Shenouda :) ");
        LCD_clear_screen();
        LCD_Send_A_String("Numbers Examples:");
        LCD_display_real_number(30.127);
        LCD_Send_A_String("  ");
        LCD_display_number(12);
        LCD_clear_screen();
        DELAY_MS (1000);
    }


    return 0 ;
}
```

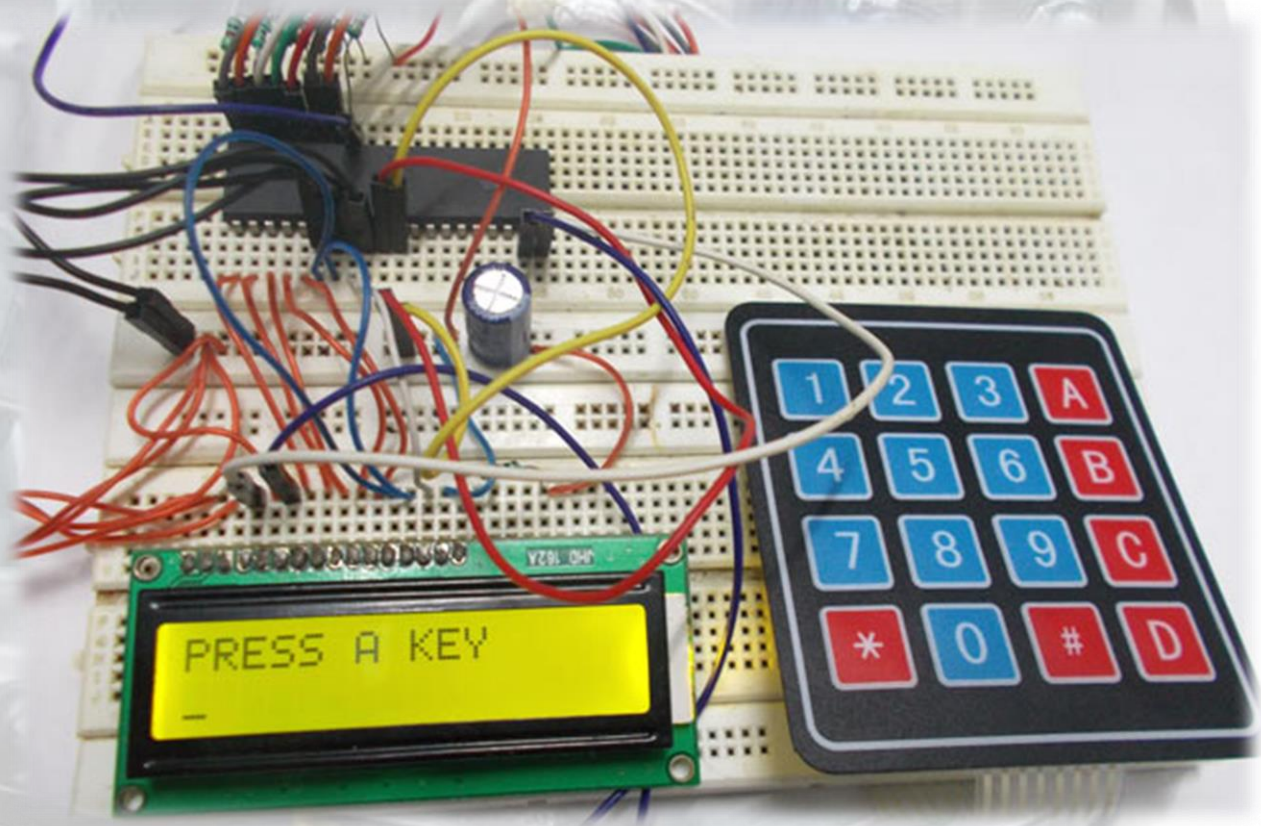

LCD1
LM016L
<TEXT>
Hello IN Embedde
d-System Diploma
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA32
<TEXT>


It is easyyyyyyyyyyy


LCD1
LM016L
<TEXT>
By:En9.Keroles
Shenouda :)
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA32
<TEXT>

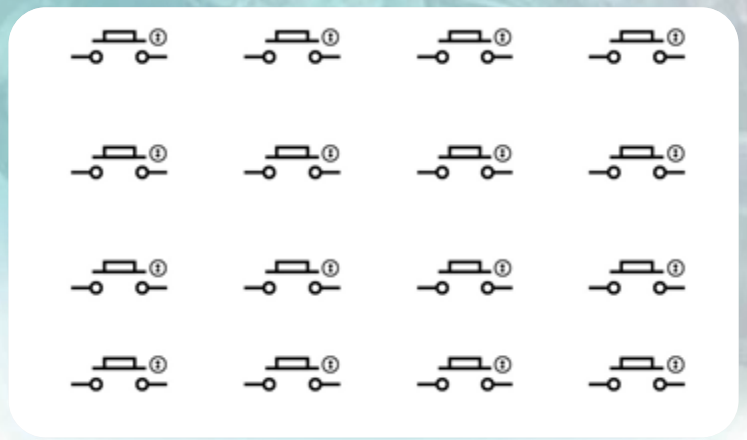

LCD1
LM016L
<TEXT>
Numbers Examples
:30.1270 12
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA32
<TEXT>

Keypad 4*4

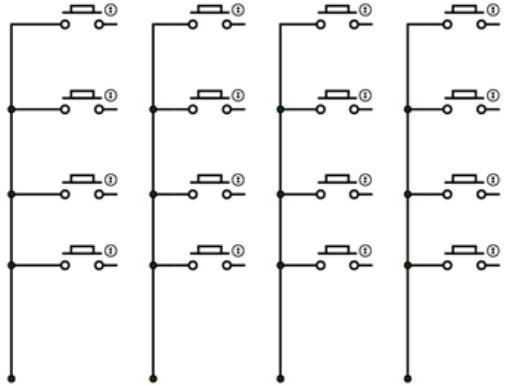


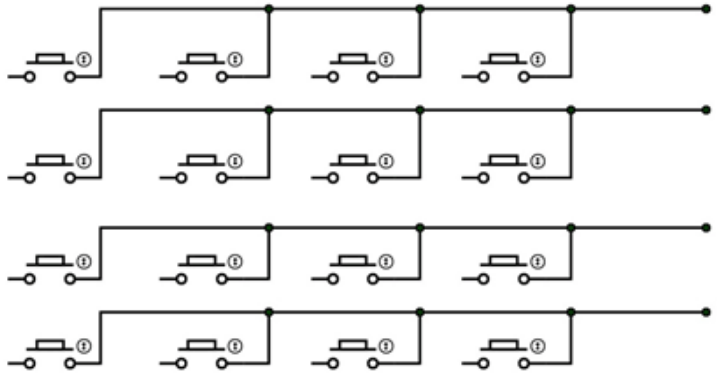
keypad

- keypad is **multiplexed** keys. Buttons are connected in a multiplexed form for reducing the pin usage of control system.
- Consider we have a 4x4 keypad, in this keypad we have **16 buttons**, in normal cases we need 16 controller pins to interface 16 buttons, but this is not good in control system point of view. This pin usage can be reduced by connecting the buttons in **multiplex form.**
- For example consider we have 16 buttons and we want to attach it to a controller to form a keypad, these keys are arranged as shown



These buttons are connected by common columns as shown in figure:

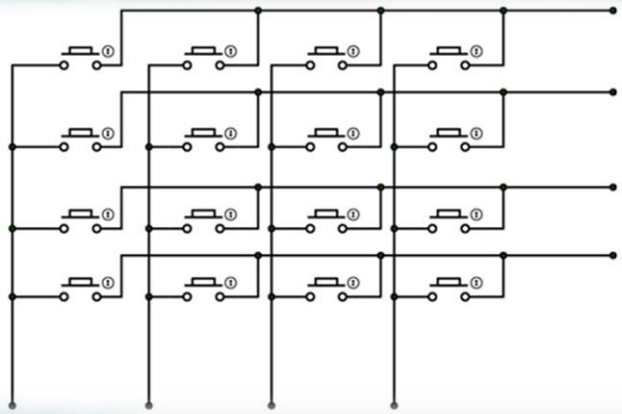




As shown in figure, for 16 keys we will have four rows as shown in figure.

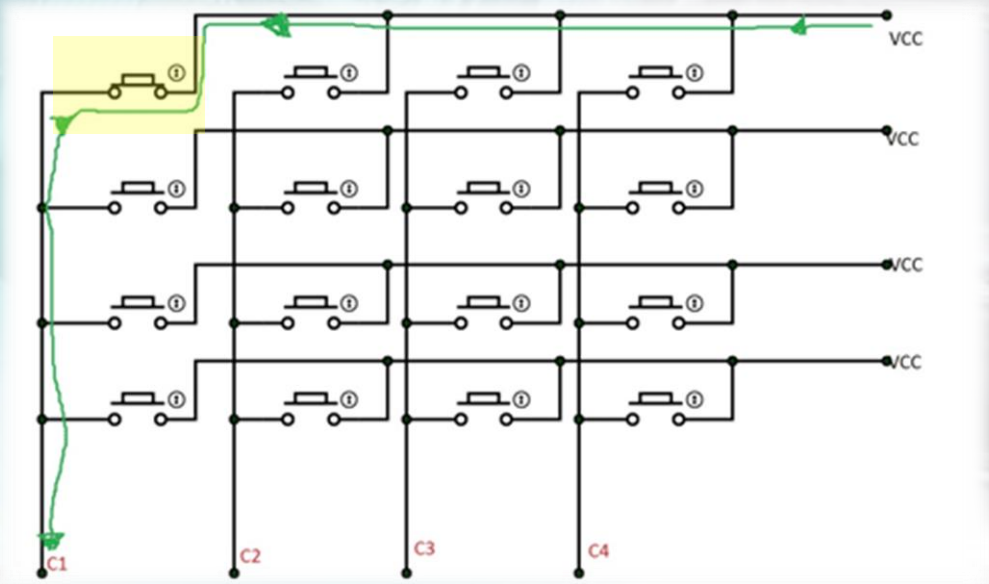
keypad

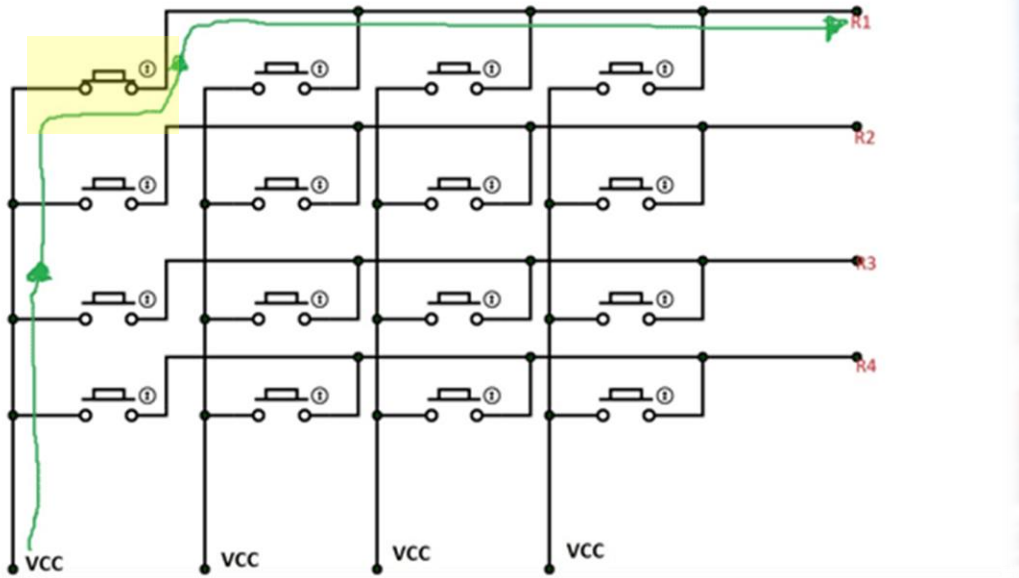
- Here we have connected **16 keys in a multiplexed** form so to reduce the pin usage of controller. When compared to the first case of connected 16 keys we needed 16pins on controller but now after multiplexing **we need simply 8 pins of controller to connect 16 keys.**
- Normally this is what presented inside a keypad:

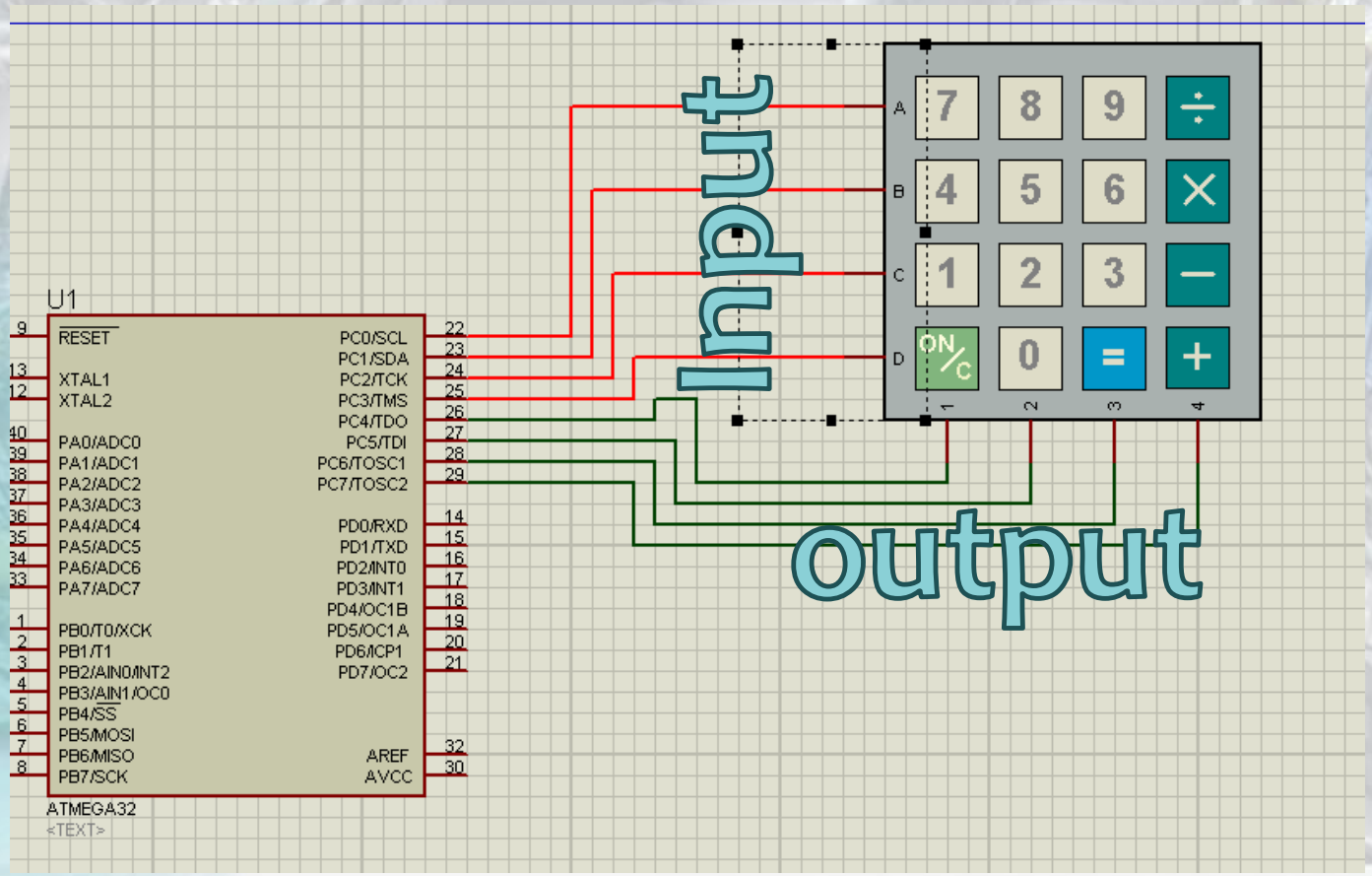


KeyPad

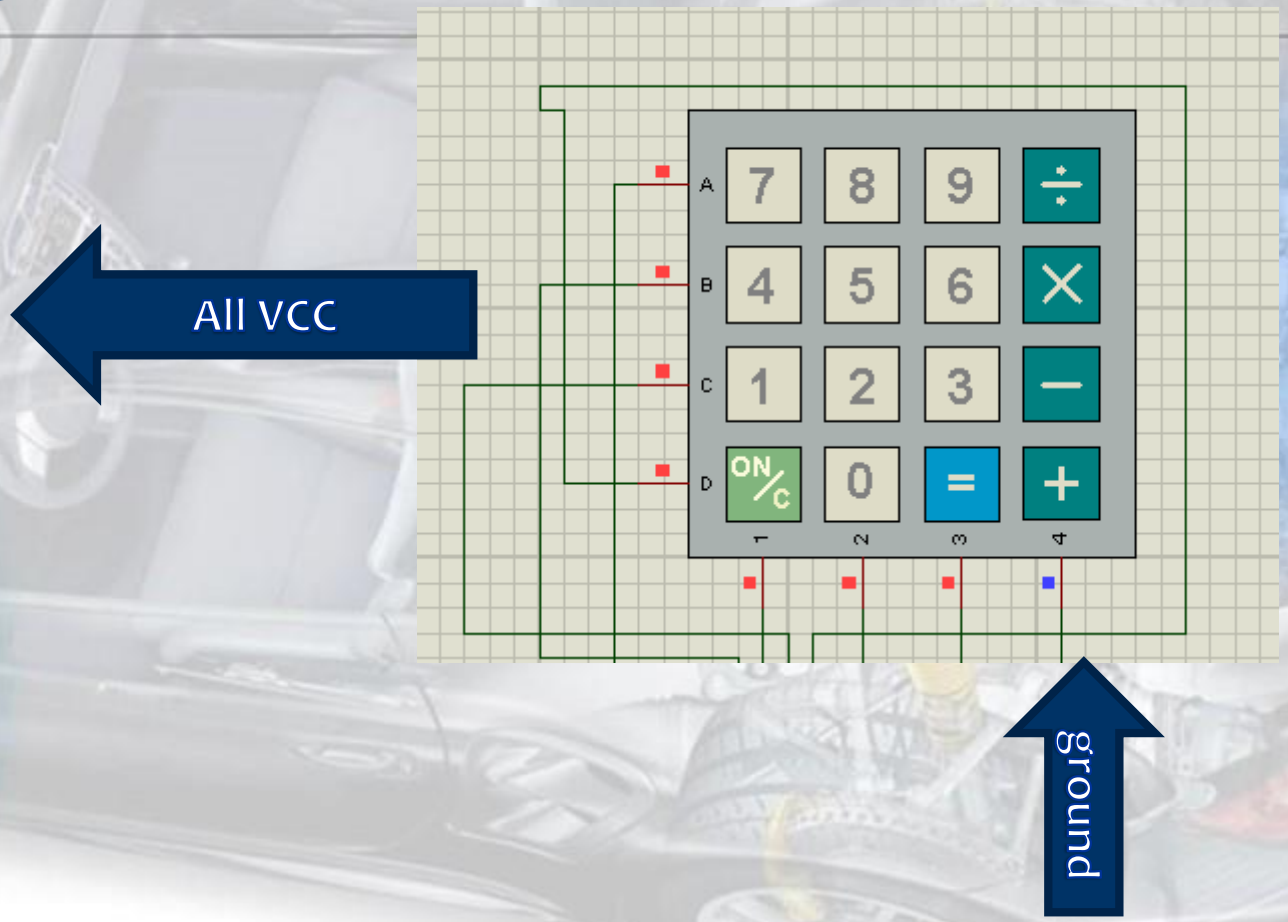
- The keypad here has **four columns** and **four rows**, **for identification of button pressed,** we are going to use **cross reference method**. Here first we are going to either connect all columns or all rows to **vcc**, so if rows are connected to common vcc, we are going to take the **columns as inputs to controller**.



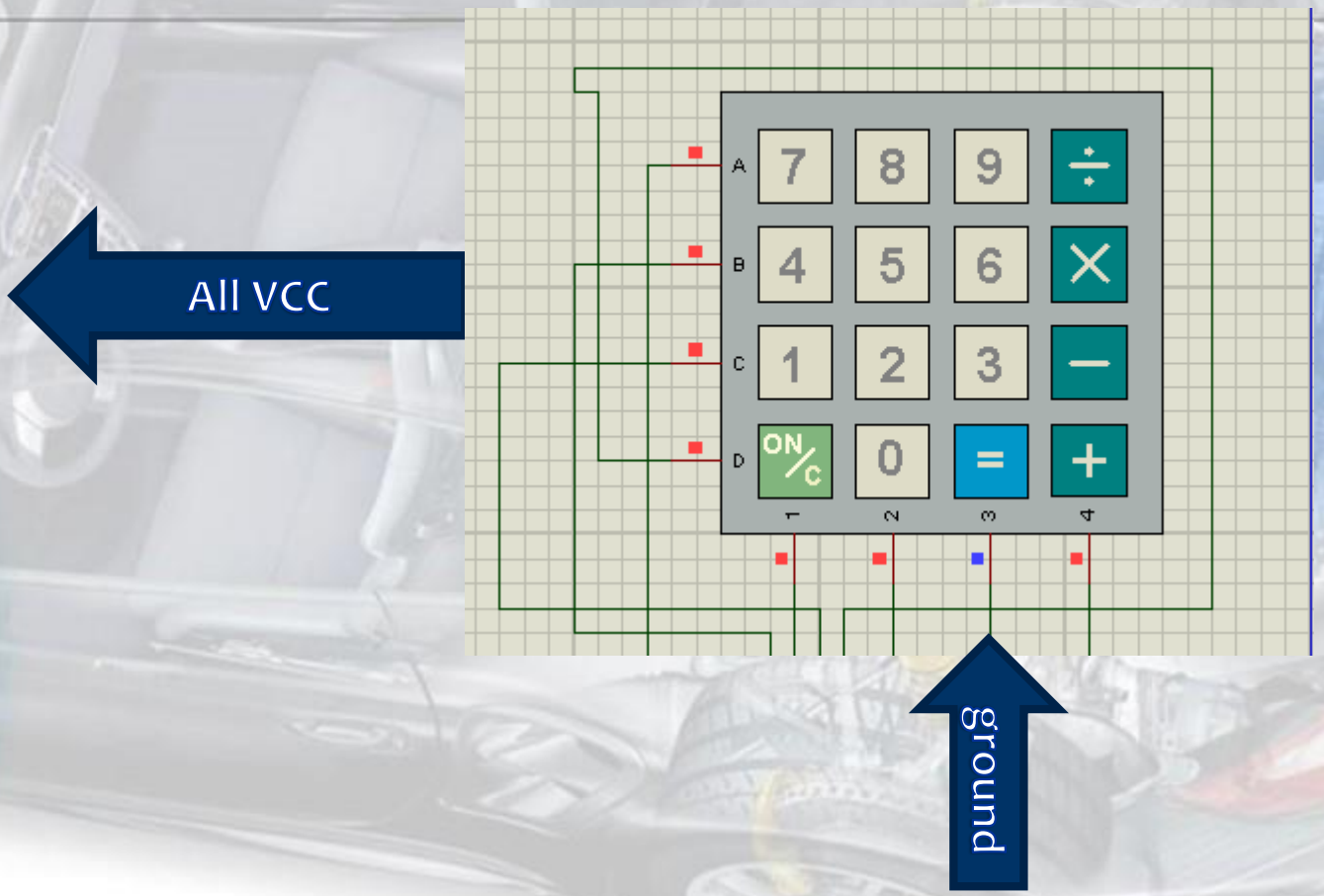



Input
output
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA32
<TEXT>
A
B
C
D
7
8
9
÷
4
5
6
×
1
2
3
−
ON/C
0
=
+

Multiplexing

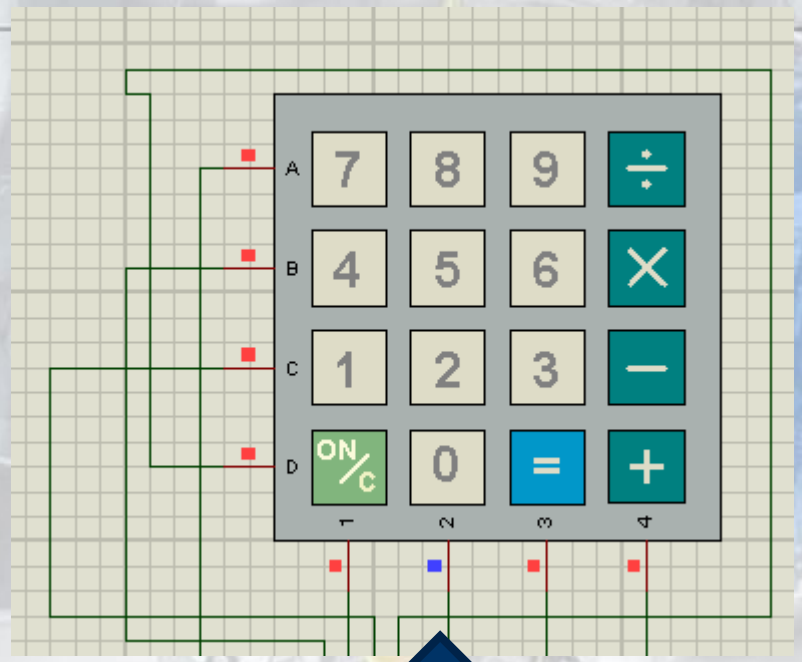


Multiplexing



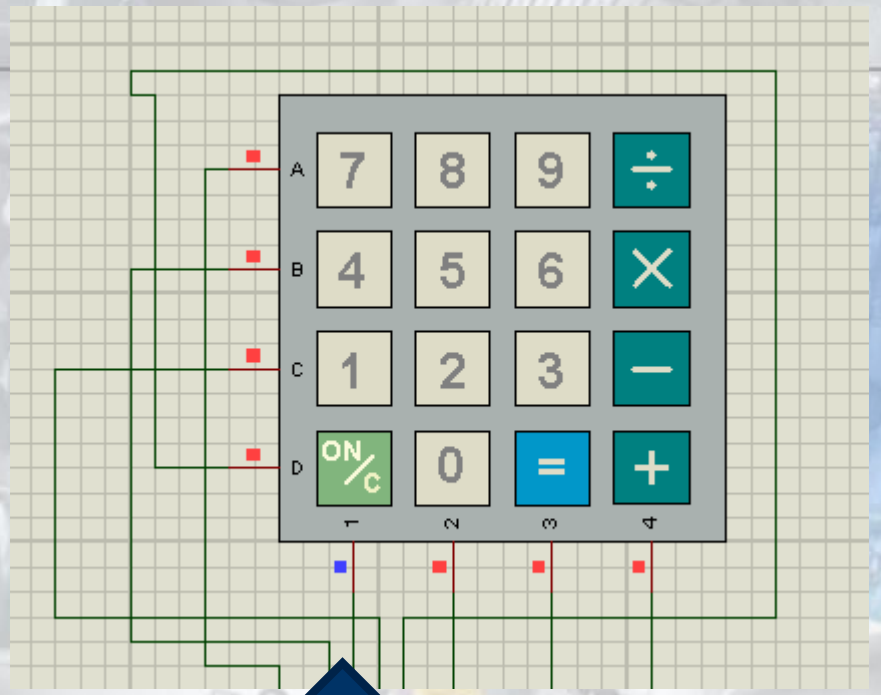
Multiplexing

All VCC



ground

Multiplexing

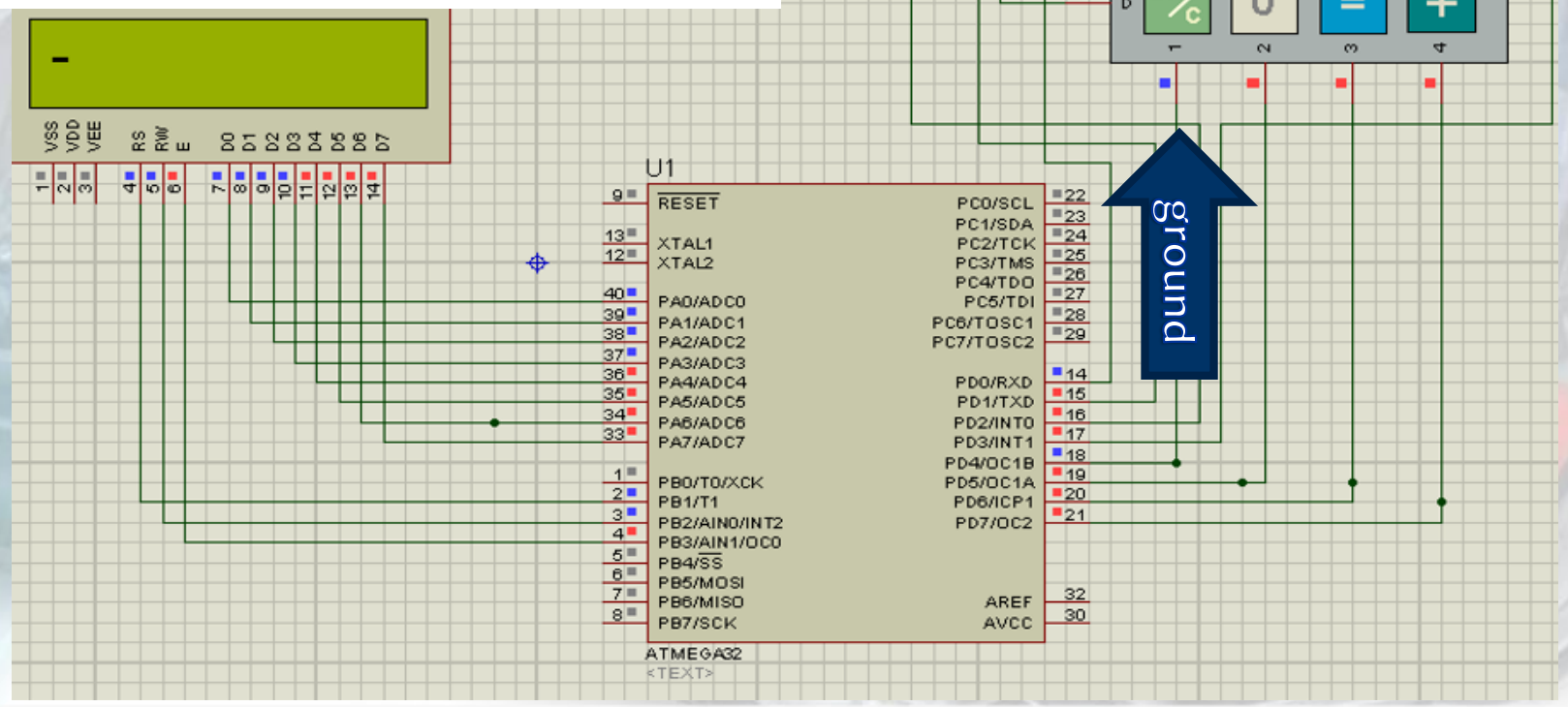


All VCC

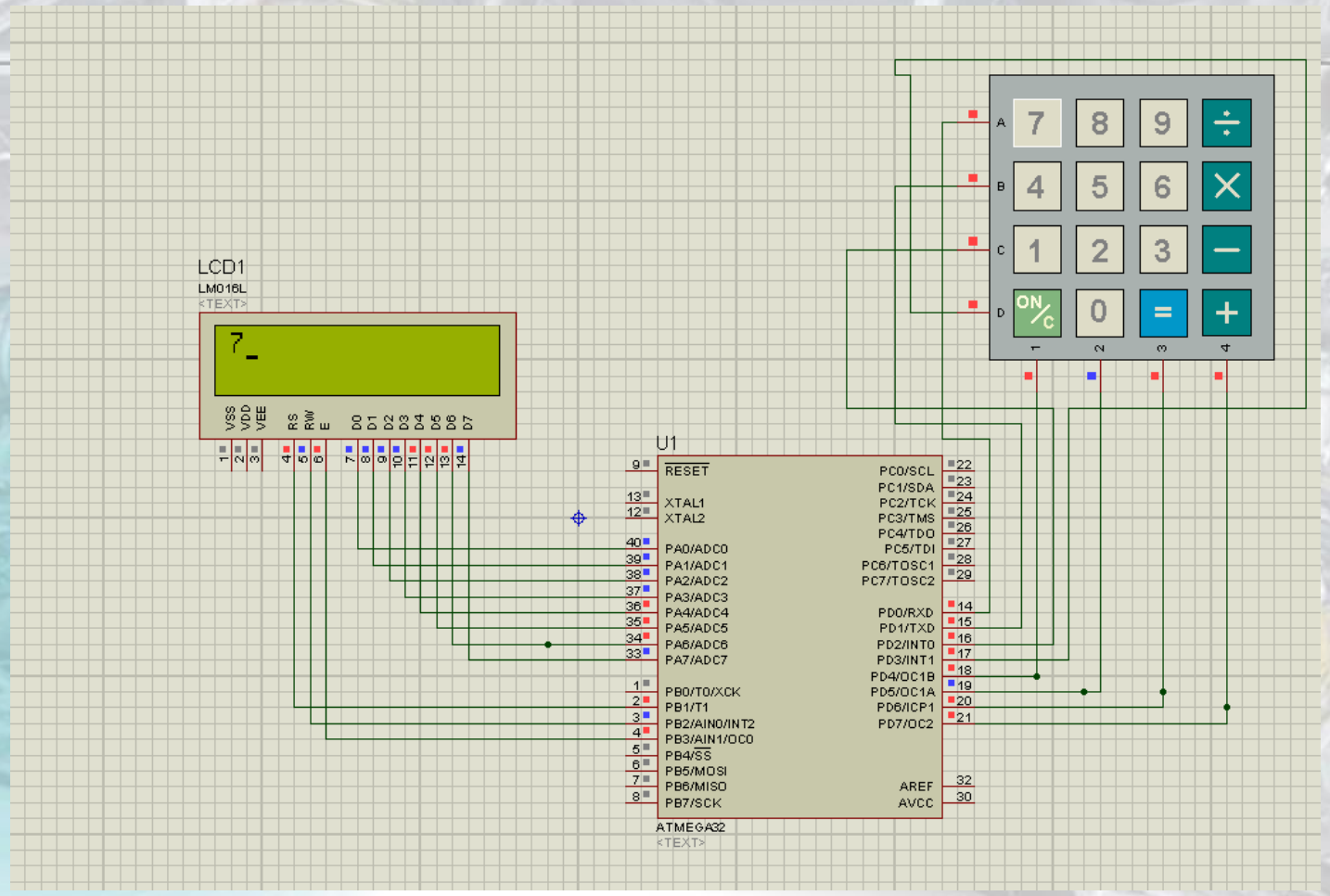
ground

Still pressed

```
char Keypad_getkey()
{
   int i, j;
   for(i = 0; i < 4; i++)
   {
    keypadPORT |= ((1<<Key_padCol[0] ) | (1<<Key_padCol[1] ) | (1<<Key_padCol[2] ) | (1<<Key_padCol[3] ));

    keypadPORT &= ~ (1 << Key_padCol[i]);//send a low to a particular row of the keypad

    for(j = 0; j < 4; j++)
    {
        if(!(keypadPIN & (1<<Key_padRow[j])))//check if key is pressed
        {
            while(!(keypadPIN & (1<<Key_padRow[j])));  //wait for key to be released  (Single press)
            switch(i)
            {
                case(0):
                {
```



Once released



KeyPad Driver

https://github.com/keroles/Embedded_System/tree/master/Atmega32_Drivers/ATMEGA32_Drivers/keypad

Keypad.h

keypad.h

```c
/*
 * keypad.h
 *
 *  Created on: Aug 11, 2017
 *      Author: Keroles Shenouda
 */

#ifndef KEYPAD_KEYPAD_H_
#define KEYPAD_KEYPAD_H_

#include <avr/io.h>
#include<util/delay.h>



//Keypad Information
#define R0 0
#define R1 1
#define R2 2
#define R3 3
#define C0 4
#define C1 5
#define C2 6
#define C3 7

#define keypadPORT PORTD
#define keypadPIN PIND
#define keypadDDR DDRD


//Keypad functions and global variables



void  Keypad_init();
char Keypad_getkey();

#endif /* KEYPAD_KEYPAD_H_ */

```

Keypad.c

keybad_4_4.c

```c
/*
 * keybad_4_4.c
 *
 *  Created on: Aug 11, 2017
 *      Author: Keroles Shenouda
 */


#include "keypad.h"
int Key_padRow[] = {R0, R1, R2, R3}; //rows of the keypad
int Key_padCol[] = {C0, C1, C2, C3};//columnd
void  Keypad_init()
{
    keypadDDR &= ~ (1<<R0)|(1<<R1)|(1<<R2)|(1<<R3);//set upper part of keypad port as input
    //this will be required for scanning the rows
    keypadDDR |= ((1<<C0)|(1<<C1)|(1<<C2)|(1<<C3));//set lower part of keypad port as output.This is
    //the part of the keypad port where the rows are connected.
    keypadPORT = 0xFF ;
}


```

Keypad.c


```
23  char Keypad_getkey()
24  {
25      int i, j;
26      for(i = 0; i < 4; i++)
27      {
28          keypadPORT |= ((1<<Key_padCol[0] ) | (1<<Key_padCol[1] ) | (1<<Key_padCol[2] ) | (1<<Key_padCol[3] ));
29
30          keypadPORT &= ~ (1 << Key_padCol[i]);//send a low to a particular row of the keypad
31
32          for(j = 0; j < 4; j++)
33          {
34              if(!(keypadPIN & (1<<Key_padRow[j])))//check if key is pressed
35              {
36                  while(!(keypadPIN & (1<<Key_padRow[j])));  //wait for key to be released  (Single press)
37                  switch(i)
38                  {
39                  case(0):
40                      {
41                      if (j == 0) return '7';
42                      else if (j == 1) return '4';
43                      else if (j == 2) return '1';
44                      else if (j == 3) return '?';
45                      break;
46                      }
47                  case(1):
48                      {
49                      if (j == 0) return '8';
50                      else if (j == 1) return '5';
51                      else if (j == 2) return '2';
52                      else if (j == 3) return '0';
53                      break;
54                      }
```

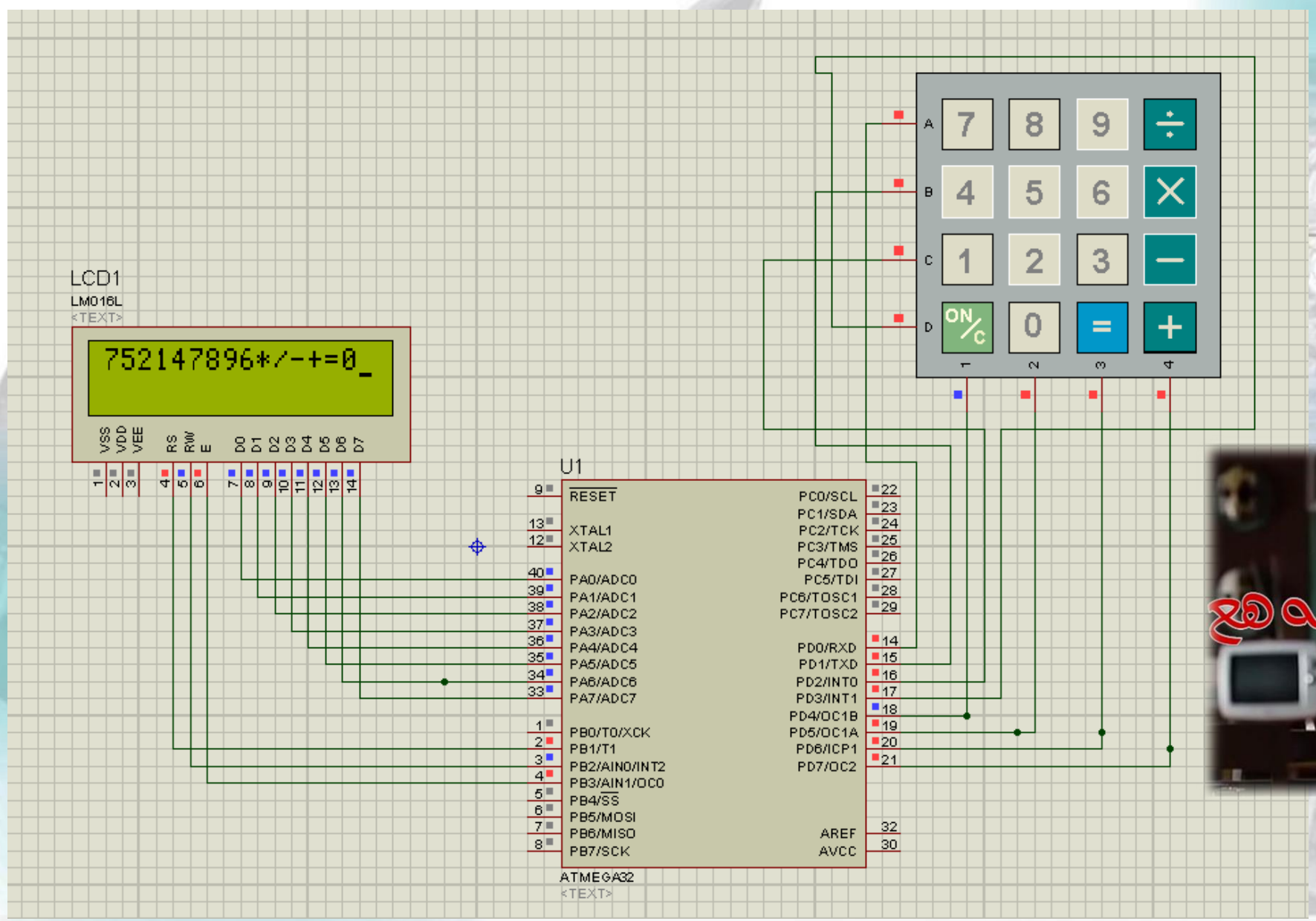
Keypad.c


```
55                  case(2):
56                      {
57                      if (j == 0) return '9';
58                      else if (j == 1) return '6';
59                      else if (j == 2) return '3';
60                      else if (j == 3) return '=';
61                      break;
62                      }
63                  case(3):
64                      {
65                      if (j == 0) return '/';
66                      else if (j == 1) return '*';
67                      else if (j == 2) return '-';
68                      else if (j == 3) return '+';
69                      break;
70                      }
71                  }
72              }
73          }
74      }
75      return 'A';//Return 'A' if no key is pressed.
76  }
77
```

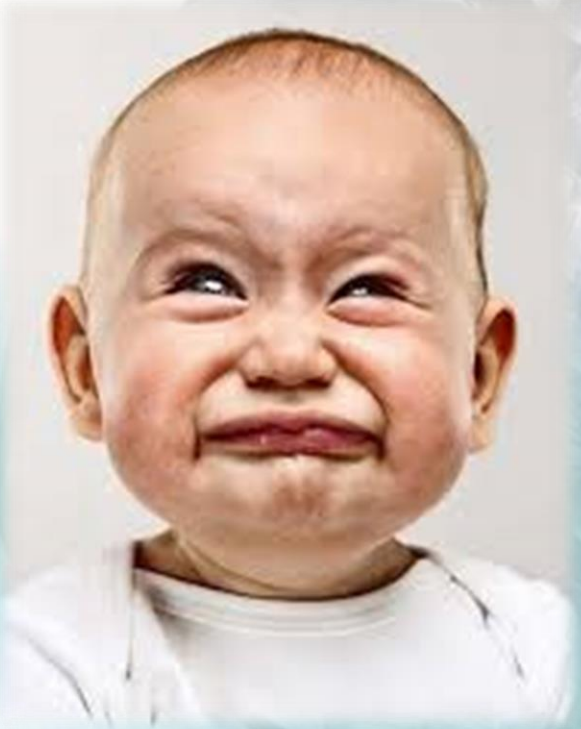
main.c

keybad_4_4.c | main.c

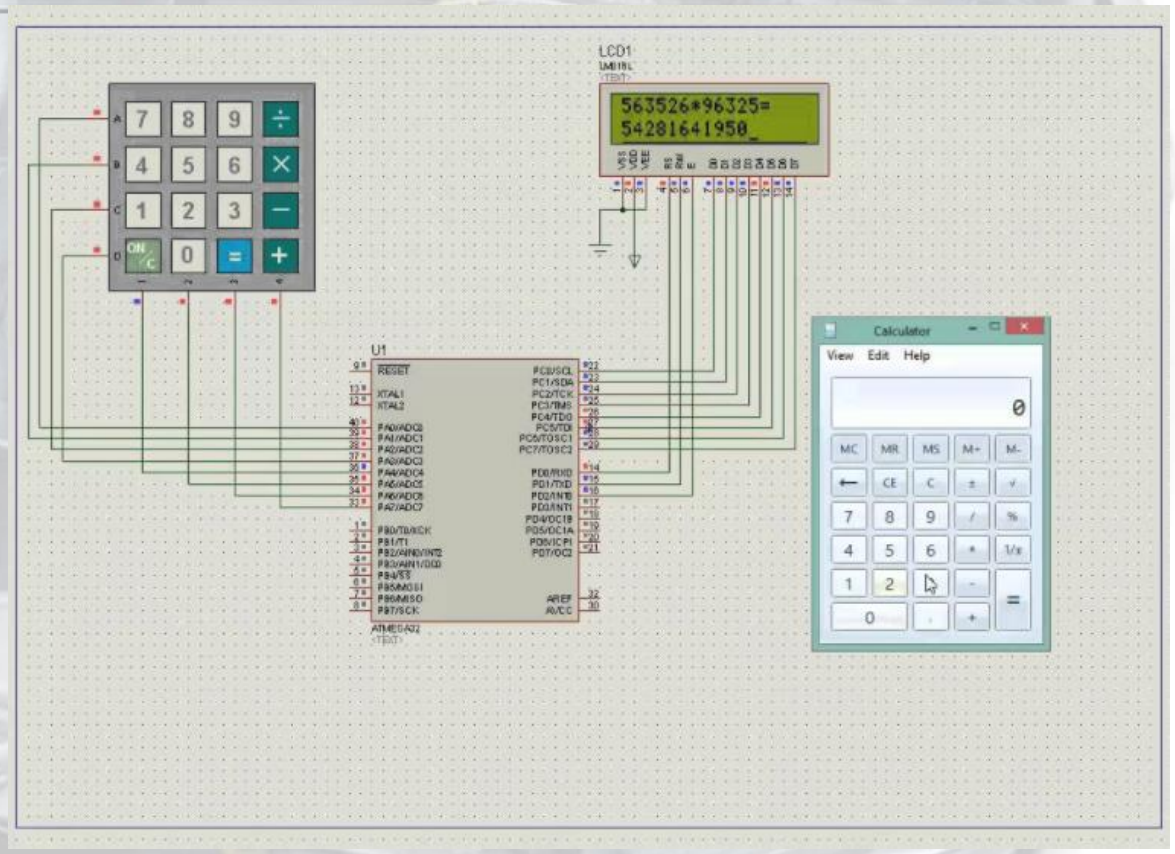
```c
/*
 * main.c
 *
 *  Created on: Aug 4, 2017
 *      Author: Keroles Shenouda
 *      https://www.facebook.com/groups/embedded.system.KS/
 */

#include "GPIO/GPIO.h"
#include "lcd/lcd.h"
#include "mydelay.h"
#include "keypad/keypad.h"
#define output 1
#define input 0

int main ()
{
    char key_pressed;
    LCD_lcd_init();
    Keypad_init();
    while (1)
    {

        key_pressed = Keypad_getkey();
        switch(key_pressed)
        {
        case('A'):
         break;//do nothing if no key is pressed
        case('?'):
            LCD_clear_screen();
         break;//do nothing if no key is pressed
        default:
            LCD_Send_A_Character(key_pressed);//send the key pressed to LCD
        }
        //_delay_ms(100);

    }


    return 0 ;
}
```


LCD1
LM016L
<TEXT>
752147896*/-+=0_
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10 11 12 13 14
A 7 8 9 ÷
B 4 5 6 ×
C 1 2 3 −
D ON/C 0 = +
1 2 3 4
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA32
<TEXT>


ههههه
i2Picture.com



Lab: Create a Calculator on ATmega32



References

- https://www.newbiehack.com/MicrocontrollersABeginnersGuideIntroductionandInterfacinganLCD.aspx
- http://www.slideshare.net/MathivananNatarajan/asynchronous-serial-data-communication-and-standards
- **4x4 Keypad Interfacing with ATmega32 Microcontroller**


Thank
You