# Introduction to Embedded System Design

## Interfacing Liquid Crystal Displays with MSP430


Dhananjay V. Gadre
Associate Professor
ECE Division
Netaji Subhas University of Technology, New Delhi

Badri Subudhi
Assistant Professor
Electrical Engineering Department
Indian Institute of Technology, Jammu

# Liquid Crystal Displays

- Numeric
- Alphanumeric
- Character
- Graphics

# Liquid Crystal Displays

- Liquid crystal display technology works by manipulating light.
- Uses two polarizers - horizontal and vertical placed on top of each other, with a source of light at the bottom.
- Liquid crystal between the two polarizers
- Liquid Crystals, when potential is applied, have the ability to ‘twist’ the light.

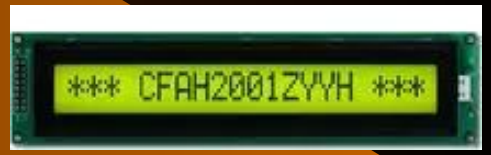
# Liquid Crystal Display Types

- Character LCD Type: 8*1, 8*2, 12*2, 16*1, 16*2, 16*4, 20*1, 20*2, 20*4, 40*1


8*2


12*2

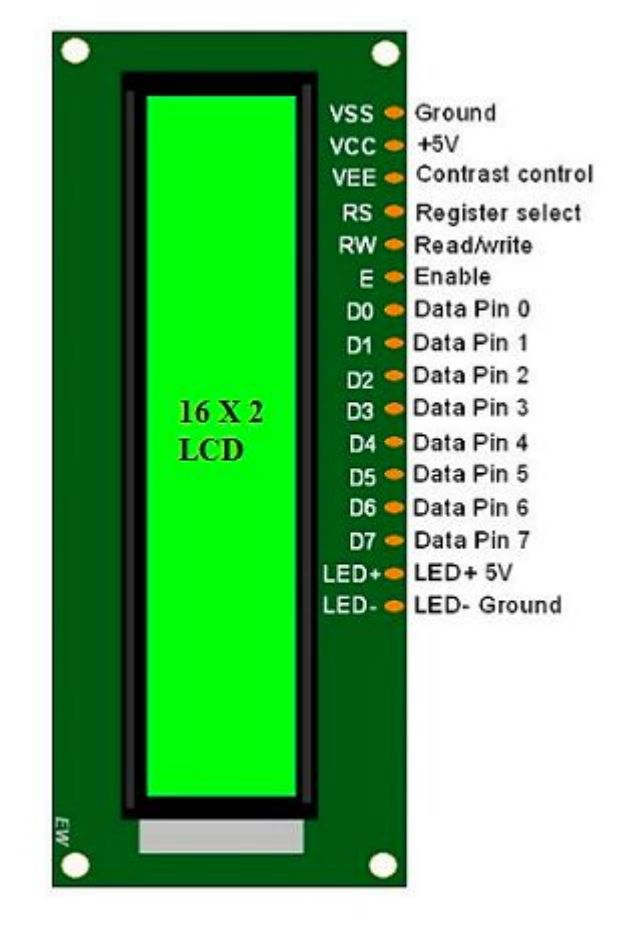

16*2



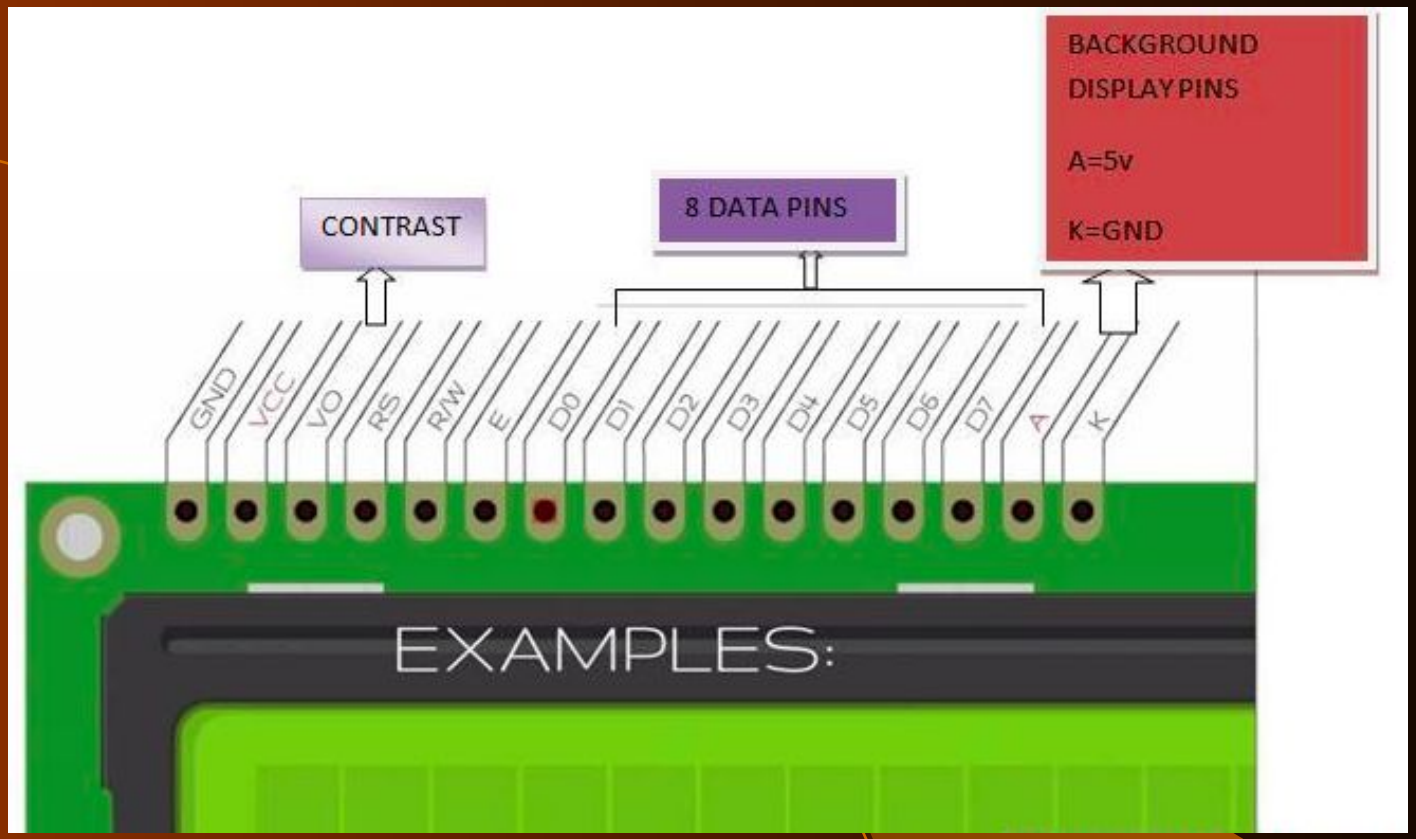

20*1

# Hitachi HD44780 LCD Controller

- In this course, we will use the 16×2 LCD display, which is a very basic module (16-pin) commonly used in DIYs circuits.
- The 16×2 translates to a display 16 characters per line in 2 such lines.
- In LCD displays, each character is in a 5×8 matrix. Where 5 are the number of columns and 8 is the number of rows. This knowledge will be used when we have to print custom text or letters.



# Pinout for 16X2 LCD




As shown in the figure above, it a has pin for contrast, 8 Data pins along with  pins for GND and VCC.
R/W is Read Write pin. RS is the Register Select pin
E is the Enable pin.

# 16X2 LCD Pin Description:

| PIN | | Description |
|---|---|---|
| • **Ground Pin (GND) and**<br>• **Supply Pin (4.7V - 5.3V)** | GND<br>VCC | **Grounded pin and Vcc is given on the respective pin.** |
| • **Read/Write and**<br>• **An Enable pin (-ve Edge triggered for Write and +ve edge triggered for Read)** | R/W*<br>E | 1. **Low to write to the register. High to read from the register**<br>2. **An edge triggered signal used to writing or reading data to/from LCD** |
| • Contrast adjustment.<br>A variable resistor (mostly a preset) is generally attached on this pin. | VO/VEE | Output of the potentiometer is connected to this pin. Rotate the potentiometer knob forward and backwards to adjust the LCD contrast. |
| • 8 Data pins | D0-D7 | 8 data pins for data transfer. |
| • Register Select: (A 16X2 LCD has two registers, namely, command and data.) | RS | The register select is used to switch from one register to other. RS=0 for command register, whereas RS=1 for data register. |
| • Backlit Supply (5V) | LED+ | |
| • Backlit Ground (GND) | LED- | |

# Command Codes for LCD

| S.no | Hex Code | Command to LCD Instruction Register |
|---|---|---|
| 1. | 01 | Clear Display Screen |
| 2. | 02 | Return Home |
| 3. | 04 | Decrement Cursor(shift to left) |
| 4. | 06 | Increment Cursor |
| 5. | 05 | Shift Display Right |
| 6. | 07 | Shift Display Left |
| 7. | 08 | Display Off, Cursor Off |
| 8. | 0A | Display Off, Cursor On |
| 9. | 0C | Display On, Cursor Off |
| 10. | 0E | Display Off, Cursor On |
| 11. | 0F | Display On, Cursor Blinking |
| 12. | 10 | Shift Cursor to Left |
| 13. | 14 | Shift Cursor to Right |
| 14. | 18 | Shift Entire display to the Left |
| 15. | 1C | Shift Entire display to the Right |
| 16. | 80 | Force Cursor to Beginning (1st line) |
| 17. | C0 | Force Cursor to Beginning (2nd line) |
| 18. | 38 | 2 line and 5X7 Matrix |
| 19. | 28 | 2 line 5X7 matrix in 4bit mode |
| 20. | 32 | Send for 4bit initialisation of LCD |
| 21. | 33 | Send for 4bit initialisation of LCD |

# 16*2 LCD Working Modes

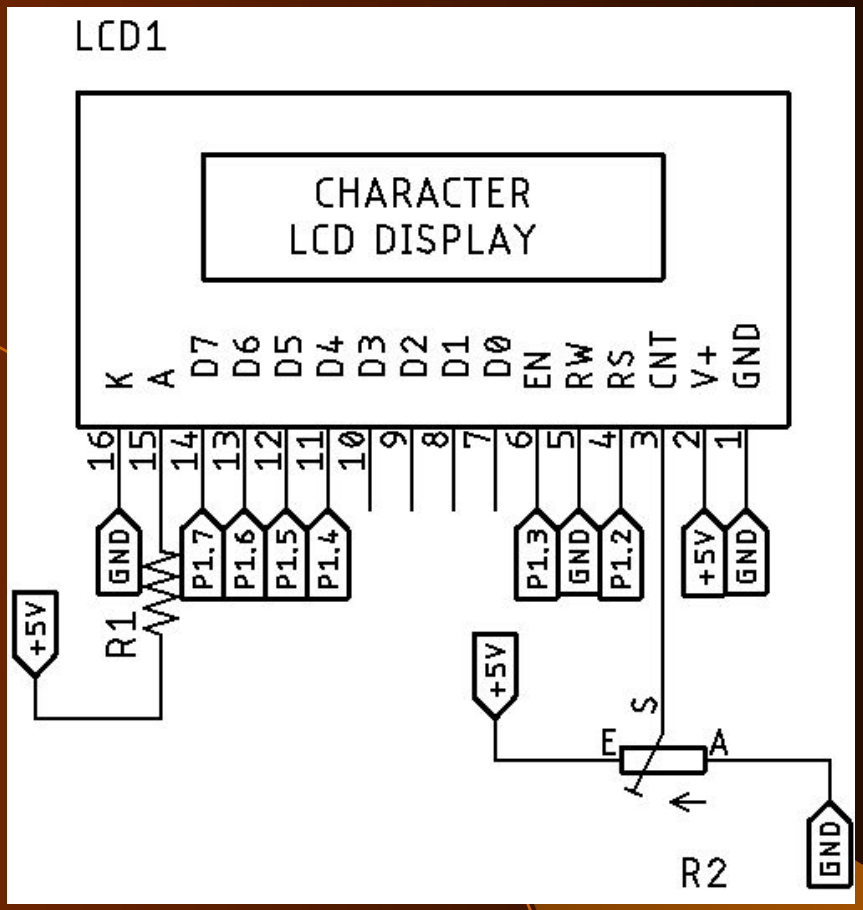
- 16*2 character display can be operated in two modes -
  - 8 Bit mode - This is the default mode in which all the data pins (D0 - D7) are used.
  - 4 Bit mode - Only four data pins (D4 - D7) are used.

# 4 Bit mode

- In 4-bit mode, data/command is sent in 4-bit (nibble) format.
- To do this 1st send Higher 4-bit and then send lower 4-bit of data / command.
- Only 4 data (D4 - D7) pins of 16x2 of LCD are connected to microcontroller and other control pins RS (Register select), RW (Read / write), E (Enable) are connected to other GPIO Pins of controller.

```
LCD_Command(0x33);
LCD_Command(0x32);      /* Send for 4 bit initialization of LCD  */
LCD_Command(0x28);      /* 2 line, 5*7 matrix in 4-bit mode */
LCD_Command(0x0c);      /* Display on cursor off */
LCD_Command(0x06);      /* Increment cursor (shift cursor to right)
LCD_Command(0x01);      /* Clear display screen */
```

# LCD Connections



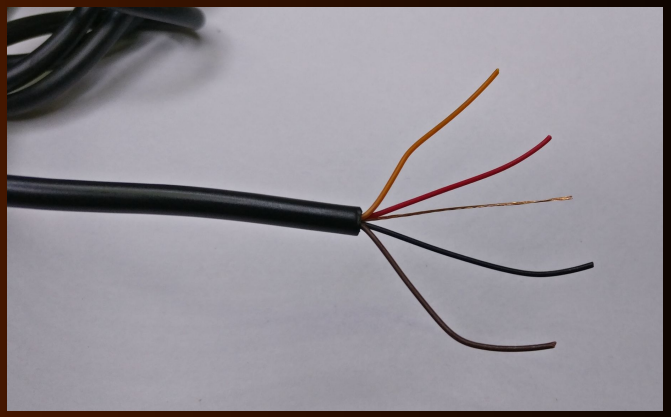

- Data pins, EN, RW connected to LunchBox
- +5V from external power supply.
  ***Remember to connect GND of +5V supply and the GND of LunchBox.***

# +5V Power Supply Options

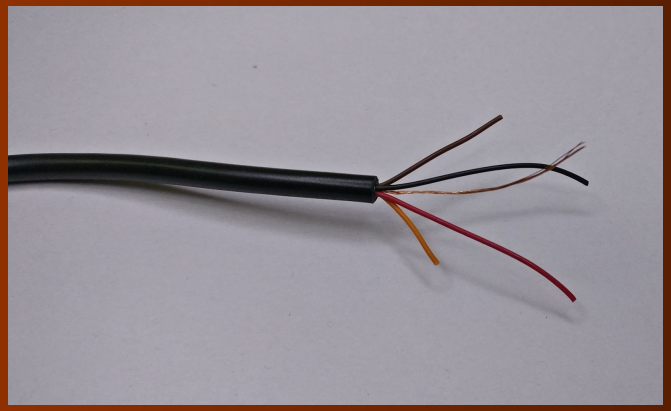
1. Modified USB cable
2. Breadboard Power Supply
3. Lab Bench Power Supply

# Modifying the USB Cable

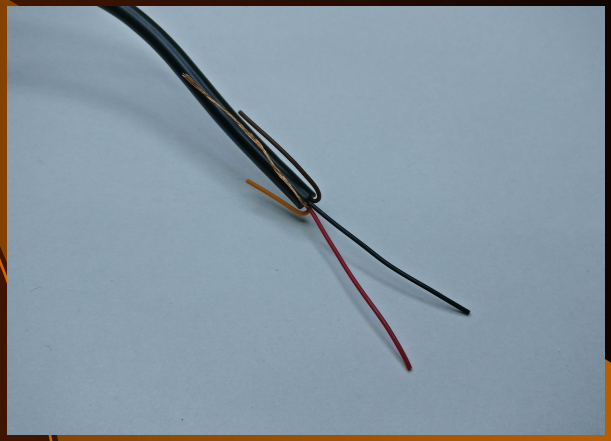
Step1: Cut the end (other than the USB Type A male end) of the cable.



Step2: Strip the insulator part of the cable to see five different wires.

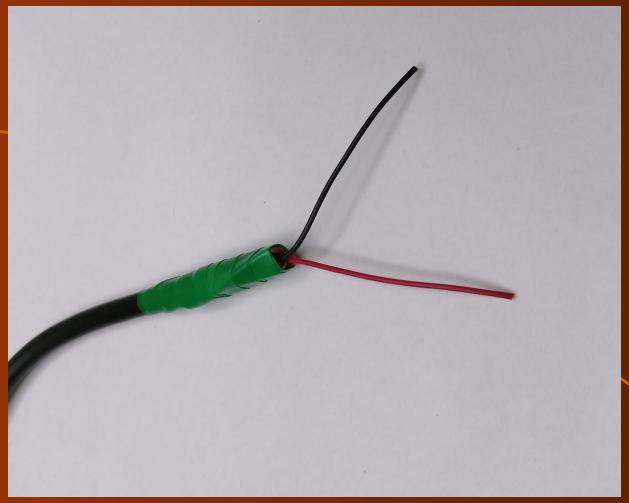


Step3: The Red and Black ones are used for the Power supply. Cut the three other wires at different lengths.

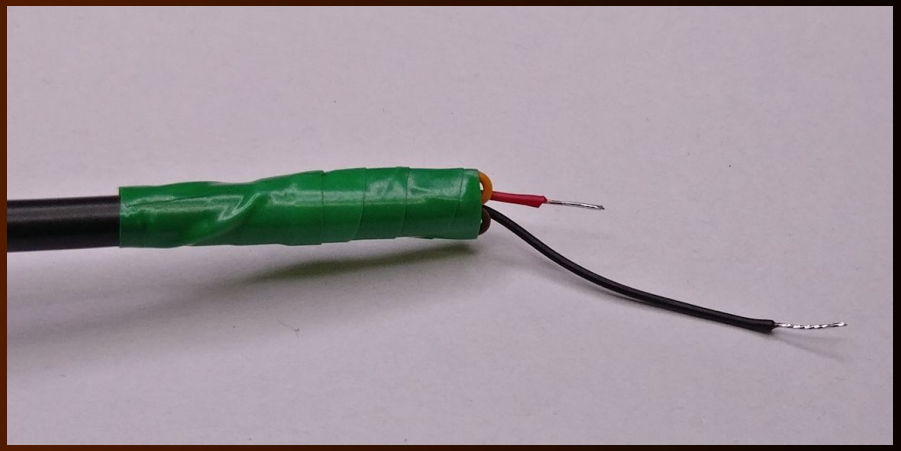


Step4: Now, bend the three wires.

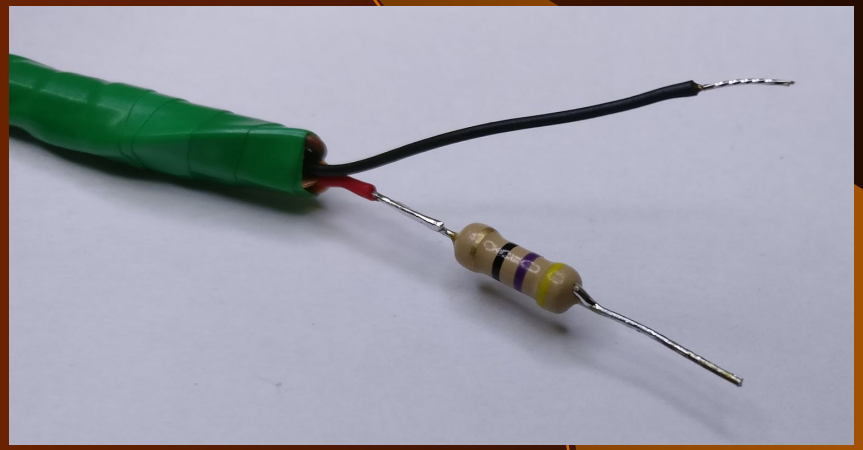
# Modifying the USB Cable



Step5: Tape the three ends with the help of an insulation tape.



Step6: Cut the red wire short. Now, strip the ends of the red and black wires and use a soldering iron to tin the ends.



Step7: Solder one end of a 47 ohm resistor with the Red wire.

# The 'Hello LCD' Code

```
1  #include <msp430.h>
2  #include <inttypes.h>
3
4  #define CMD         0
5  #define DATA        1
6
7  #define LCD_OUT     P1OUT
8  #define LCD_DIR     P1DIR
9  #define D4          BIT4
10 #define D5          BIT5
11 #define D6          BIT6
12 #define D7          BIT7
13 #define RS          BIT2
14 #define EN          BIT3
15
16 //
17 /**
18  *@brief Delay function for producing delay in 0.1 ms increments
19  *@param t milliseconds to be delayed
20  *@return void
21  **/
22 void delay(uint8_t t)
23 {
24     uint8_t i;
25     for(i=t; i > 0; i--)
26         __delay_cycles(100);
27 }
```

We have used uint8_t, over here, because our variable is such that its value will not exceed 255 in delay function

In this code, uint8_t datatype is used to declare the variable.
uint8 is used to write implementation-independent code.
unsigned char is not guaranteed to be an 8-bit type in many implementation but. uint8_t is.
uint8 is defined in inttypes.h header file.

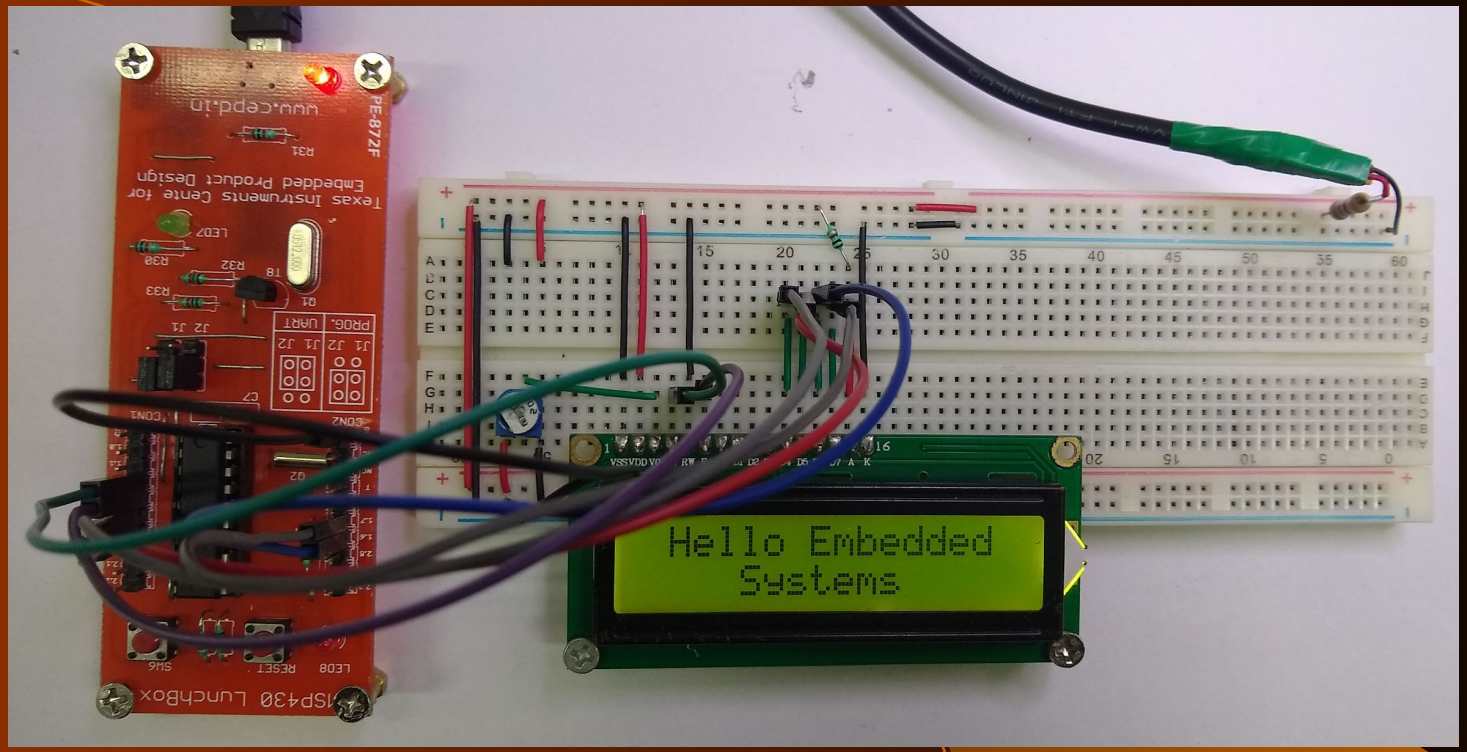
```
28
29  /**
30   *@brief Function to pulse EN pin after data is written
31   *@return void
32   **/
33  void pulseEN(void)
34  {
35      LCD_OUT |= EN;
36      delay(1);
37      LCD_OUT &= ~EN;
38      delay(1);
39  }
40
41  /**
42   *@brief Function to write data/command to LCD
43   *@param value Value to be written to LED
44   *@param mode Mode -> Command or Data
45   *@return void
46   **/
47  void lcd_write(uint8_t value, uint8_t mode)
48  {
49      if(mode == CMD)
50          LCD_OUT &= ~RS;             // Set RS -> LOW for Command mode
51      else
52          LCD_OUT |= RS;              // Set RS -> HIGH for Data mode
53
54      LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));              // Write high nibble first
55      pulseEN();
56      delay(1);
57
58      LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));       // Write low nibble next
59      pulseEN();
60      delay(1);
61  }
```

We know any character will be less than (0-255) 2^8 value in its ASCII notation.

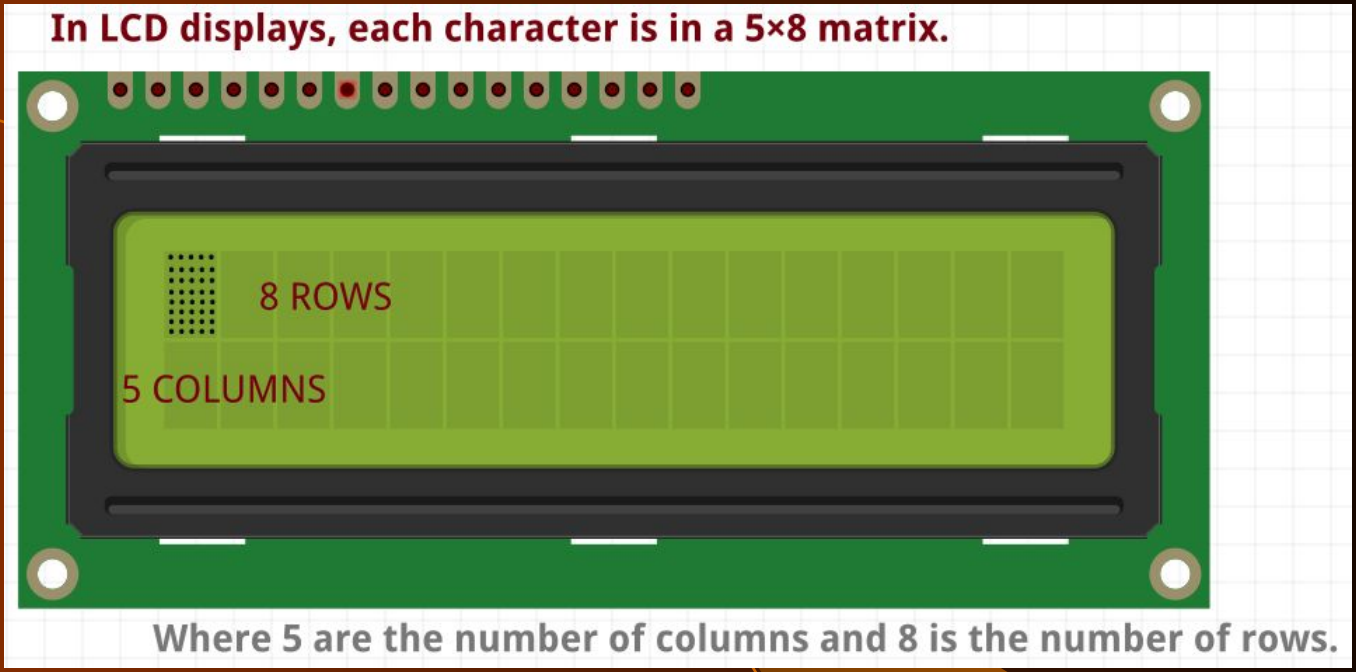
```
62
63  /**
64   *@brief Function to print a string on LCD
65   *@param *s pointer to the character to be written.
66   *@return void
67   **/
68  void lcd_print(char *s)
69  {
70      while(*s)
71      {
72          lcd_write(*s, DATA);
73          s++;
74      }
75  }
76
77  /**
78   *@brief Function to move cursor to desired position on LCD
79   *@param row Row Cursor of the LCD
80   *@param col Column Cursor of the LCD
81   *@return void
82   **/
83  void lcd_setCursor(uint8_t row, uint8_t col)
84  {
85      const uint8_t row_offsets[] = { 0x00, 0x40};
86      lcd_write(0x80 | (col + row_offsets[row]), CMD);
87      delay(1);
88  }
89
```

```
90  /**
91   *@brief Initialize LCD
92   **/
93  void lcd_init()
94  {
95      LCD_DIR |= (D4+D5+D6+D7+RS+EN);
96      LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
97
98      delay(150);                     // Wait for power up ( 15ms )
99      lcd_write(0x33, CMD);           // Initialization Sequence 1
100     delay(50);                      // Wait ( 4.1 ms )
101     lcd_write(0x32, CMD);           // Initialization Sequence 2
102     delay(1);                       // Wait ( 100 us )
103
104     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
105
106     lcd_write(0x28, CMD);           // 4 bit mode, 2 line
107     delay(1);
108
109     lcd_write(0x0C, CMD);           // Display ON, Cursor OFF, Blink OFF
110     delay(1);
111
112     lcd_write(0x01, CMD);           // Clear screen
113     delay(20);
114
115     lcd_write(0x06, CMD);           // Auto Increment Cursor
116     delay(1);
117
118     lcd_setCursor(0,0);             // Goto Row 1 Column 1
119 }
120
```

```
120
121 /*@brief entry point for the code*/
122 void main(void)
123 {
124     WDTCTL = WDTPW + WDTHOLD;       //! Stop Watchdog (Not recommended for code in production and devices working in field)
125
126     lcd_init();
127     lcd_setCursor(0,1);
128     lcd_print("Hello Embedded");
129     lcd_setCursor(1,5);
130     lcd_print("Systems!");
131     while(1);
132 }
```


Hello Embedded
Systems

# Custom Display on the LCD




CG-RAM is the main component in making custom characters. It stores the custom characters once declared in the code.

- CG-RAM size is 64 byte providing the option of creating eight characters at a time. Each character is eight byte in size.
- CG-RAM address starts from 0x40 (Hexadecimal) or 64 in decimal.
- We can generate custom characters at these addresses.
- Once we generate our characters at these addresses, now we can print them on the LCD at any time by just sending simple commands to the LCD as shown on the right.

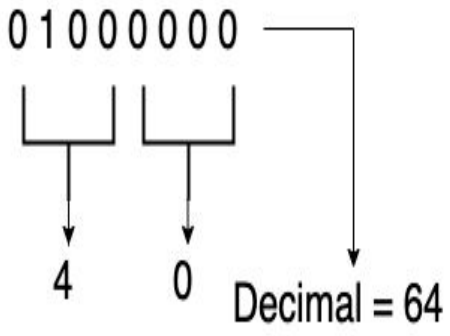
| CG-RAM Characters | CG-RAM Address (Hexadecimal) | Commands to display Generated Characters |
|---|---|---|
| 1st Character | 0x40 | 0 |
| 2nd Character | 0x48 | 1 |
| 3rd Character | 0x56 | 2 |
| 4th Character | 0x64 | 3 |
| 5th Character | 0x72 | 4 |
| 6th Character | 0x80 | 5 |
| 7th Character | 0x88 | 6 |
| 8th Character | 0x96 | 7 |

In the table above, you can see starting addresses for each character with their printing commands.

The first character is generated at address 0x40 to 0x47 and is printed on LCD by just sending simple command 0 to the LCD. The second character is generated at address 0x48 to 0x55 and is printed by sending 1 to LCD.

# Display letter ‘b’ on the LCD

Starting Address of CG-RAM

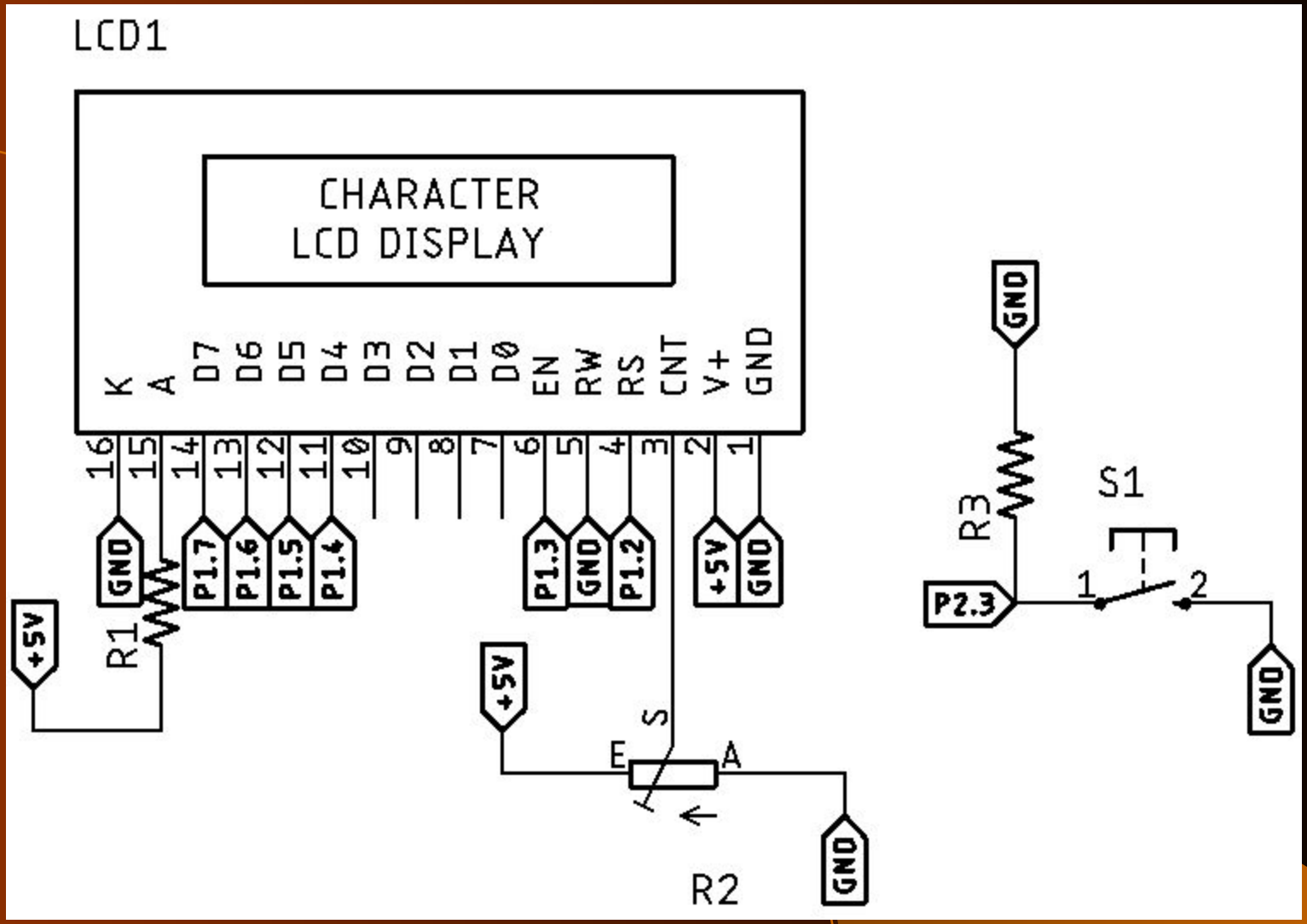



| Byte Address | C1 | C2 | C3 | C4 | C5 | | Pattern For 'b' is: |
|---|---|---|---|---|---|---|---|
| 0 0 0 0 | 1 | 0 | 0 | 0 | 0 | Row1 | 0x10 |
| 0 0 0 1 | 1 | 0 | 0 | 0 | 0 | Row2 | 0x10 |
| 0 0 1 0 | 1 | 0 | 1 | 1 | 0 | Row3 | 0x16 |
| 0 0 1 1 | 1 | 1 | 0 | 0 | 1 | Row4 | 0x19 |
| 0 1 0 0 | 1 | 0 | 0 | 0 | 1 | Row5 | 0x11 |
| 0 1 0 1 | 1 | 0 | 0 | 0 | 1 | Row6 | 0x11 |
| 0 1 1 0 | 1 | 1 | 1 | 1 | 0 | Row7 | 0x1E |
| 0 1 1 1 | 0 | 0 | 0 | 0 | 0 | Row8 | <--Cursor Position |

**Here, in the 8 bit data,**

- First 3 bits are treated as Don’t cares.
- Rest 5 bits are loaded at the address

- Send address where you want to create character. Now create your character at this address.
- Send the ‘b’ character array values defined above one by one to the data register of LCD.
- To print the generated character at 0x40.
- Send command 0 to command register of LCD.


LCD1
CHARACTER
LCD DISPLAY
K
A
D7
D6
D5
D4
D3
D2
D1
D0
EN
RW
RS
CNT
V+
GND
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
GND
P1.7
P1.6
P1.5
P1.4
P1.3
GND
P1.2
+5V
GND
+5V
R1
+5V
S
E
A
R2
GND
GND
R3
S1
P2.3
1
2
GND

# The ‘Hello LCD With Custom Character’ Code

```
#include <msp430.h>
#include <inttypes.h>

#define CMD         0
#define DATA        1

#define LCD_OUT     P1OUT
#define LCD_DIR     P1DIR
#define D4          BIT4
#define D5          BIT5
#define D6          BIT6
#define D7          BIT7
#define RS          BIT2
#define EN          BIT3

#define SW          BIT3

#define LCD_SETCGRAMADDR 0x40

//Heart character
uint8_t heart[8] = {
  0x00,
  0x0A,
  0x1F,
  0x1F,
  0x1F,
  0x0E,
  0x04,
  0x00
};

```

```
30 //
31 /**
32  *@brief Delay function for producing delay in 0.1 ms increments
33  *@param t milliseconds to be delayed
34  *@return void
35  **/
36 void delay(uint16_t t)
37 {
38     uint16_t i;
39     for(i=t; i > 0; i--)
40         __delay_cycles(100);
41 }
42
43 /**
44  *@brief Function to pulse EN pin after data is written
45  *@return void
46  **/
47 void pulseEN(void)
48 {
49     LCD_OUT |= EN;                          // Giving a falling edge at EN pin
50     delay(1);
51     LCD_OUT &= ~EN;
52     delay(1);
53 }
54
```

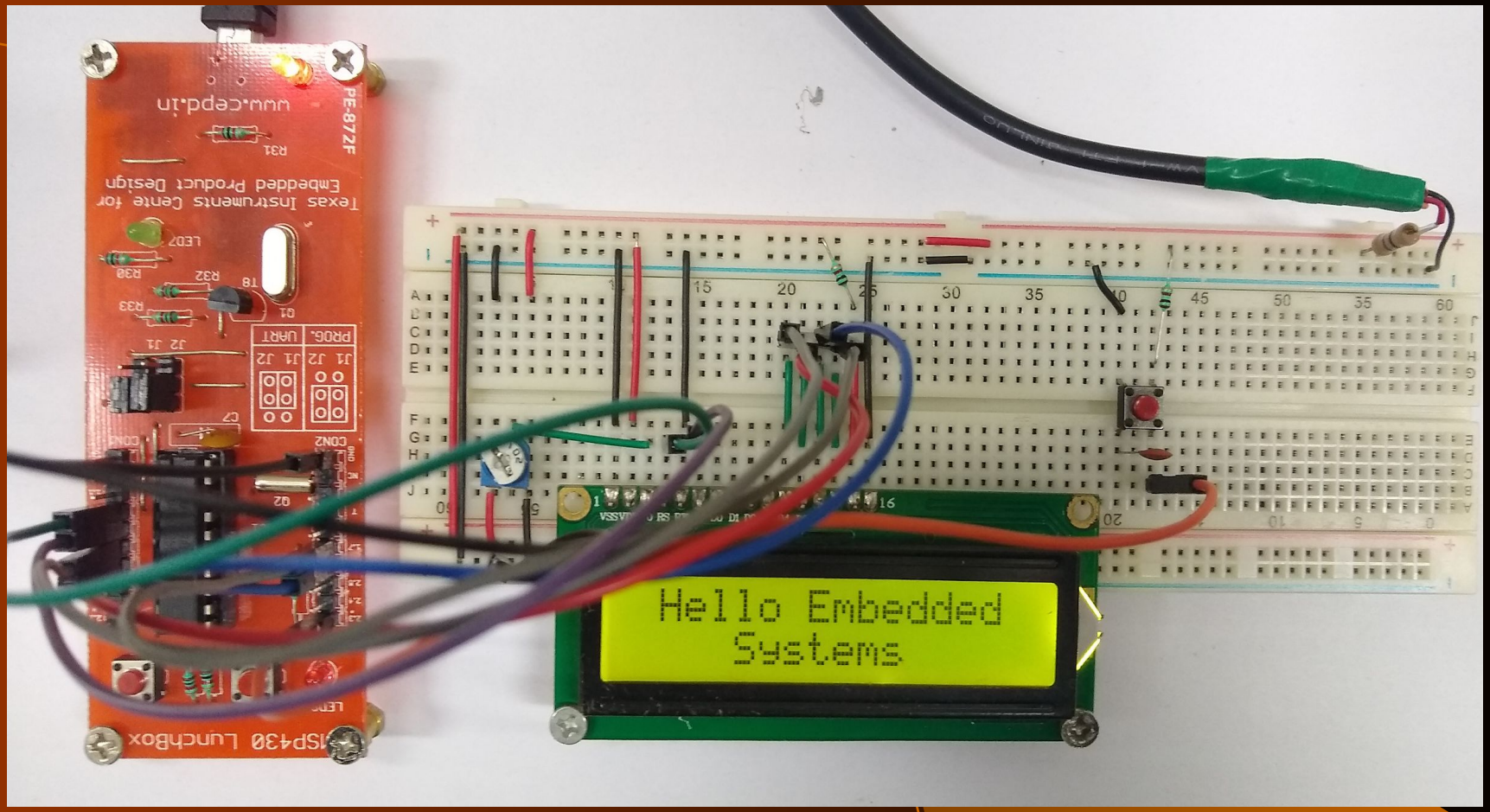
```
54
55 /**
56  *@brief Function to write data/command to LCD
57  *@param value Value to be written to LED
58  *@param mode Mode -> Command or Data
59  *@return void
60  **/
61 void lcd_write(uint8_t value, uint8_t mode)
62 {
63     if(mode == CMD)
64         LCD_OUT &= ~RS;             // Set RS -> LOW for Command mode
65     else
66         LCD_OUT |= RS;              // Set RS -> HIGH for Data mode
67
68     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));              // Write high nibble first
69     pulseEN();
70     delay(1);
71
72     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));       // Write low nibble next
73     pulseEN();
74     delay(1);
75 }
```

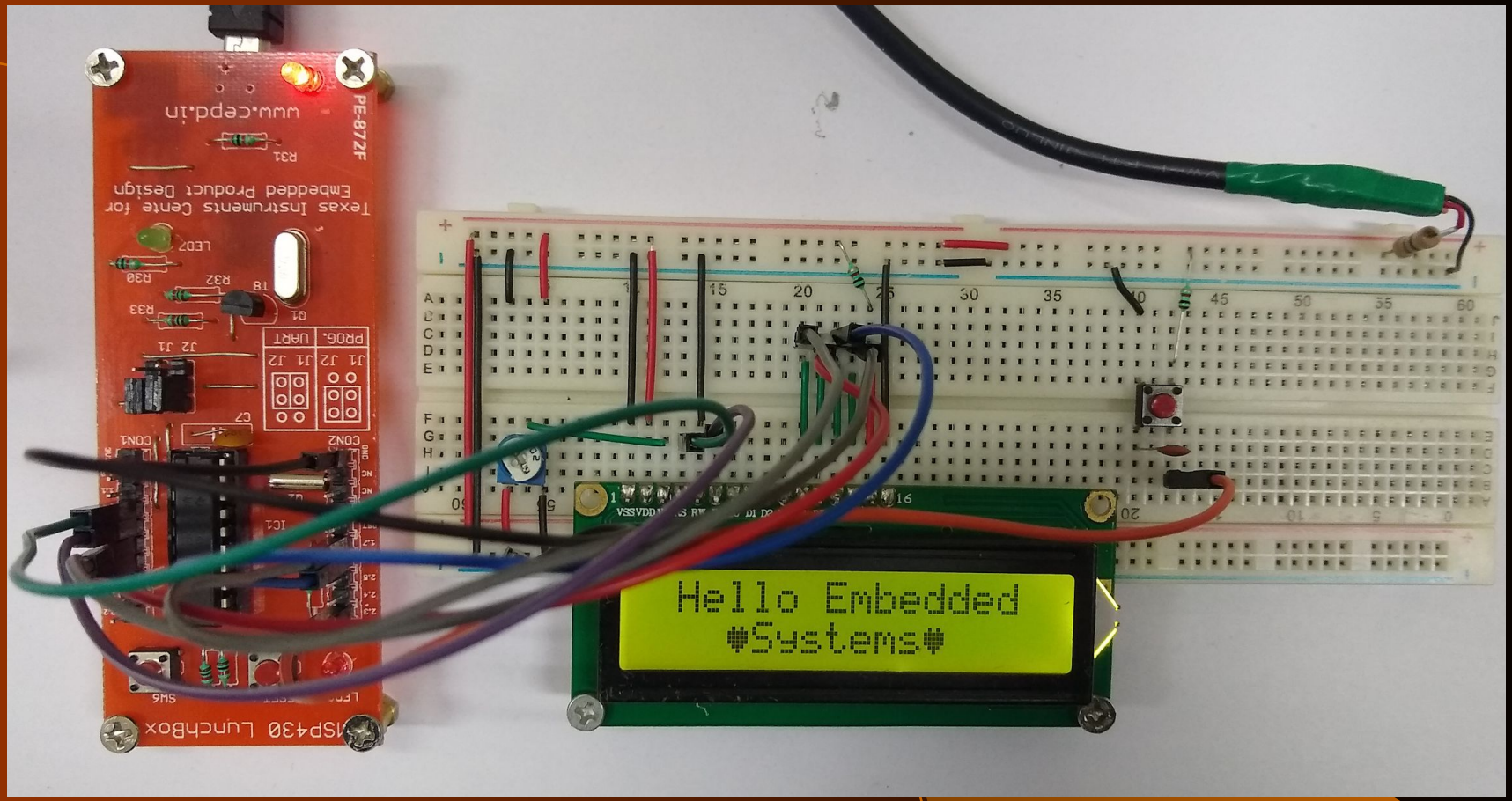
```
77 /**
78  *@brief Function to print a string on LCD
79  *@param *s pointer to the character to be written.
80  *@return void
81  **/
82 void lcd_print(char *s)
83 {
84     while(*s)
85     {
86         lcd_write(*s, DATA);
87         s++;
88     }
89 }
90
91 /**
92  *@brief Function to move cursor to desired position on LCD
93  *@param row Row Cursor of the LCD
94  *@param col Column Cursor of the LCD
95  *@return void
96  **/
97 void lcd_setCursor(uint8_t row, uint8_t col)
98 {
99     const uint8_t row_offsets[] = { 0x00, 0x40};
100    lcd_write(0x80 | (col + row_offsets[row]), CMD);
101    delay(1);
102 }
103
```

```
110 void lcd_createChar(uint8_t location, uint8_t charmap[]) {
111   location &= 0x7; // we only have 8 locations 0-7
112   lcd_write(LCD_SETCGRAMADDR | (location << 3), CMD);
113   int i = 0;
114   for (i=0; i<8; i++) {
115       lcd_write(charmap[i], DATA);
116   }
117 }
118
119 /**
120  *@brief Initialize LCD
121  **/
122 void lcd_init()
123 {
124     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
125     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
126
127     delay(150);                     // Wait for power up ( 15ms )
128     lcd_write(0x33, CMD);           // Initialization Sequence 1
129     delay(50);                      // Wait ( 4.1 ms )
130     lcd_write(0x32, CMD);           // Initialization Sequence 2
131     delay(1);                       // Wait ( 100 us )
132
133     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
134
135     lcd_write(0x28, CMD);           // 4 bit mode, 2 line
136     delay(1);
137
138     lcd_write(0x0C, CMD);           // Display ON, Cursor OFF, Blink OFF
139     delay(1);
140
141     lcd_write(0x06, CMD);           // Auto Increment Cursor
142     delay(1);
143
144     lcd_write(0x01, CMD);           // Clear screen
145     delay(20);
146
147     lcd_setCursor(0,0);             // Goto Row 1 Column 1
148 }
```

```
151
152 /*@brief entry point for the code*/
153 void main(void)
154 {
155     WDTCTL = WDTPW + WDTHOLD;       //! Stop Watchdog (Not recommended for code in production and devices working in field)
156
157     uint8_t count = 0;
158
159     P2DIR &=~ SW;
160
161     lcd_init();                             // Initialising LCD
162
163     lcd_createChar(0, heart);               // Creating Custom Character
164
165     lcd_setCursor(0,1);                     // Cursor position (0,1)
166     lcd_print("Hello Embedded");            // Print
167
168     lcd_setCursor(1,4);                     // Cursor position (1,3)
169     lcd_print("Systems");                   // Print
170
```

```
171     while(1)
172     {
173         if(!(P2IN & SW))            // If SW is Pressed
174         {
175             __delay_cycles(20000);  // Wait 20ms to debounce
176             while(!(P2IN & SW));    // Wait till SW Released
177             __delay_cycles(20000);  // Wait 20ms to debounce
178             switch(count)
179             {
180                 case 0:
181                 {
182                     lcd_setCursor(1,3);                 // Cursor position (1,3)
183                     lcd_write(0x00, DATA);              // Printing Custom Char (Heart)
184                     lcd_setCursor(1,11);                // Cursor position (1,11)
185                     lcd_write(0x00, DATA);              // Printing Custom Char (Heart)
186                     count = 1;
187                     break;
188                 }
189
190                 case 1:
191                 {
192                     lcd_setCursor(1,3);                 // Cursor position (1,3)
193                     lcd_write(0x20, DATA);              // Printing Space
194                     lcd_setCursor(1,11);                // Cursor position (1,11)
195                     lcd_write(0x20, DATA);              // Printing Space
196                     count = 0;
197                     break;
198                 }
199             }
200         }
201     }
202 }
203
```


Hello Embedded
Systems


Hello Embedded
Systems
www.cepd.in
PE-872F
Texas Instruments Cente. for
Embedded Product Design
MSP430 LunchBox

Thank you!