Microcontrollers

Tecnológico de Monterrey

Professor: Dr. Gilberto Ochoa Ruiz

Topic 3: ARM Basic Programming Techniques Part 2

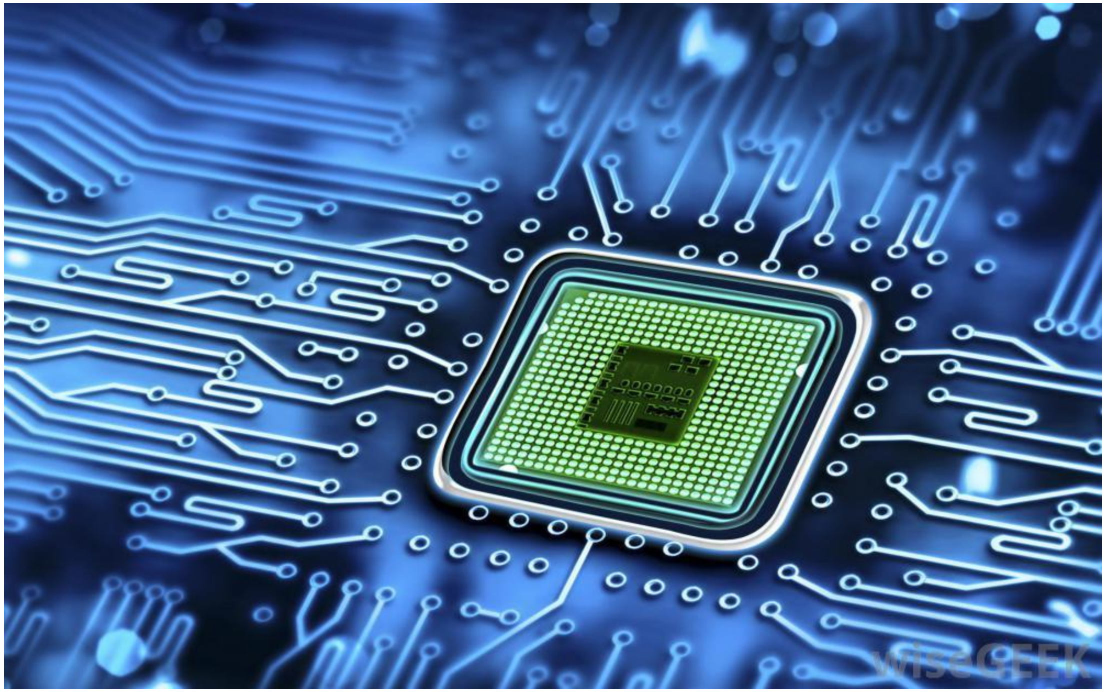
EIAD-204

Wednesday

March 25th 2020

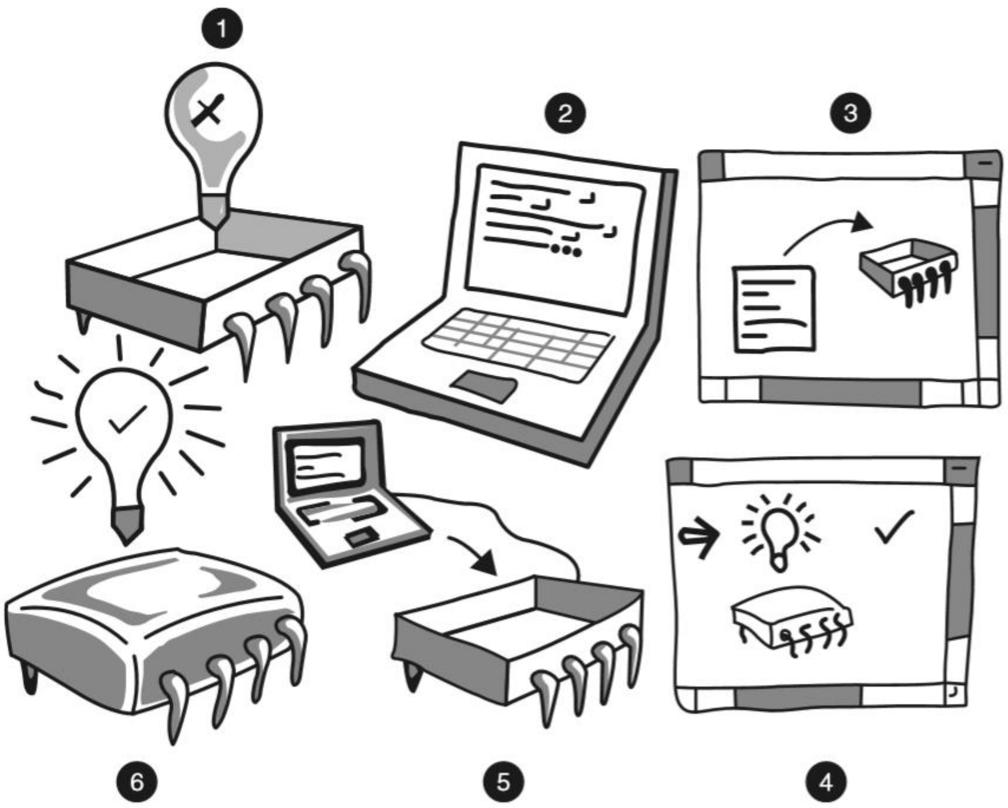
6h30 – 9h30 PM

Guadalajara, Mexico



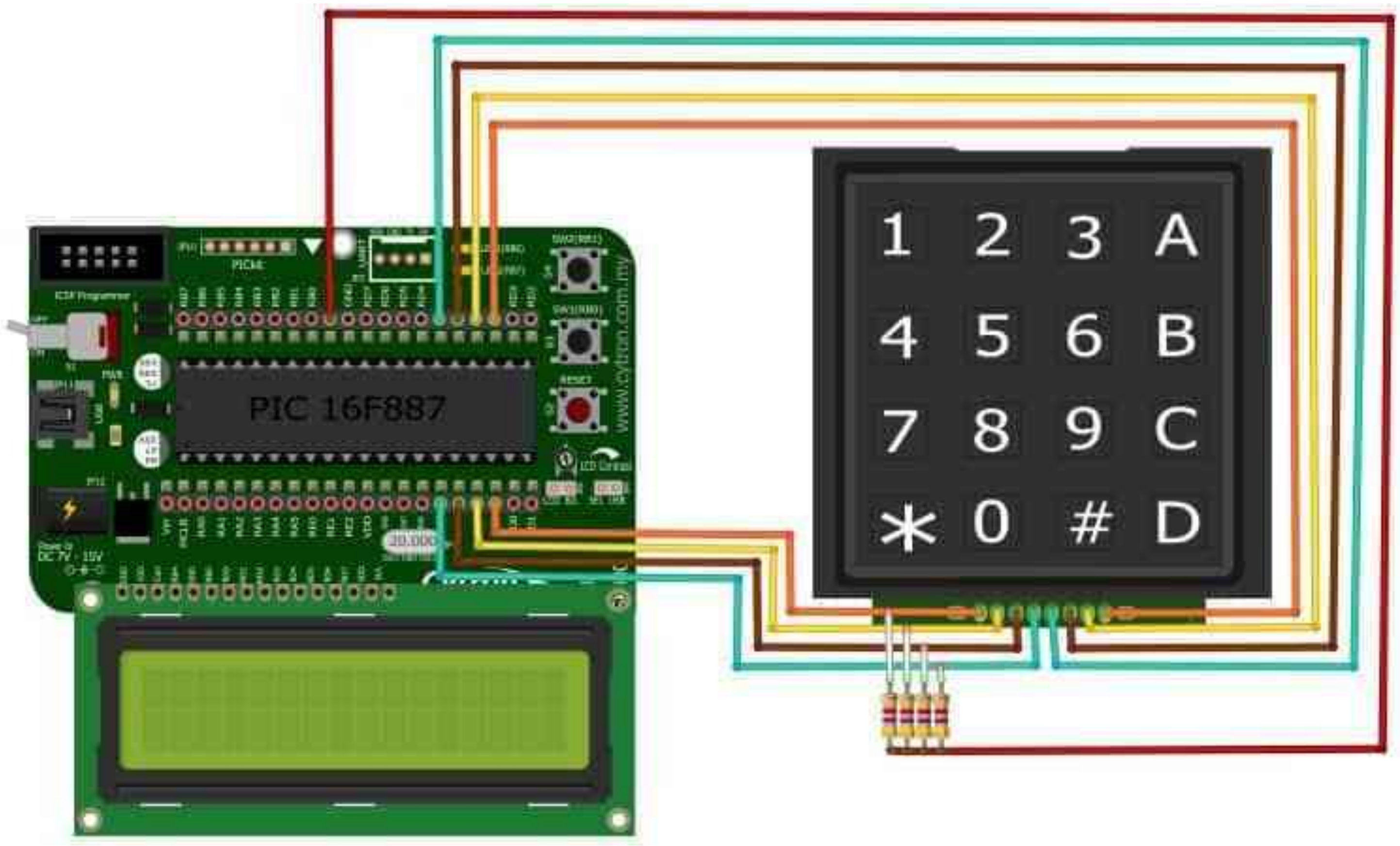
Outline

- Interfacing to an LCD
- Sending Commands and Data to an LCD
- LCD 4-bit option
- Interfacing the Keyboard to the Microcontroller
- Key press detection and ID



Basic Programming Techniques Part 2

LCD and Keyboard Interfacing



Basic Programming Techniques Part 2

Interfacing to an LCD

This section describes the operation modes of the LCDs, then describes how to program and interface an LCD to the Freescale FRDM board.

LCD operation

In recent years the LCD is replacing LEDs (seven-segment LEDs or other multi-segment LEDs). This is due to the following reasons:

Basic Programming Techniques Part 2

1. The declining prices of LCDs.

2. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.

3. Incorporation of the refreshing controller into the LCD itself, thereby relieving the CPU of the task of refreshing the LCD.

4. Ease of programming for both characters and graphics.

5. The extremely low power consumption of LCD (when backlight is not used).

LCD module pin descriptions

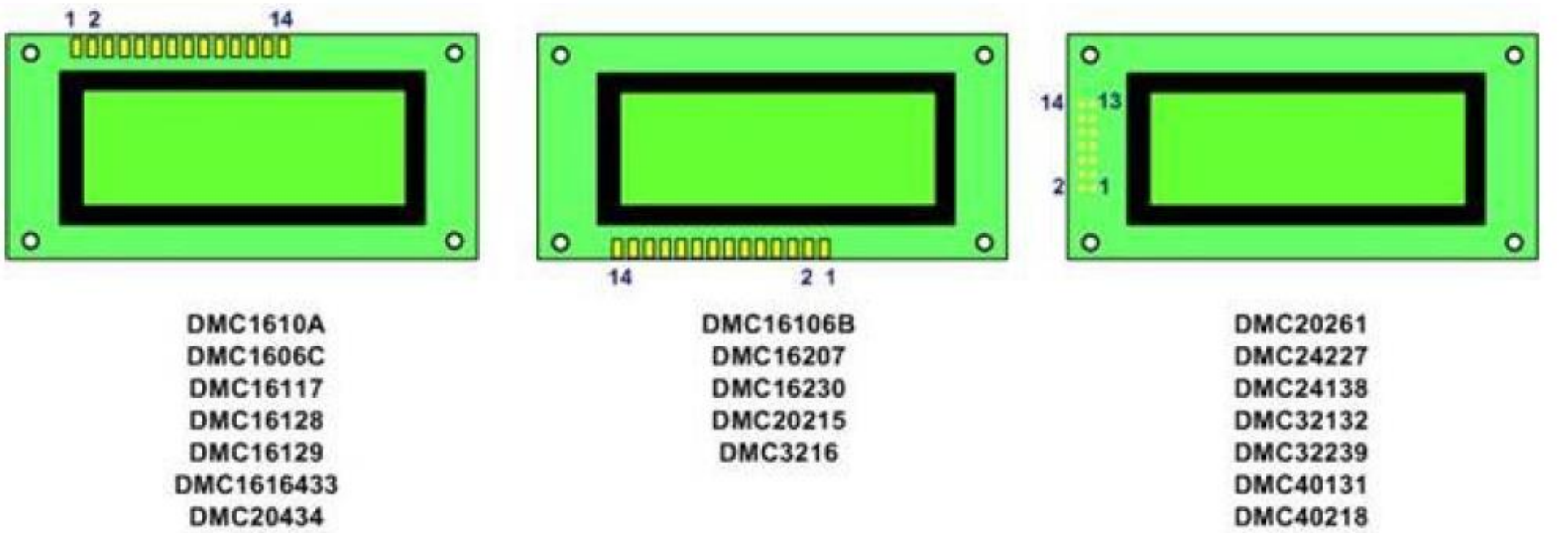
For many years, the use of Hitachi HD44780 LCD controller dominated the character LCD modules.

Even today, most of the character LCD modules still use HD44780 or a variation of it.

The HD44780 controller has a 14 pin interface for the microprocessor.

We will discuss this 14 pin interface in this section.

LCD module pin descriptions



LCD module pin descriptions

Pin	Symbol	I/O	Description
1	VSS	—	Ground
2	VCC	—	+5V power supply
3	VEE	—	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I	Enable
7	DB0	I/O	The 8-bit data bus

LCD module pin descriptions

Pin	Symbol	I/O	Description
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 4/8-bit data bus
12	DB5	I/O	The 4/8-bit data bus
13	DB6	I/O	The 4/8-bit data bus
14	DB7	I/O	The 4/8-bit data bus

LCD module pin descriptions

VEE: VEE is used for controlling the LCD contrast.

RS, register select: There are two registers inside the LCD and the RS pin is used for their selection as follows. If RS = 0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, and so on. If RS = 1, the data register is selected, allowing the user to send data to be displayed on the LCD (or to retrieve data from the LCD controller).

R/W, read/write: R/W input allows the user to write information into the LCD controller or read information from it. R/W = 1 when reading and R/W = 0 when writing.

LCD module pin descriptions

E, enable: The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins

D0–D7: The 8-bit data pins are used to send information to the LCD or read the contents of the LCD's internal registers. The LCD controller is capable of operating with 4-bit data and only D4-D7 are used. We will discuss this in more details later.

To display letters and numbers, we send ASCII codes for the letters A–Z, a–z, numbers 0–9, and the punctuation marks to these pins while making RS = 1

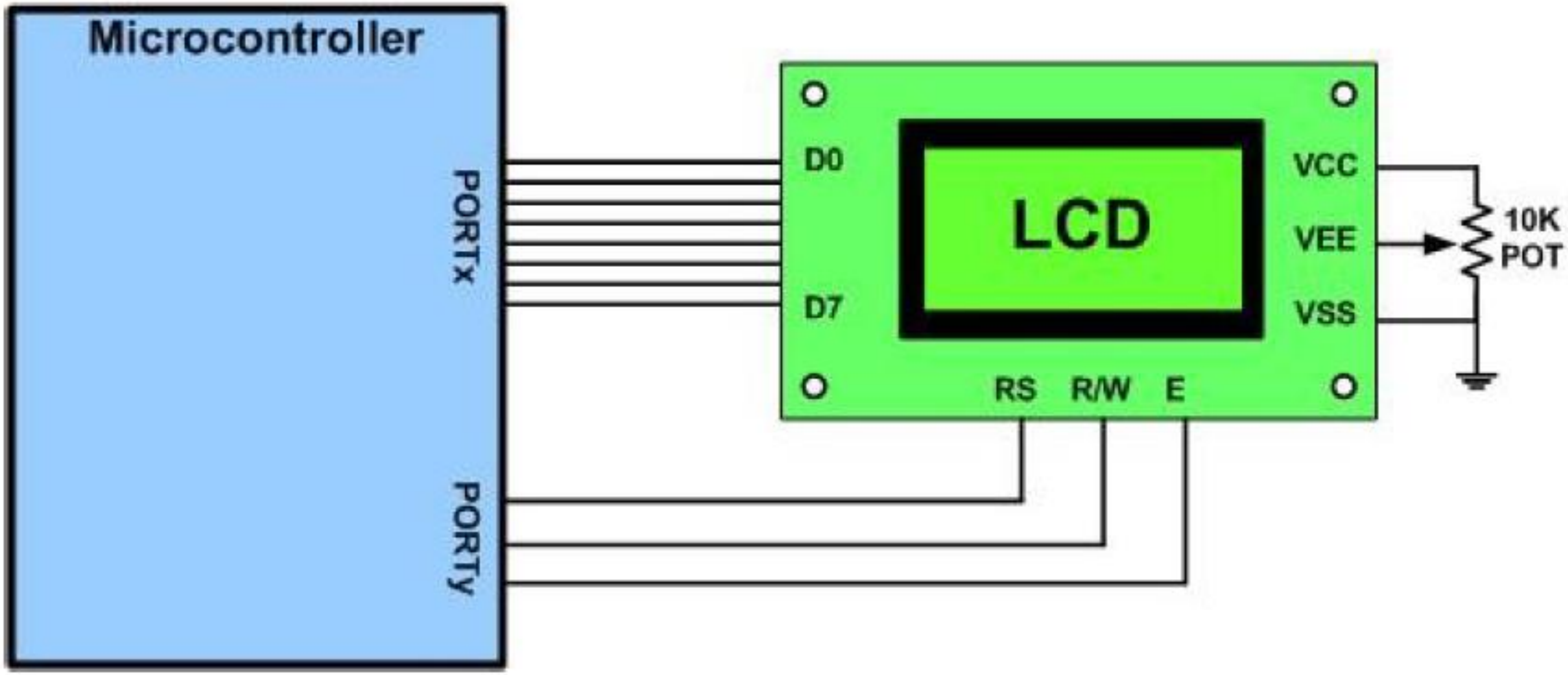
Basic Programming Techniques Part 2

There are also instruction command codes that can be sent to the LCD in order to clear the display, force the cursor to the home position, or blink the cursor

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return cursor home
6	Increment cursor (shift cursor to right)
F	Display on, cursor blinking
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 character (8-bit data, D0 to D7)
28	2 lines and 5x7 character (4-bit data, D4 to D7)

Sending commands to LCDs

To send any of the commands to the LCD, make pins RS = 0, R/W = 0, and send a pulse (L-to-H-to-L) on the E pin to enable the internal latch of the LCD.



Basic Programming Techniques Part 2

Notice the following for the connection in previous slide

1. The LCD’s data pins are connected to PORTD of the microcontroller.

2. The LCD’s RS pin is connected to Pin 2 of PORTA of the microcontroller.

3. The LCD’s R/W pin is connected to Pin 4 of PORTA of the microcontroller.

4. The LCD’s E pin is connected to Pin 5 of PORTA of the microcontroller. 5. Both Ports D and A are configured as output ports.

Basic Programming Techniques Part 2

Sending data to the LCD

In order to send data to the LCD to be displayed, we must set pins RS = 1, R/W = 0, and also send a pulse (L-to-H-to-L) to the E pin to enable the internal latch of the LCD.

Because of the extremely low power feature of the LCD controller, it runs much slower than the microcontroller

After one command or data is written to the LCD controller, one must wait until the LCD controller is ready before issuing the next command/data

An easy way (not as efficient though) is to delay the microcontroller for the previous command.

Sending data to the LCD

```
/*  Initialize and display "Hello" on the LCD using 8-bit
data mode.

* Data pins use Port D, control pins use Port A.
* This program does not poll the status of the LCD.
* It uses delay to wait out the time LCD controller is
busy.

* Timing is more relax than the HD44780 datasheet to
accommodate the
* variations among the LCD modules.

* You may want to adjust the amount of delay for your LCD
controller. */
```

Sending data to the LCD

```
#include <MKL25Z4.H>

#define RS 0x04 /* PTA2 mask */
#define RW 0x10 /* PTA4 mask */
#define EN 0x20 /* PTA5 mask */

void delayMs(int n);

void LCD_command(unsigned char command);

void LCD_data(unsigned char data);

void LCD_init(void);
```

Sending data to the LCD

```
int main(void)
{
LCD_init();
for(;;)
{
LCD_command(1); /* clear display */
delayMs(500);
LCD_command(0x80); /* set cursor at first line */
LCD_data('H'); /* write the word */
LCD_data('e');
LCD_data('l');
LCD_data('l');
LCD_data('o');
delayMs(500);
}
}
```

Sending data to the LCD

```
void LCD_init(void)
{
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[0] = 0x100; /* make PTD0 pin as GPIO */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
……

PORTD->PCR[6] = 0x100; /* make PTD6 pin as GPIO */
PORTD->PCR[7] = 0x100; /* make PTD7 pin as GPIO */
PTD->PDDR = 0xFF; /* make PTD7-0 as output pins */
SIM->SCGC5 |= 0x0200; /* enable clock to Port A */
PORTA->PCR[2] = 0x100; /* make PTA2 pin as GPIO */
PORTA->PCR[4] = 0x100; /* make PTA4 pin as GPIO */
PORTA->PCR[5] = 0x100; /* make PTA5 pin as GPIO */
PTA->PDDR |= 0x34; /* make PTA5, 4, 2 as output pins */
```

Sending data to the LCD

```
delayMs(30); /* initialization sequence */

LCD_command(0x30);
delayMs(10);
LCD_command(0x30);
delayMs(1);

LCD_command(0x30);

LCD_command(0x38); /* set 8-bit data, 2-line, 5x7 font */
LCD_command(0x06); /* move cursor right */
LCD_command(0x01); /* clear screen, move cursor to home */
LCD_command(0x0F); /* turn on display, cursor blinking */
}
```

Basic Programming Techniques Part 2

Sending data to the LCD

```
void LCD_command(unsigned char command)
{

PTA->PCOR = RS | RW; /* RS = 0, R/W = 0 */
PTD->PDOR = command;
PTA->PSOR = EN; /* pulse E */

delayMs(0);

PTA->PCOR = EN;

if (command < 4)
delayMs(4); /* command 1 and 2 needs up to 1.64ms */
else
delayMs(1); /* all others 40 us */
}
```

Sending data to the LCD

```
void LCD_data(unsigned char data)
{

PTA->PSOR = RS; /* RS = 1, R/W = 0 */

PTA->PCOR = RW;

PTD->PDOR = data;

PTA->PSOR = EN; /* pulse E */

delayMs(0);
PTA->PCOR = EN;
delayMs(1);
}
```

Sending data to the LCD

```
/* Delay n milliseconds

* The CPU core clock is set to MCGFLLCLK at 41.94
MHz in SystemInit().
*/

void delayMs(int n) {

int i;
int j;
for(i = 0 ; i < n; i++)
for(j = 0 ; j < 7000; j++) { }

}
```

Checking the LCD busy flag

The above programs used a time delay before issuing the next data or command.

This allows the LCD a sufficient amount of time to get ready to accept the next data.

However, the LCD has a busy flag. We can monitor the busy flag and issue data when it is ready. This will speed up the process.

To check the busy flag, we must read the command register (R/W = 1, RS = 0).

Checking the LCD busy flag

The busy flag is the D7 bit of that register. Therefore, if R/W = 1, RS = 0.

When D7 = 1 (busy flag = 1), the LCD is busy taking care of internal operations and will not accept any new information.

When D7 = 0, the LCD is ready to receive new information.

Doing so requires switching the direction of the port connected to the data bus to input mode when polling the status register then switch the port direction back to output mode to send the next command.

Checking the LCD busy flag

```
/* Initialize and display “hello” on the LCD using 8-
bit data mode.

* Data pins use Port D, control pins use Port A.

* Polling of the busy bit of the LCD status bit is
used for timing. */

#include <MKL25Z4.H>

#define RS 0x04 /* PTA2 mask */
#define RW 0x10 /* PTA4 mask */
#define EN 0x20 /* PTA5 mask */
```

Checking the LCD busy flag

```
#define RS 0x04 /* PTA2 mask */
#define RW 0x10 /* PTA4 mask */
#define EN 0x20 /* PTA5 mask */

void delayMs(int n);

void LCD_command(unsigned char command);

void LCD_command_noWait(unsigned char command);

void LCD_data(unsigned char data);

void LCD_init(void);

void LCD_ready(void);
```

Checking the LCD busy flag

```
int main(void)
{
LCD_init();
for(;;)
{
LCD_command(1); /* clear display */
delayMs(500);
LCD_command(0xC0); /* set cursor at 2nd line */
LCD_data('h'); /* write the word on LCD */
LCD_data('e');
LCD_data('l');
LCD_data('l');
LCD_data('o');
delayMs(500);
}
}
```

Checking the LCD busy flag

```
void LCD_init(void)
{
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[0] = 0x100; /* make PTD0 pin as GPIO */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
......
PORTD->PCR[5] = 0x100; /* make PTD5 pin as GPIO */
PORTD->PCR[6] = 0x100; /* make PTD6 pin as GPIO */
PORTD->PCR[7] = 0x100; /* make PTD7 pin as GPIO */
PTD->PDDR = 0xFF; /* make PTD7-0 as output pins */
SIM->SCGC5 |= 0x0200; /* enable clock to Port A */
PORTA->PCR[2] = 0x100; /* make PTA2 pin as GPIO */
PORTA->PCR[4] = 0x100; /* make PTA4 pin as GPIO */
PORTA->PCR[5] = 0x100; /* make PTA5 pin as GPIO */
PTA->PDDR |= 0x34; /* make PTA5, 4, 2 as output pins */
```

Checking the LCD busy flag

```
delayMs(20); /* initialization sequence */
LCD_command_noWait(0x30); /* LCD does not respond
to status poll */
delayMs(5);
LCD_command_noWait(0x30);
delayMs(1);
LCD_command_noWait(0x30);
LCD_command(0x38); /* set 8-bit data, 2-line, 5x7
font */
LCD_command(0x06); /* move cursor right */
LCD_command(0x01); /* clear screen, move cursor to
home */
LCD_command(0x0F); /* turn on display, cursor
blinking */
}
```

Checking the LCD busy flag

```
/* This function waits until LCD controller is ready to
* accept a new command/data before returns. */

void LCD_ready(void)
{
char status;
PTD->PDDR = 0; /* PortD input */
PTA->PCOR = RS; /* RS = 0 for status */
PTA->PSOR = RW; /* R/W = 1, LCD output */
do { /* stay in the loop until it is not busy */
PTA->PSOR = EN; /* raise E */
delayMs(0);
status = PTD->PDIR; /* read status register */
PTA->PCOR = EN;
```

Checking the LCD busy flag

```
delayMs(0); /* clear E */

} while (status & 0x80); /* check busy bit */
PTA->PCOR = RW; /* R/W = 0, LCD input */
PTD->PDDR = 0xFF; /* PortD output */
}

void LCD_command(unsigned char command)
{
LCD_ready(); /* wait until LCD is ready */
PTA->PCOR = RS | RW; /* RS = 0, R/W = 0 */
PTD->PDOR = command;
PTA->PSOR = EN; /* pulse E */
delayMs(0);
PTA->PCOR = EN;
}
```

Checking the LCD busy flag

```
void LCD_command_noWait(unsigned char command){
PTA->PCOR = RS | RW; /* RS = 0, R/W = 0 */
PTD->PDOR = command;
PTA->PSOR = EN; /* pulse E */
delayMs(0);
PTA->PCOR = EN; }

void LCD_data(unsigned char data)
{ LCD_ready(); /* wait until LCD is ready */
PTA->PSOR = RS; /* RS = 1, R/W = 0 */
PTA->PCOR = RW;
PTD->PDOR = data;
PTA->PSOR = EN; /* pulse E */
delayMs(0);
PTA->PCOR = EN;
}
```

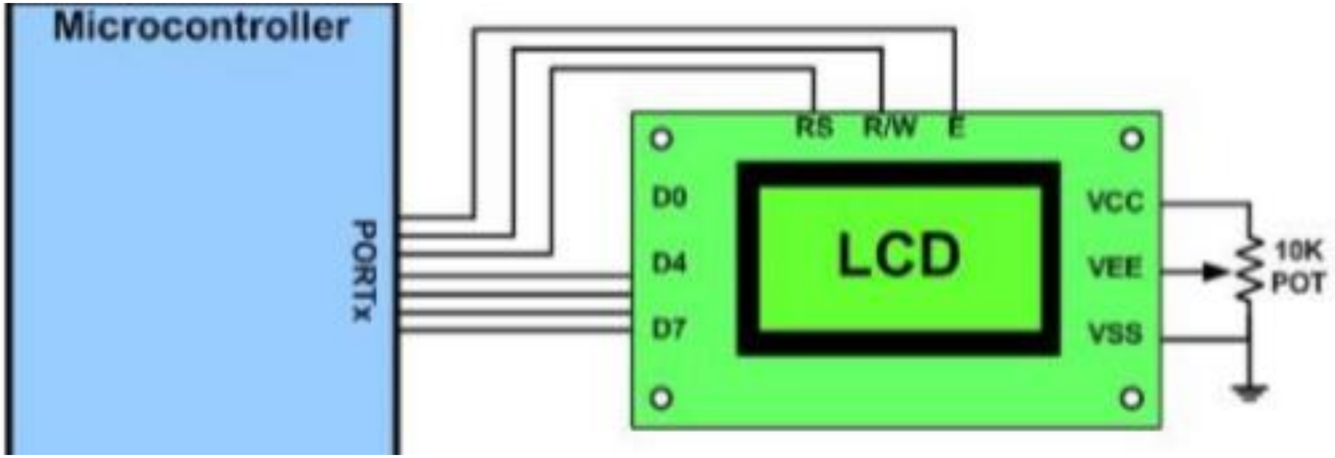
Basic Programming Techniques Part 2

LCD 4-bit Option

To save the number of microcontroller pins used by LCD interfacing, we can use the 4-bit data option

We only need to connect D7-D4 to microcontroller.

Together with the 3 control lines, the interface between the MCU and the LCD will fit in a single 8-bit port.



LCD 4-bit Option

With 4-bit data option, the microcontroller needs to issue commands to put the LCD controller in 4-bit mode during initialization.

This is done with command 0x20

After that, every command or data needs to be broken down to two 4-bit operations, upper nibble first.

In the following program, the upper nibble is extracted using command & 0xF0 and the lower nibble is shifted into place by command << 4.

LCD 4-bit Option

```
/* Initialize and display "hello" on the LCD using 4-
bit data mode.
* All interface uses Port D. Bit 7-4 are used for
data.
* Bit 4, 2, 1 are used for control.
* This program does not poll the status of the LCD.
* It uses delay to wait out the time LCD controller
is busy.
* Timing is more relax than the HD44780 datasheet to
accommodate the
* variations of the devices.
* You may want to adjust the amount of delay for your
LCD controller.
*/
#include <MKL25Z4.H>
```

LCD 4-bit Option

```
#include <MKL25Z4.H>

#define RS 1 /* BIT0 mask */
#define RW 2 /* BIT1 mask */
#define EN 4 /* BIT2 mask */

void delayMs(int n);
void delayUs(int n);
void LCD_nibble_write(unsigned char data,
unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
```

LCD 4-bit Option

```
int main(void)
{
LCD_init();
for(;;)
{
LCD_command(1); /* clear display */

delayMs(500);
LCD_command(0x85); /* set cursor at first line */
LCD_data('h'); /* write the word */
LCD_data('e');
LCD_data('l');
LCD_data('l');
LCD_data('o');
delayMs(500);
}
}
```

LCD 4-bit Option

```
void LCD_init(void)
{
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */

PORTD->PCR[0] = 0x100; /* make PTD0 pin as GPIO */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PORTD->PCR[2] = 0x100; /* make PTD2 pin as GPIO */
PORTD->PCR[4] = 0x100; /* make PTD4 pin as GPIO */
PORTD->PCR[5] = 0x100; /* make PTD5 pin as GPIO */
PORTD->PCR[6] = 0x100; /* make PTD6 pin as GPIO */
PORTD->PCR[7] = 0x100; /* make PTD7 pin as GPIO */

PTD->PDDR |= 0xF7; /* make PTD7-4, 2, 1, 0 as
output pins */
```

LCD 4-bit Option

```
delayMs(30); /* initialization sequence */
LCD_nibble_write(0x30, 0);
delayMs(10);
LCD_nibble_write(0x30, 0);
delayMs(1);
LCD_nibble_write(0x30, 0);
delayMs(1);
LCD_nibble_write(0x20, 0); /* use 4-bit data mode */
delayMs(1);
LCD_command(0x28); /* set 4-bit data, 2-line, 5x7 font */
LCD_command(0x06); /* move cursor right */
LCD_command(0x01); /* clear screen, move cursor to home */
LCD_command(0x0F); /* turn on display, cursor blinking */
}
```

Basic Programming Techniques Part 2

LCD 4-bit Option

```
void LCD_nibble_write(unsigned char data, unsigned
char control)
{

data &= 0xF0; /* clear lower nibble for control */

control &= 0x0F; /* clear upper nibble for data */

PTD->PDOR = data | control; /* RS = 0, R/W = 0 */
PTD->PDOR = data | control | EN; /* pulse E */

delayMs(0);
PTD->PDOR = data;
PTD->PDOR = 0;
}
```

LCD 4-bit Option

```
void LCD_command(unsigned char command)
{
/* upper nibble first */

LCD_nibble_write(command & 0xF0, 0);

/* then lower nibble */

LCD_nibble_write(command << 4, 0);

if (command < 4)

delayMs(4); /* commands 1 and 2 need up to 1.64ms */
else
delayMs(1); /* all others 40 us */
}
```

LCD 4-bit Option

```
void LCD_data(unsigned char data)
{
LCD_nibble_write(data & 0xF0, RS); /* upper nibble first */
LCD_nibble_write(data << 4, RS); /* then lower nibble */
delayMs(1);
}
/* Delay n milliseconds
* The CPU core clock is set to MCGFLLCLK at 41.94 MHz in
SystemInit().
*/
void delayMs(int n) {
int i;
int j;
for(i = 0 ; i < n; i++)
for(j = 0 ; j < 7000; j++) { }
}
```

LCD Cursor Position

In the LCD, one can move the cursor to any location in the display by issuing an address command. The next character sent will appear at the cursor position.

For the two-line LCD, the address command for the first location of line 1 is 0x80, and for line 2 it is 0xC0.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A6	A5	A4	A3	A2	A1	A0

where $A_6A_5A_4A_3A_2A_1A_0$= 0000000 to 0100111 for line 1 and $A_6A_5A_4A_3A_2A_1A_0$ = 1000000 to 1100111 for line 2. See Table 3-3.

LCD Cursor Position

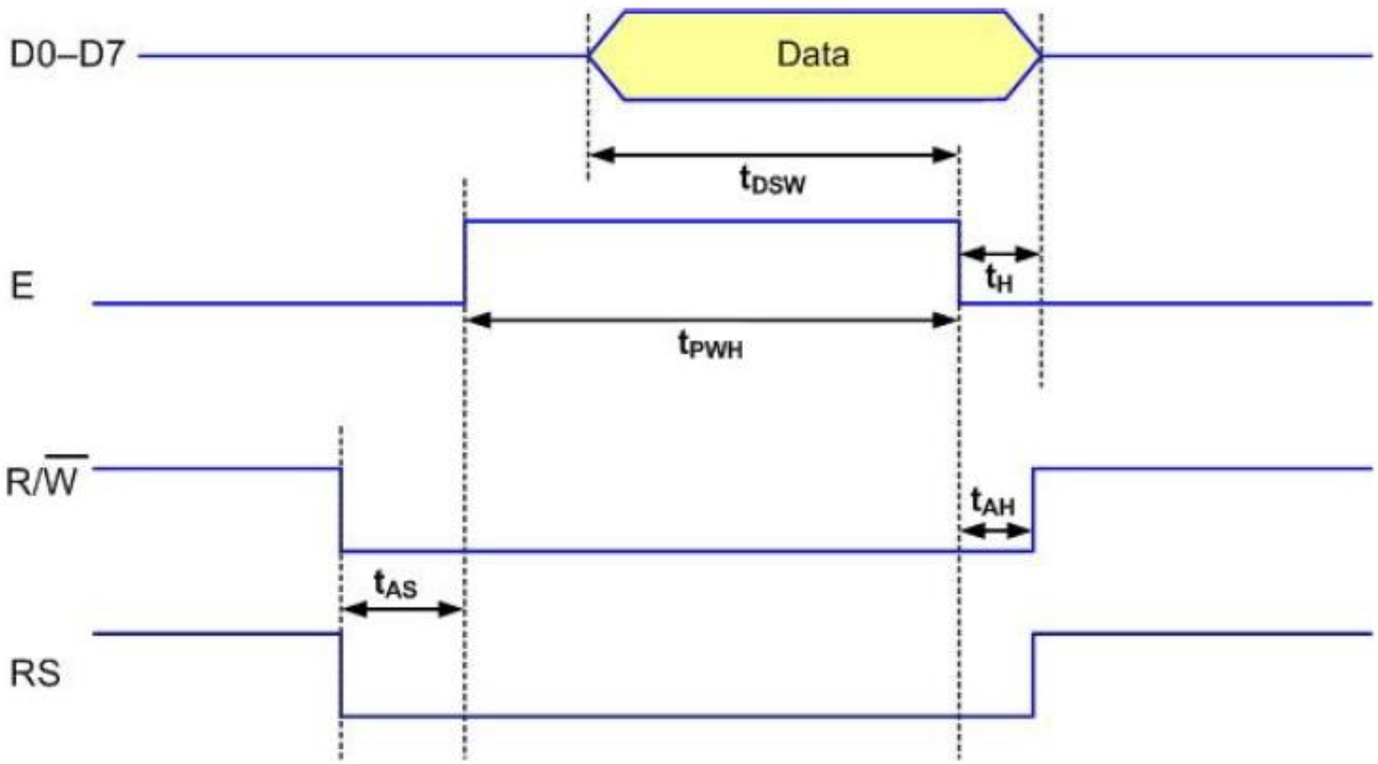
	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Line 1 (min)	1	0	0	0	0	0	0	0
Line 1 (max)	1	0	1	0	0	1	1	1
Line 2 (min)	1	1	0	0	0	0	0	0
Line 2 (max)	1	1	1	0	0	1	1	1

The upper address range can go as high as 0100111 for the 40-character wideLCD while for the 20-character-wide LCD the address of the visible positions goes up to 010011 (19 decimal = 10011 binary).

Basic Programming Techniques Part 2

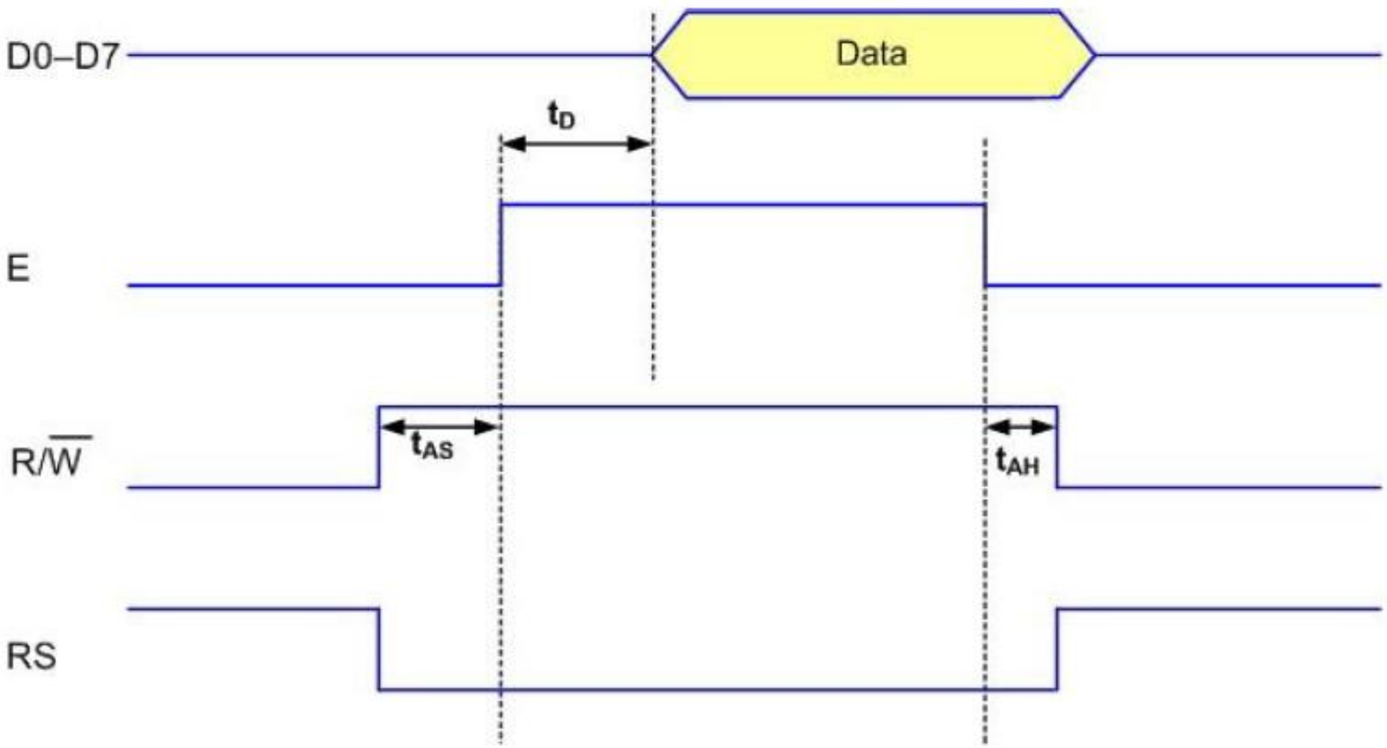
LCD Cursor Position

LCD timing and datasheet (writing timing)



t_{PWH} = Enable pulse width = 230 ns (minimum)
t_{DSW} = Data setup time = 80 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 40 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

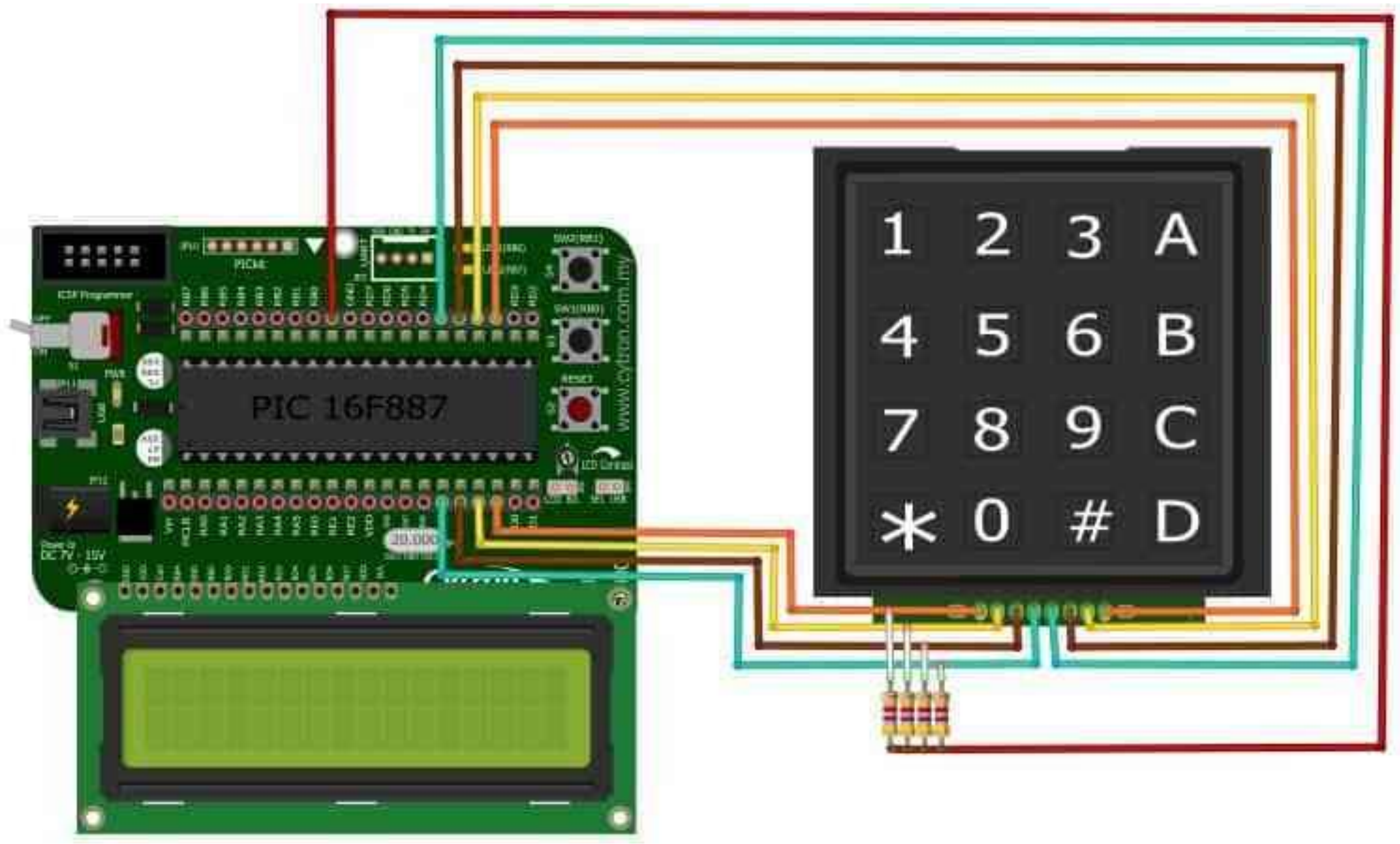
LCD timing and datasheet (read timing)



t_D = Data output delay time
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 40 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Note: Read requires an L-to-H pulse for the E pin.

Interfacing the Keyboard to the CPU



Interfacing the Keyboard to the CPU

To reduce the microcontroller I/O pin usage, keyboards are organized in a matrix of rows and columns.

The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 × 8 matrix of 64 keys can be connected to a microprocessor.

When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns

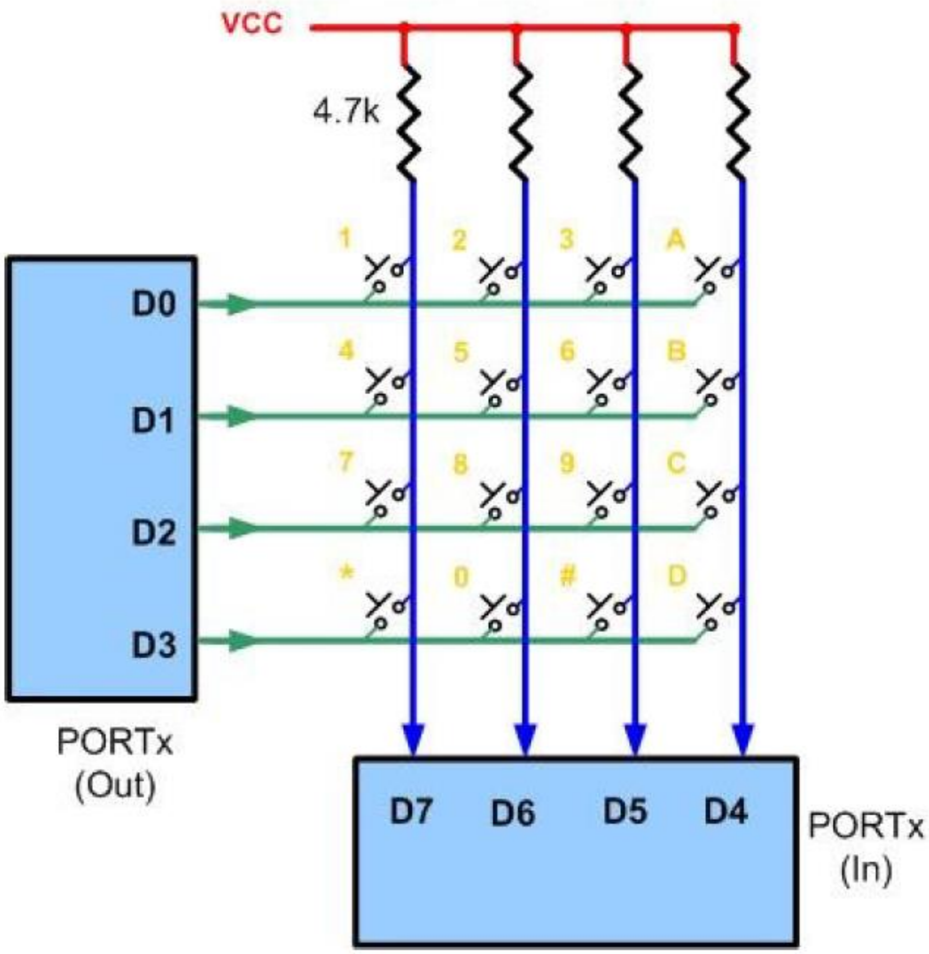
In this section, we look at the mechanism by which the microprocessor scans and identifies the key.

Scanning and identifying the key

The figure shows a 4 × 4 matrix connected to two ports.

The rows are connected to an output port and the columns are connected to an input port.

All the input pins have pull-up resistor connected.

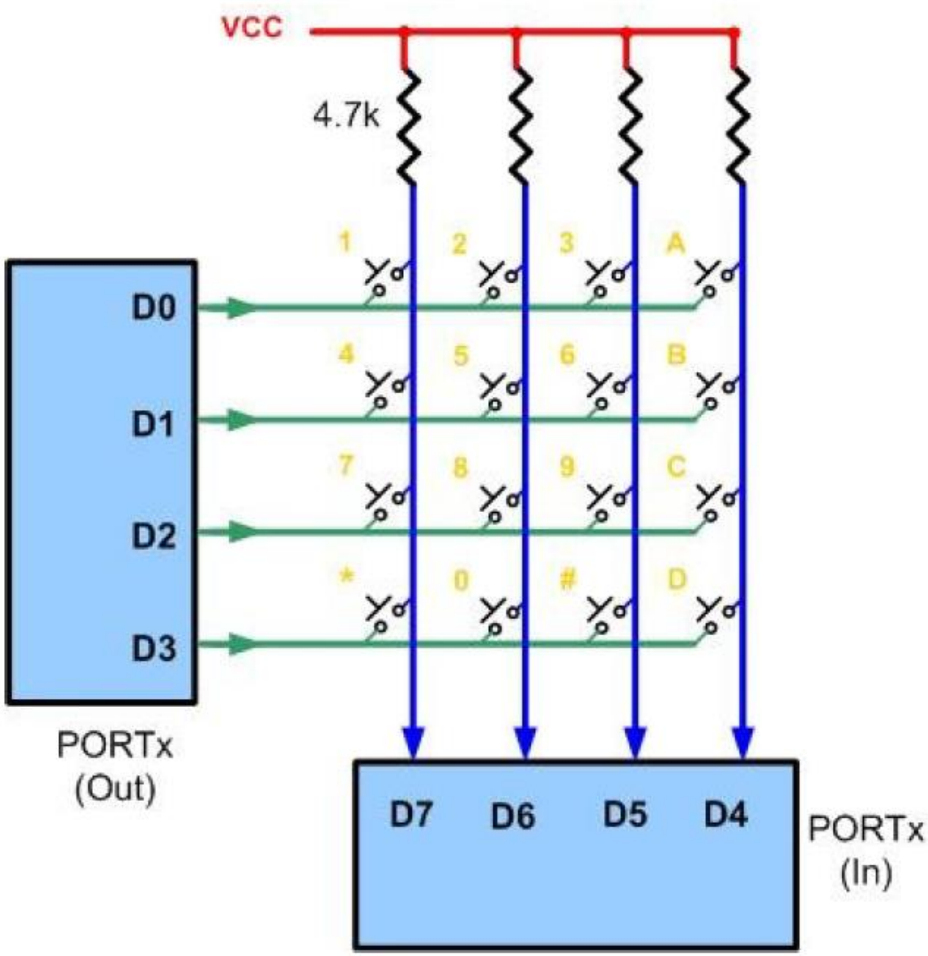


Scanning and identifying the key

If no key has been pressed, reading the input port will yield 1s for all columns.

If all the rows are driven low and a key is pressed, the column will read back a 0

It is the function of the microprocessor to scan the keyboard continuously to detect and identify the key pressed.



Key press detection

To detect the key pressed, the microprocessor drives all rows low then it reads the columns.

If the data read from the columns is D7–D4 = 1111, no key has been pressed and the process continues until a key press is detected

However, if one of the column bits has a zero, this means that a key was pressed.

For example, if D7–D4= 1101, this means that a key in the D5 column has been pressed

Key press detection

```
/* Matrix keypad detect
* This program checks a 4x4 matrix keypad to see
whether
* a key is pressed or not. When a key is pressed, it
turns
* on the blue LED.
*
* PortC 7-4 are connected to the columns and PortC
3-0 are connected
* to the rows.
*/
#include <MKL25Z4.H>
void delayUs(int n);
void keypad_init(void);
char keypad_kbhit(void);
```

Key press detection

```
int main(void)
{
keypad_init();
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */
while(1)
{ if (
keypad_kbhit() != 0) /* if a key is pressed? */

PTD->PCOR |= 0x02; /* turn on blue LED */
else
PTD->PSOR |= 0x02; /* turn off blue LED */
}
}
```

Key press detection

```
/* Initializes PortC that is connected to the keypad.
*  Pins configured as GPIO input pin with pullup enabled.
*/
void keypad_init(void)
{
SIM->SCGC5 |= 0x0800; /* enable clock to Port C */
PORTC->PCR[0] = 0x103; /* PTD0 as GPIO and enable pullup*/
PORTC->PCR[1] = 0x103; /* PTD1 as GPIO and enable pullup*/
PORTC->PCR[2] = 0x103; /* PTD2 as GPIO and enable pullup*/
PORTC->PCR[3] = 0x103; /* PTD3 as GPIO and enable pullup*/
PORTC->PCR[4] = 0x103; /* PTD4 as GPIO and enable pullup*/
PORTC->PCR[5] = 0x103; /* PTD5 as GPIO and enable pullup*/
PORTC->PCR[6] = 0x103; /* PTD6 as GPIO and enable pullup*/
PORTC->PCR[7] = 0x103; /* PTD7 as GPIO and enable pullup*/
PTD->PDDR = 0x0F; /* make PTD7-0 as input pins */
}
```

Key press detection

```
/* This is a non-blocking function.
* If a key is pressed, it returns 1.
* Otherwise, it returns a 0 (not ASCII '0'). */
char keypad_kbhit(void)
{
int col;
PTC->PDDR |= 0x0F; /* enable all rows */
PTC->PCOR = 0x0F;
delayUs(2); /* wait for signal return */
col = PTC->PDIR & 0xF0; /* read all columns */
PTC->PDDR = 0; /* disable all rows */
if (col == 0xF0)
return 0; /* no key pressed */
else
return 1; /* a key is pressed */
}
```

Basic Programming Techniques Part 2

Key Identification

After a key press is detected, the microprocessor will go through the process of identifying the key.

Starting from the top row, the microprocessor drives one row low at a time; then it reads the columns.

If the data read is all 1s, no key in that row is pressed and the process is moved to the next row. It drives the next row low, reads the columns, and checks for any zero.

This process continues until a row is identified with a zero in one of the columns.

Key Identification

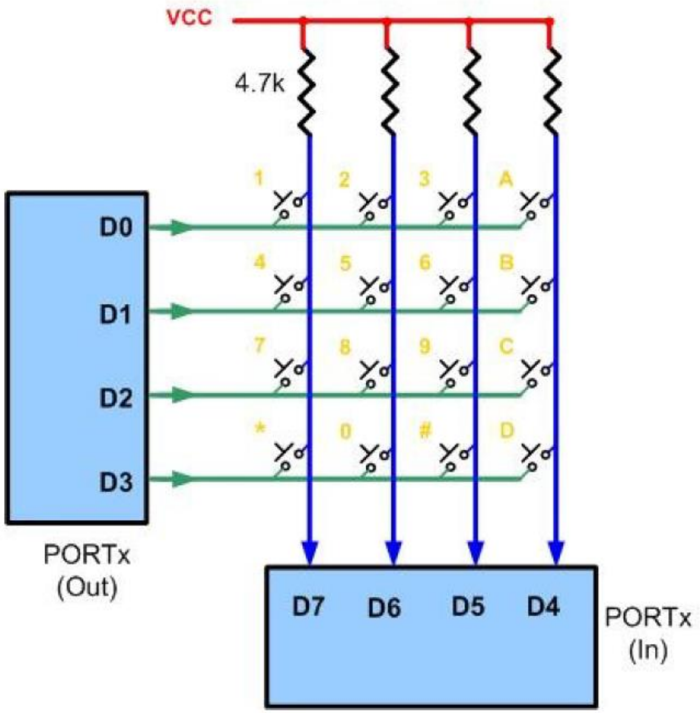
The next task is to find out which column the pressed key belongs to. This should be easy since each column is connected to a separate input pin

Example

identify the row and column of the pressed key for each of the following.

(a) D3–D0 = 1110 for the row,
D7–D4= 1011 for the column

(b) D3–D0 = 1101 for the row
D7–D4= 0111 for the column



Basic Programming Techniques Part 2

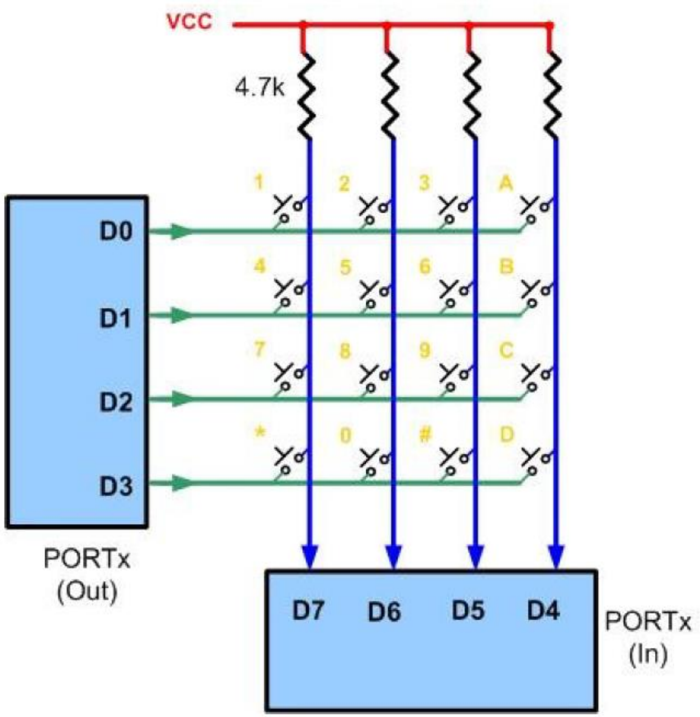
Key Identification

The next task is to find out which column the pressed key belongs to. This should be easy since each column is connected to a separate input pin

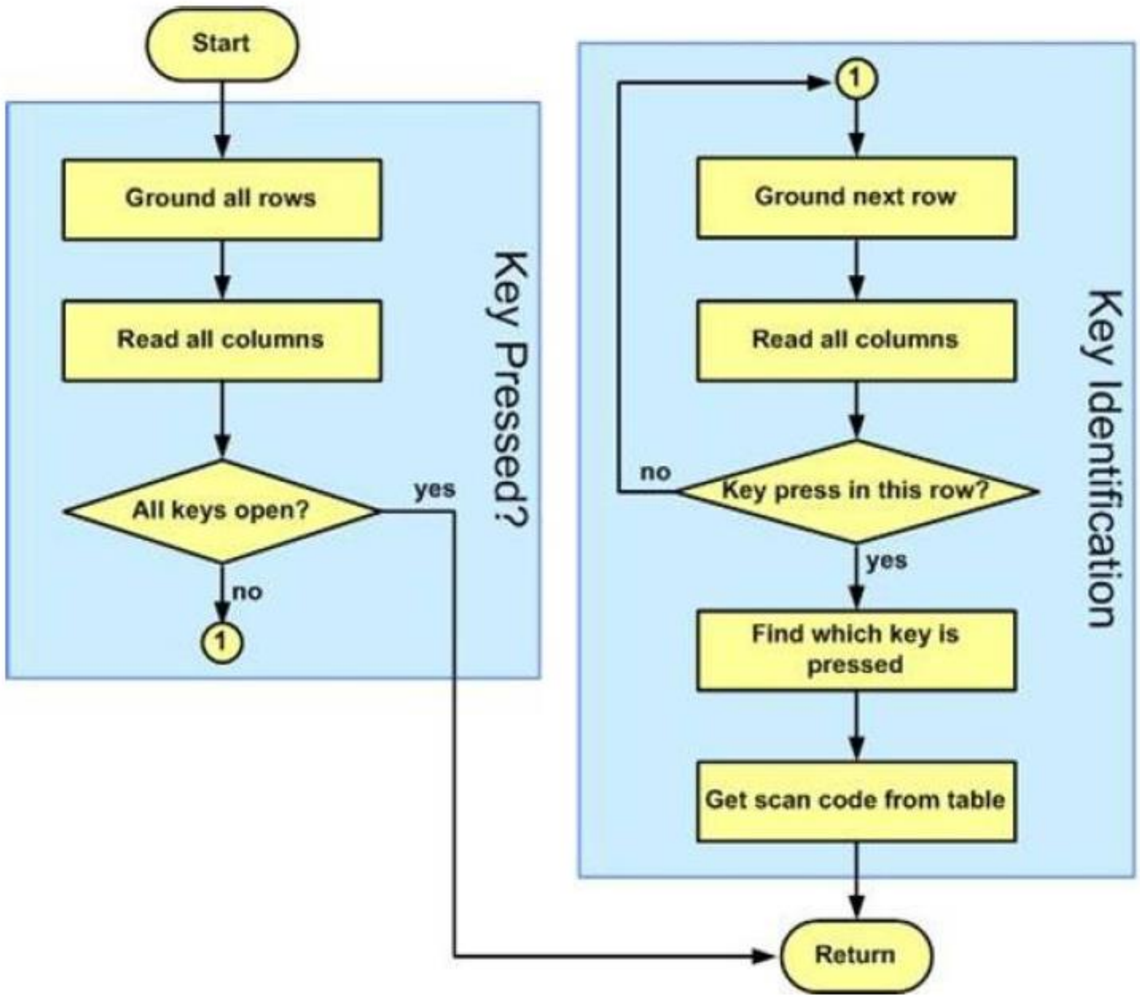
Solution

(a) The row belongs to D0 and the column belongs to D6; therefore, the key number 2 was pressed.

(b) The row belongs to D1 and the column belongs to D7; therefore, the key number 4 was pressed.



Key Press Detection and Identification



Key Press Detection and Identification

The next program provides an implementation of the detection and identification algorithm in C language

First for the initialization of the ports, Port C pins 3-0 are used for rows. The Port C pins 7-4 are used for columns.

They are all configured as input digital pin to prevent accidental short circuit of two output pins.

If output pins are driven high and low and two keys of the same column are pressed at the same time by accident, they will short the output low to output high of the adjacent pins and cause damages to these pins.

Basic Programming Techniques Part 2

Key Press Detection and Identification

To prevent this, all pins are configured as input pin and only one pin is configured as output pin at a time.

Since only one pin is actively driving the row, shorting two rows will not damage the circuit.

The input pins are configured with pull-up enabled so that when the connected keys are not pressed, they stay high and read as 1.

The key scanning function is a non-blocking function, meaning the function returns regardless of whether there is a key pressed or not.

Key Press Detection and Identification

The function first drives all rows low and check to see if any key pressed.

If no key is pressed, a zero is returned. Otherwise the code will proceed to check one row at a time by driving only one row low at a time and read the columns.

If one of the columns is active, it will find out which column it is.

With the combination of the active row and active column, the code will find out the key that is pressed and return a unique numeric code.

Key Press Detection and Identification

```
/* This program scans a 4x4 matrix keypad and
returns a unique number or each key pressed.
The number is displayed on the tri-color
* LEDs using previous cod
*
* PortC 7-4 are connected to the columns and
PortC 3-0 are connected to the rows. */

#include <MKL25Z4.H>

void delayMs(int n);
void delayUs(int n);
void keypad_init(void);
char keypad_getkey(void);
void LED_init(void);
void LED_set(int value);
```

Key Press Detection and Identification

```
int main(void)
{
unsigned char key;

keypad_init();
LED_init();

while(1)
{

key = keypad_getkey();
LED_set(key); /* set LEDs according to the key code */

}
}
```

Key Press Detection and Identification

```
/* Initializes PortC that is connected to the keypad.
*  Pinsconfigured as GPIO input pin with pull-up enabled.
*/
void keypad_init(void)
{
SIM->SCGC5 |= 0x0800; /* enable clock to Port C */
PORTC->PCR[0] = 0x103; /* PTD0 as GPIO and enable pullup*/
PORTC->PCR[1] = 0x103; /* PTD1 as GPIO and enable pullup*/
PORTC->PCR[2] = 0x103; /* PTD2 as GPIO and enable pullup*/
PORTC->PCR[3] = 0x103; /* PTD3 as GPIO and enable pullup*/
PORTC->PCR[4] = 0x103; /* PTD4 as GPIO and enable pullup*/
PORTC->PCR[5] = 0x103; /* PTD5 as GPIO and enable pullup*/
PORTC->PCR[6] = 0x103; /* PTD6 as GPIO and enable pullup*/
PORTC->PCR[7] = 0x103; /* PTD7 as GPIO and enable pullup*/
PTD->PDDR = 0x0F; /* make PTD7-0 as input pins */
}
```

Key Press Detection and Identification

```
/* keypad_getkey()
* If a key is pressed, it returns a key code. Otherwise,
a zero is returned.
•  The upper nibble of Port C is used as input. Pull-ups
   are enabled when the keys are not pressed
* The lower nibble of Port C is used as output that
drives the keypad rows.
* First all rows are driven low and the input pins are
read. If no key is pressed, it will read as all ones.
Otherwise, some key is pressed.
* If any key is pressed, the program drives one row low
at a time and leave the rest of the rows inactive (float)
then read the input pins.
* Knowing which row is active and which column is active,
the program can decide which key is pressed. */
```

Key Press Detection and Identification

```
char keypad_getkey(void) {

int row, col;

const char row_select[] = {0x01, 0x02, 0x04, 0x08};
/* one row is active */

/* check to see any key pressed */

PTC->PDDR |= 0x0F; /* enable all rows */
PTC->PCOR = 0x0F;
delayUs(2); /* wait for signal return */
col = PTC->PDIR & 0xF0; /* read all columns */
PTC->PDDR = 0; /* disable all rows */
if (col == 0xF0)
return 0; /* no key pressed */
```

Key Press Detection and Identification

```
/* If a key is pressed, it gets here to find out which key.
* It activates one row at a time and read the input to see
* which column is active. */

for (row = 0; row < 4; row++)
{ PTC->PDDR = 0; /* disable all rows */

PTC->PDDR |= row_select[row]; /* enable one row */
PTC->PCOR = row_select[row]; /* drive the active row low */

delayUs(2); /* wait for signal to settle */
col = PTC->PDIR & 0xF0; /* read all columns */

if (col != 0xF0) break; /* if one of the input is low, some
key is pressed. */
}
```

Key Press Detection and Identification

```
PTC->PDDR = 0; /* disable all rows */

if (row == 4)
return 0; /* if we get here, no key is pressed */

/* gets here when one of the rows has key pressed, check
which column it is*/

if (col == 0xE0) return row * 4 + 1; /* key in column 0 */
if (col == 0xD0) return row * 4 + 2; /* key in column 1 */
if (col == 0xB0) return row * 4 + 3; /* key in column 2 */
if (col == 0x70) return row * 4 + 4; /* key in column 3 */
return 0; /* just to be safe */
}
```

Key Press Detection and Identification

```
/* initialize all three LEDs on the FRDM board */
void LED_init(void)
{
SIM->SCGC5 |= 0x400; /* enable clock to Port B */
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
PTB->PDDR |= 0x40000; /* make PTB18 as output pin */
PTB->PSOR |= 0x40000; /* turn off red LED */
PORTB->PCR[19] = 0x100; /* make PTB19 pin as GPIO */
PTB->PDDR |= 0x80000; /* make PTB19 as output pin */
PTB->PSOR |= 0x80000; /* turn off green LED */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */
PTD->PSOR |= 0x02; /* turn off blue LED */
}
```

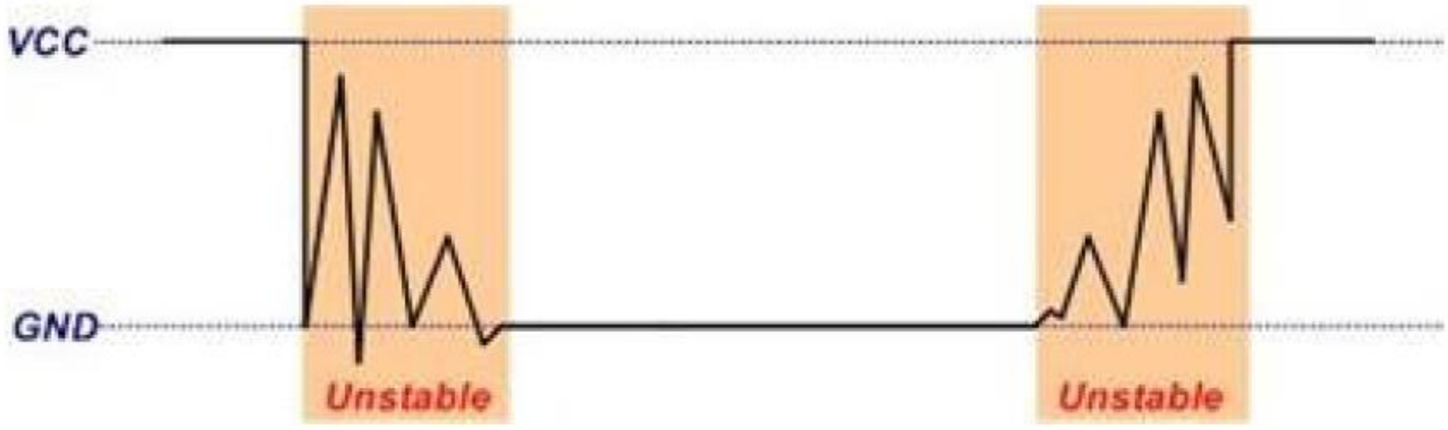
Key Press Detection and Identification

```
/* turn on or off the LEDs according to bit 2-0 of the value */
void LED_set(int value)
{
if (value & 1) /* use bit 0 of value to control red LED */
PTB->PCOR = 0x40000; /* turn on red LED */
else
PTB->PSOR = 0x40000; /* turn off red LED */
if (value & 2) /* use bit 1 of value to control green LED */
PTB->PCOR = 0x80000; /* turn on green LED */
else
PTB->PSOR = 0x80000; /* turn off green LED */
if (value & 4) /* use bit 2 of value to control blue LED */
PTD->PCOR = 0x02; /* turn on blue LED */
else
PTD->PSOR = 0x02; /* turn off blue LED */
}
```

Contact Bounce and Debounce

When a mechanical switch is closed or opened, the contacts do not make a clean transition instantaneously, rather the contacts open and close several times before they settle.

This event is called contact bounce



Basic Programming Techniques Part 2

Contact Bounce and Debounce

So it is possible when the program first detects a switch in the keypad is pressed but when interrogating which key is pressed, it would find no key pressed.

This is the reason we have a return 0 after checking all the rows. Another problem manifested by contact bounce is that one key press may be recognized as multiple key presses by the program.

Contact bounce also occurs when the switch is released. Because the switch contacts open and close several times before they settle, the program may detect a key press when the key is released.

Contact Bounce and Debounce

For many applications, it is important that each key press is only recognized as one action.

When you press a numeral key of a calculator, you expect to get only one digit. A contact bounce results in multiple digits entered with a single key press.

A simple software solution is that when a transition of the contact state change is detected such as a key pressed or a key released, the software does a delay for about 10 – 20 ms to wait out the contact bounce.

After the delay, the contacts should be settled and stable.

Contact Bounce and Debounce