LCDs, Switches and Bouncing Effect, Keypads, and Relays (Chapters 14 & 17)

Updated: 4/9/18
Liquid-Crystal Display (LCD)
Read/Write (RW) connects to the ground if processor does not read data from LCD

LED+ and LED- pins provide 5V voltage for backlights.

Enable (E) pin provides clock signal

Register Select (RS)
- RS = 0: data on the data bus is a command
- RS = 1: data on the data bus is actual data.
Saves four pins for the processor
For each 8-bit data, transfer the upper 4 bits first and then the lower 4 bits.
LCD Display

- Each character has 40 pixels
- The image pattern is encoded
- Note that it is possible to change the FONT

NxK
N Characters per line
K lines
e.g., 20x2
HD44780 Controller: Internal Diagram

<table>
<thead>
<tr>
<th>DDRAM Address Register</th>
<th>Display Memory (DDRAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>(It stores all characters received.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Register</th>
<th>Table Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAM for user-defined fonts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command Register</th>
<th>ROM (Read-only Memory) for standard fonts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Character Generator (CG)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register Select (RS)</th>
<th>Binary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>111110</td>
</tr>
<tr>
<td>1</td>
<td>10001</td>
</tr>
<tr>
<td></td>
<td>10001</td>
</tr>
<tr>
<td></td>
<td>11110</td>
</tr>
<tr>
<td></td>
<td>10100</td>
</tr>
<tr>
<td></td>
<td>10010</td>
</tr>
<tr>
<td></td>
<td>10001</td>
</tr>
<tr>
<td></td>
<td>00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COM Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD Driver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEG Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD Display</td>
</tr>
</tbody>
</table>
Addressing

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
</tbody>
</table>

Display DRAM

![Diagram showing address mapping and display DRAM layout](image)
Clock Signal

On the falling edge of E (Enable), data are latched into the data or command register
- RS = 0: command; RS = 1: data

E is a data transmission clock signal provided by the processor

A nibble = 4 bits

For each byte, the upper nibble is sent out first.
Generating Clock Signal

```c
void LCD_Pulse(void) {
    LCD_Port->ODR |= 1<<LCD_EN;       // Set E high
    Delay(4);                         // Delay 40us
    LCD_Port->ODR &= ~(1<<LCD_EN);    // Set E low
    Delay(4);                         // Delay 40us
}
```

8-bit Data Bus Mode

- **E**
- **RS**
- **DB[7..0]**

Latch 8 bits

4-bit Data Bus Mode

- **E**
- **RS**
- **DB[7..4]**

Latch upper 4 bits
Latch lower 4 bits
Example Signals

Assume RW is low.

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>60</td>
<td>.</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
</tbody>
</table>

4 or 8 bit connection?
Where is the MSB (DB7-DB4)?
Data or command is being stored in the command reg?
Is the LCD in write-only or read-only mode?
What is the value we are writing to the LCD?
Example Signals

- Sending “hello”

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>60</td>
<td>' '</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
</tbody>
</table>

- E
- RS
- DB7
- DB6
- DB5
- DB4

0x68 'h'
0x65 'e'
0x6C 'l'
0x6C 'l'
0x6F 'o'
Sending Command and Data

Sending Command to LCD

```c
void LCD_SendCmd(uint8_t c) {
    // RS: 0 = command, 1 = data
    LCD_Port->ODR &= ~(1<<LCD_RS);

    // Send Upper 4 bits
    LCD_PutNibble( c >> 4 );
    LCD_Pulse();

    // Send Lower 4 bits
    LCD_PutNibble( c & 0xF );
    LCD_Pulse();

    // Return to default
    LCD_Port->ODR |= 1<<LCD_RS;
}
```
Sending Command and Data

Sending Command to LCD

```c
void LCD_SendCmd(uint8_t c) {
    // RS: 0 = command, 1 = data
    LCD_Port->ODR &= ~(1<<LCD_RS);

    // Send Upper 4 bits
    LCD_PutNibble( c >> 4 );
    LCD_Pulse();

    // Send Lower 4 bits
    LCD_PutNibble( c & 0xF );
    LCD_Pulse();

    // Return to default
    LCD_Port->ODR |= 1<<LCD_RS;
}
```

Sending Data to LCD

```c
void LCD_SendData(uint8_t c) {
    // RS defaults to 1
    // Send Upper 4 bits
    LCD_PutNibble( c >> 4 );
    LCD_Pulse();

    // Send Lower 4 bits
    LCD_PutNibble( c & 0xF );
    LCD_Pulse();
}
```
Sending Command and Data

void LCD_SendCmd(uint8_t c) {
    // RS: 0 = command, 1 = data
    LCD_Port > ODR &= ~((1 << LCD_RS));
    // Send Upper 4 bits
    LCD_PutNibble(c >> 4);
    LCD_Pulse();
    // Send Lower 4 bits
    LCD_PutNibble(c & 0xF);
    LCD_Pulse();
    // Return to default
    LCD_Port > ODR |= 1 << LCD_RS;
}

void LCD_SendData(uint8_t c) {
    // RS defaults to 1
    // Send Upper 4 bits
    LCD_PutNibble(c >> 4);
    LCD_Pulse();
    // Send Lower 4 bits
    LCD_PutNibble(c & 0xF);
    LCD_Pulse();
}
## Command Format

<table>
<thead>
<tr>
<th>LCD Commands</th>
<th>RS</th>
<th>RW</th>
<th>DB7</th>
<th>DB6</th>
<th>DB5</th>
<th>DB4</th>
<th>DB3</th>
<th>DB2</th>
<th>DB1</th>
<th>DB0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Clear screen and set DDRAM address to 0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Return home (set DDRAM address to 0)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>I/D S</td>
<td>Entry mode set</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>Control on/off of display, cursor, and blink</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SC</td>
<td>RL</td>
<td>-</td>
<td>-</td>
<td>Cursor/display shift</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DL</td>
<td>N</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>Set of data length, lines, and font type</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A5</td>
<td>A4</td>
<td>A3</td>
<td>A2</td>
<td>A1</td>
<td>A0</td>
<td></td>
<td>Set code generator CG-RAM address A5-A0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>A6</td>
<td>A5</td>
<td>A4</td>
<td>A3</td>
<td>A2</td>
<td>A1</td>
<td>A0</td>
<td></td>
<td>Set display memory DDRAM address A6-A0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>BF</td>
<td>A6</td>
<td>A5</td>
<td>A4</td>
<td>A3</td>
<td>A2</td>
<td>A1</td>
<td>A0</td>
<td></td>
<td>Ready busy flag (BF) and address A6-A0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>Write data D7-D0 to DDRAM or CG-RAM</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>Read data D7-D0 from DDRAM or CG-RAM</td>
</tr>
</tbody>
</table>
## Command Format

<table>
<thead>
<tr>
<th>LCD Commands</th>
<th>CG-RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>RW</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Command Examples:

- **0x01 → 0000 0001 → Clear screen**: (assume RS & RW are LOW)
- **0x06 → 0000 0110 → I=1, Increment ; S=1, shift one character**
- **0x40 → Set the CG RAM position address 0x0**

See LCD Setup Document on the web page…..
LCD Initialization

1. Wait for $\geq 30 \text{ ms}$
2. Send command 0x30
3. Wait for $\geq 4.1 \text{ ms}$
4. Send command 0x30
5. Wait for $\geq 160 \mu\text{s}$
6. Send command 0x30
7. Wait for $\geq 160 \mu\text{s}$

8-bit data bus mode

8a. Send command 0b0011NF**
8b. Wait for $\geq 40 \mu\text{s}$

Does data bus have 8 bits?

Yes

4-bit data bus mode

8a. Send command 0x2 to switch to 4-bit mode
8b. Wait for $\geq 40 \mu\text{s}$
8c. Send command 0b0010NF**
8d. Wait for $\geq 40 \mu\text{s}$

No

9. Send command 0x01 to clear display
10. Wait for $\geq 1.5 \text{ ms}$
11. Send command 0x06 to set entry mode
12. Wait for $\geq 40 \mu\text{s}$
13. Send command 0x0F to control display ON/OFF
14. Wait for $\geq 40 \mu\text{s}$

Initialization completes
Customized Font

// Select CGRAM and set address to 0x00
LCD_SendCmd(0x40 + 0x00);
Delay(4); // Wait > 39us

// Define smile face
LCD_SendData(0x00); // 1st row byte
LCD_SendData(0x0A); // 2nd row byte
LCD_SendData(0x0A); // 3rd row byte
LCD_SendData(0x0A); // 4th row byte
LCD_SendData(0x00); // 5th row byte
LCD_SendData(0x11); // 6th row byte
LCD_SendData(0x0E); // 7th row byte
LCD_SendData(0x00); // 8th row byte

// Select display RAM & set address to 0
LCD_SendCmd(0x80); // First character
Delay(4); // Wait > 39us

LCD_SendData(0x00); // Display the font
Interfacing Push-Button Keys (1 of 2)

- Electrical connection of a push-button key is same as that of a DIP switch except that the connection is temporary when the key is pressed.
  - When a key is pressed (or released), mechanical metal contact bounces momentarily and can be read as multiple inputs.
  - The reading of one contact as multiple inputs can be eliminated by a key-debounce technique, using either hardware or software.
Interfacing Push-Button Keys (2 of 2)
Key Debounce Techniques

- **Hardware technique**
  - Two NAND gates connected back to back, equivalent of a S-R latch. The output of the S-R latch is a pulse without a bounce.
  - An integrated circuit (MAX 6816) that bounces the key internally and provides a steady output.
  - Use an RC circuit to slow down the rise time.
Slew Rate

Slew Rate: Maximum rate of change of the output voltage

\[ Slew \ Rate = \max \left( \frac{\Delta V}{\Delta t} \right) \]

A high slew rate allows the output to be toggled at a fast speed.
Problem statement

- A bank of push-button keys are connected as inputs to PORTB.
- The pull-up resistors are internal to PORTB.
- Write a program to recognize a key pressed, debounce the key, and identify its location in the key bank with numbers from 0 to 7.
Interfacing Push-Button Keys (3 of 6)

- **Hardware**
  - PORTB should be set up as input port
  - Internal pull-up resistors should be enabled

- **Software**
  - Checking a key closure
  - Debouncing the key
  - Encoding the key

Alternatively
Interfacing Push-Button Keys - Software Debounding

- Checking a key closure
  - When a key is **open**, the logic level is **one** (assuming pull-ups are enabled) and when it is **closed**, the logic level is **zero**.
  - When all keys are open, the reading will be 0xFF, and when a key is closed, the reading will be less than 0xFF.
    - Therefore, any reading less than FFH indicates a key closure.
    - This will be the first read!

- Debouncing the key
  - Software technique
    - Wait for 20 ms.
    - Read the port again.
    - If the reading is still less than FFH, it indicates that a key is pressed.

- Encoding the key
  - Key closure can be identified by rotating the reading right and looking for ‘No Carry’ and counting the rotations
Basic Structure of an I/O Port Bit:

**Input**

- **Input Data Register (IDR):**
  - To on-chip peripheral
  - Alternate function input

- **Input trigger:**
  - Schmitt trigger
  - Reduce noise
  - Increase slew rate

- **GPIO Pull-up/Pull-down Register (PUPDR):**
  - 00 = No pull-up, pull-down
  - 01 = Pull-up
  - 10 = Pull-down
  - 11 = Reserved
Schmitt Trigger

Analog signal can be Noisy with small slew rate

$V_{TH}$

$V_{TL}$

$V_{DD}$

Rapid Changes!

Immunity Band

Schmitt Trigger

Simple Comparator
GPIO Output Type Register (OTYPE)

- 16 bits reserved, 16 data bits, 1 bit for each pin

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>OT15</td>
<td>OT14</td>
<td>OT13</td>
<td>OT12</td>
<td>OT11</td>
<td>OT10</td>
<td>OT9</td>
<td>OT8</td>
<td>OT7</td>
<td>OT6</td>
<td>OT5</td>
<td>OT4</td>
<td>OT3</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 OTy: Port x configuration bits (y = 0..15)
- These bits are written by software to configure the I/O output type.
- 0: Output push-pull (reset state)
- 1: Output open-drain

GPIOB->OTYPE &= ~(1UL<<2);  // Clear bit 2
A digital input can have three states: High, Low, and High-Impedance (also called floating, tri-stated, HiZ).

- **Pull-Up**
  If external input is HiZ, the input is read as a valid HIGH.

- **Pull-Down**
  If external input is HiZ, the input is read as a valid LOW.
## GPIO Pull-up/Pull-down Register (PUPDR)

- 16 pins per port, 2 bits per pin

### Table: GPIO Pull-up/Pull-down Register (PUPDR)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits $2y+1:2y$ PUPD$y[1:0]$: Port x configuration bits ($y = 0..15$)

These bits are written by software to configure the I/O pull-up or pull-down

- 00: No pull-up, pull-down
- 01: Pull-up
- 10: Pull-down
- 11: Reserved

```c
// No pull-up, pull-down
GPIOA->PUPDR &= ~3UL;
```
Various Relays

**Mechanical relays** are devices that can turn on or turn off the power supplied to another device, like a switch.

Relays can be normally OPEN
Or normally CLOSED!

Various Relays

**Mechanical relays** are devices that can turn on or turn off the power supplied to another device, like a switch.

Relays can be normally OPEN or normally CLOSED!
Drivers

- Controlling appliances
- Driving solenoid

Relays generally need high current to be activated
Calculating the Current Requirement

- Transistor inverter acting as the driver
- Assume LED requires 3.5 Volts and 20 mA
- 2N2222 is Current Amplifier → generates collector current (multiplies base-current)
- In this case the current gain is 100!
- Note Vbe=0.7
Keypads

- 2pcs Matrix Array 4x4 16 Keys Membrane Switch 8 pins connector Switch Keypad

**Description:**

Extended keyboard for development board, can connect easily with cable.

**Maximum circuit rating:** 35V DC, 100mA.

Insulation resistance: 100M Ohm, 100V.

Dielectric withstand: 250VRms, 50 - 60Hz, 1 min.

**Contact bounce:** 5ms.

**Life expectancy:** 1 million closures.

Operating temperature: -20 to +40 °C.

Operating humidity: 40° C, 90% ~ 95%.

Connector type: 8 pin.

Mount type: adhesive.

Cable length (inc connector): 88 mm.

Size: 77*69 mm

Package Content:

2Pcs Matrix Array 4x4 16 Keys Membrane Switch 8 pins connector Switch Keypad for Arduino/AVR/PIC
Keypad Scan
Keypad Scan

Step 1: Set Output
R1, R2, R3, R4 = 0000

Step 2: Read Input
C1, C2, C3 = 111

⇒ No key pressed
Keypad Scan

Key “0” is pressed
Keypad Scan

Step 1: Set Output
R1,R2,R3,R4 = 0000

Step 2: Read Input
C1,C2,C3 = 101

⇒ Some key in 2\textsuperscript{nd} column is pressed down
Keypad Scan

Step 1: Set Output
R1, R2, R3, R4 = 0000

Step 2: Read Input
C1, C2, C3 = 101

→ Step 3a: Scan 1st row
R1, R2, R3, R4 = 0111
C1, C2, C3 = 111

⇒ No key in 1st row is pressed down

See the textbook to see the code.
Scan 2nd row

Step 1: Set Output
R1,R2,R3,R4 = 0000

Step 2: Read Input
C1,C2,C3 = 101

→ Step 3b: Scan 2nd row
R1,R2,R3,R4 = 1011
C1,C2,C3 = 111

⇒ No key in 2nd row is pressed down
Step 1: Set Output
R1,R2,R3,R4 = 0000

Step 2: Read Input
C1,C2,C3 = 101

→ Step 3c: Scan 3rd row
R1,R2,R3,R4 = 1101
C1,C2,C3 = 111

⇒ No key in 3rd row is pressed down
**Scan 4th row**

**Step 1:** Set Output  
R1, R2, R3, R4 = 0000

**Step 2:** Read Input  
C1, C2, C3 = 101

→ **Step 3d:** Scan 4th row  
R1, R2, R3, R4 = 1110  
C1, C2, C3 = 101

⇒ key in 4th row is pressed down
Keypad Scan

Key pressed is located at the second column and the fourth row.