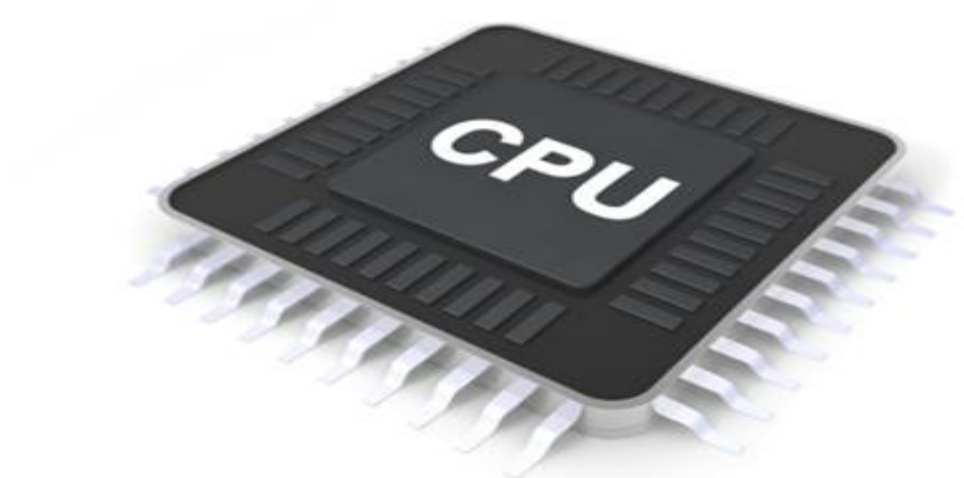


Simple Parallel Interfacing

ENEE 3587

Microp Interfacing

Simple Parallel Interfaces

- Simple parallel interfaces:
  - Simple input/output devices
  - No interrupts needed
  - Connected to port pins
  - Not using any peripherals
- Examples:
  - Drive LED’s
  - Read push button
  - Read dip switch

Voltage Characteristics

- Voltage parameters related to electrical compatibility
  - Input high voltage (V_{IH})
  - Input low voltage (V_{IL})
  - Output high voltage (V_{OH})
  - Output low voltage (V_{OL})
- For device X to drive (i.e. control/provide input to) device Y:
  - VOH of device X ≥ VIH of device Y.
  - VOL of device X ≤ VIL of device Y
- In case of miss-maching characteristics we need to condition the signals
  - E.g. use op-amps

Current Characteristics

- Current parameters related to electrical compatibility
  - Input high current (I_{IH})
  - Input low current (I_{IL})
  - Output high current (I_{OH})
  - Output low current (I_{OL})
- Device X current driving device Y:
  - Current flows out from the device X when the driving voltage is high.
  - Current flows into the device X when the driving voltage is low.
  - Device X must have enough sourcing (supply current) and sinking (absorb current) capability.
- If a device X cannot source or sink enough current, then using buffer device is a common solution.

Mega2560 DC Characteristics

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, Except XTAL1 and Reset pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$[1] $0.3V_{CC}$[1]	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$[1]	V
V_{IL2}	Input Low Voltage, RESET pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$[1]	V
V_{IH}	Input High Voltage, Except XTAL1 and RESET pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$[2] $0.6V_{CC}$[2]		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$[2] $0.7V_{CC}$[2]		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH2}	Input High Voltage, RESET pin	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$[2]		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage[3], Except RESET pin	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V			0.9 0.6	V
V_{OH}	Output High Voltage[4], Except RESET pin	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor		20		50	kΩ

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{CC}	Power Supply Current[5]	Active 1MHz, V_{CC} = 2V (ATmega640/1280/2560/1V)		0.5	0.8	mA
		Active 4MHz, V_{CC} = 3V (ATmega640/1280/2560/1L)		3.2	5	
		Active 8MHz, V_{CC} = 5V (ATmega640/1280/1281/2560/2561)		10	14	
		Idle 1MHz, V_{CC} = 2V (ATmega640/1280/2560/1V)		0.14	0.22	
		Idle 4MHz, V_{CC} = 3V (ATmega640/1280/2560/1L)		0.7	1.1	
		Idle 8MHz, V_{CC} = 5V (ATmega640/1280/1281/2560/2561)		2.7	4	
	Power-down mode	WDT enabled, V_{CC} = 3V		<5	15	µA
		WDT disabled, V_{CC} = 3V		<1	7.5	
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V $V_{in} = V_{CC}/2$		<10	40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V $V_{in} = V_{CC}/2$	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Input and output voltage levels of common logic families

Logic family	VCC	V_{IH}	V_{OH}	V_{IL}	V_{OL}
S[4]	5 V	2 V	3.0~3.4 V[1]	0.8 V	0.4~0.5 V[2]
LS[4]	5 V	2 V	3.0~3.4 V[1]	0.8 V	0.4~0.5 V[2]
AS[4]	5 V	2 V	3.0~3.4 V[1]	0.8 V	0.35 V
F[4]	5 V	2 V	3.4 V	0.8 V	0.3 V
HC[3]	5 V	3.5 V	4.9 V	1.5 V	0.1 V
HCT[3]	5 V	3.5 V	4.9 V	1.5 V	0.1 V
ACT[3]	5 V	2 V	4.9 V	0.8 V	0.1 V
ABT[5]	5 V	2 V	3 V	0.8 V	0.55 V
BCT[5]	5 V	2 V	3.3 V	0.8 V	0.42 V
FCT[5]	5 V	2 V	2.4 V	0.8 V	0.55 V

Notes.

1. V_{OH} value will get lower when output current is larger.
2. V_{OL} value will get higher when output current is larger. The V_{OL} values of different logic gates are slightly different.
3. HC, HCT, ACT are based on the CMOS technology.
4. S, LS, AS and F logic families are based on the bipolar technology.
5. ABT, BCT, and FCT are using the Bi-CMOS technology.

Source: H-W Huang

Current capabilities of common logic families [1]

Logic family	VCC	I_{IH}	I_{IL}	I_{OH}	I_{OL}
S	5 V	50 mA	1.0 mA	1 mA	20 mA
LS	5 V	20 mA	0.2 mA	15 mA	24 mA
AS	5 V	20 mA	0.5 mA	15 mA	64 mA
F	5 V	20 mA	0.5 mA	1 mA	20 mA
HC[3]	5 V	1 mA	1 mA	25 mA	25 mA
HCT[3]	5 V	1 mA	1 mA	25 mA	25 mA
ACT[3]	5 V	1 mA	1 mA	24 mA	24 mA
ABT[3]	5 V	1 mA	1 mA	32 mA	64 MA
BCT	5 V	20 mA	1 mA	15 mA	64 mA
FCT[3]	5 V	1 mA	1 mA	15 mA	64 mA

Notes.

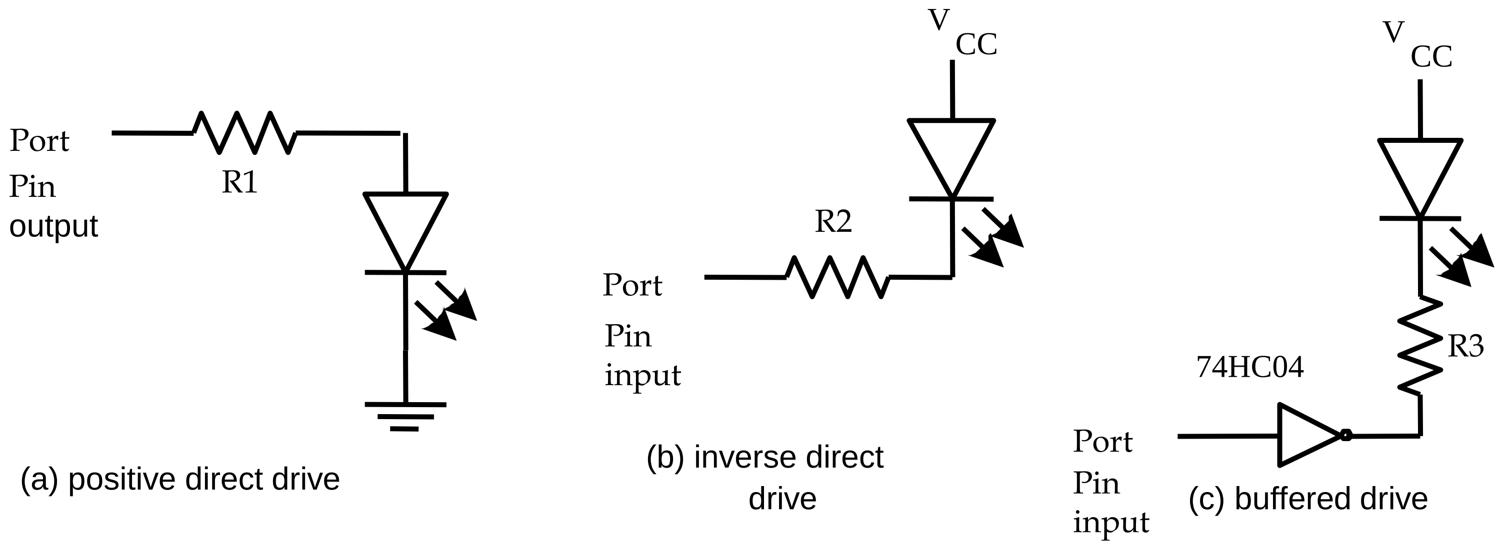
1. Values are based on the 74xx244 of Texas Instrument (xx is the technology name: S, LS, AS, F, … etc.)

2. The values for I_{IH} and I_{IL} are input leakage currents.

Source: H-W Huang

Interfacing with LED Devices

- Circuit (a) and (b) are recommended for LEDs that need only small current to light (1~2mA).
- Circuit (c) is recommended for LEDs that need larger current to light.
- Current limiting resistors should be 1.5k-2kΩ.



(a) positive direct drive

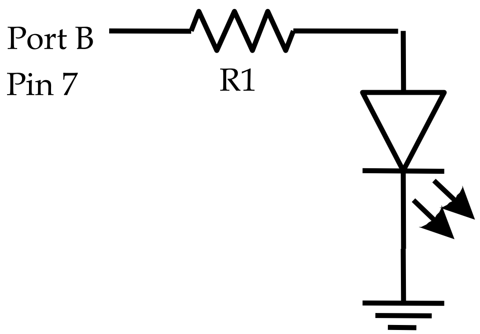
(b) inverse direct drive

(c) buffered drive

GPIO Programming for LEDs

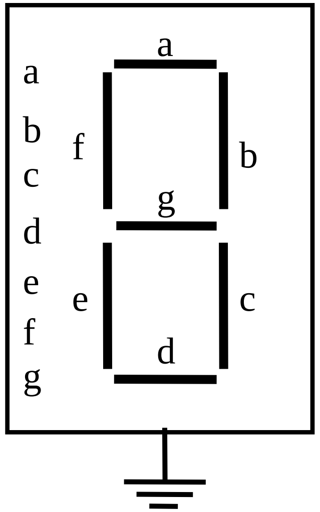
- Design Process:
  - Choose a port/pin to connect the LED to.
  - Determine LED type: forward vs inverse driven
    - Pin = HI for ON vs Pin = LO for ON
  - Set DDR register of the pin to output
  - Set/Clear pin using PORT register
- Example: Use pin 7 of port B to control direct driven LED

```
DDRB  = 0x80;  //output on pin7
PORTB = 0x80;  //portb_pin7 = HI
```



Driving a 7-Segment Displays

- Not advisable to connect without a buffer chip when other IOs are connected
- Octal buffer chip: 74HC244 provides 8 buffered lines.
  - VOH = 5V
  - Use 300Ω to provide drive current of 10mA
- 7 segments: a-g
  - Dot is a typical 8th LED
- 2 Variations
  - Common Cathode:
    - All LEDs are grounded
    - Send “1” to turn LED on
  - Common Anode:
    - All LEDs are connected to Vcc
    - Send a “0” to turn LED on



Common Cathode 7SD Table

BCD digit	Segments						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

GPIO Programming for Single 7SD

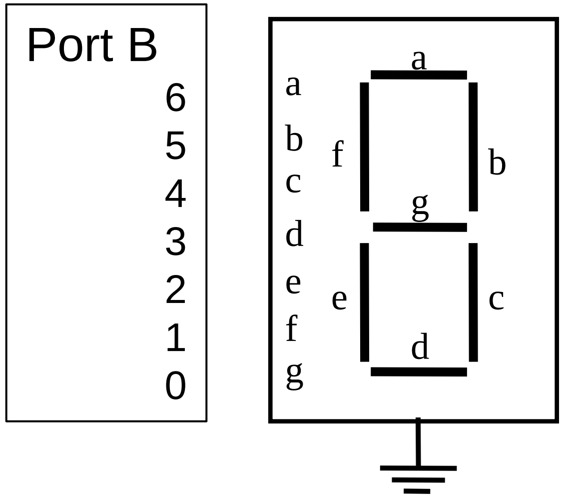
- Design Process:
  - Determine if 7SD type: common cathode vs anode
    - Common GND pin vs common VCC pin for LEDs
    - LED on: HI input vs LO input
  - Determine PORT/pins connections
    - A min of 7 pins
  - Set the DDR register to output
  - Convert the table into an array of inputs
  - To display digit i, output array[i]

7SDx1 Common Cathode Example 1

```
CHAR SSD[]={0x7e, 0x30, 0x6d, 0x79, ..., 0x7b};

DDRB = 0x7F;     //PORTB PINS 6-0 OUTPUT

PORTB = SSD[0];  //DISPLAYS "0" ON 7SD
```



Digit	xabcdefg	HEX
0	*1111110	0x7E
1	*0110000	0x30
2	*1101101	0x6D
3	*1111001	0x79
4	*0110011	0x33
5	*1011011	0x5B
6	*1011111	0x5F
7	*1110000	0x70
8	*1111111	0x7F
9	*1111011	0x7B

7SDx1 Common Cathode Example 2

- Using the same circuit as before, display 0 to 9 repeatedly on the 7SD. Each digit will be displayed for 1sec.

```
CHAR SSD[]={0x7e, 0x30, 0x6d, 0x79, ..., 0x7b};
DDRB = 0x7F;       //PORTB PINS 6-0 OUTPUT

while(1)
{   for (int i=0; i<10; i++)
        {   PORTB = SSD[i];       //DISPLAYS i ON 7SD
            delay1ms(1000);       //delay function
        }
}
```

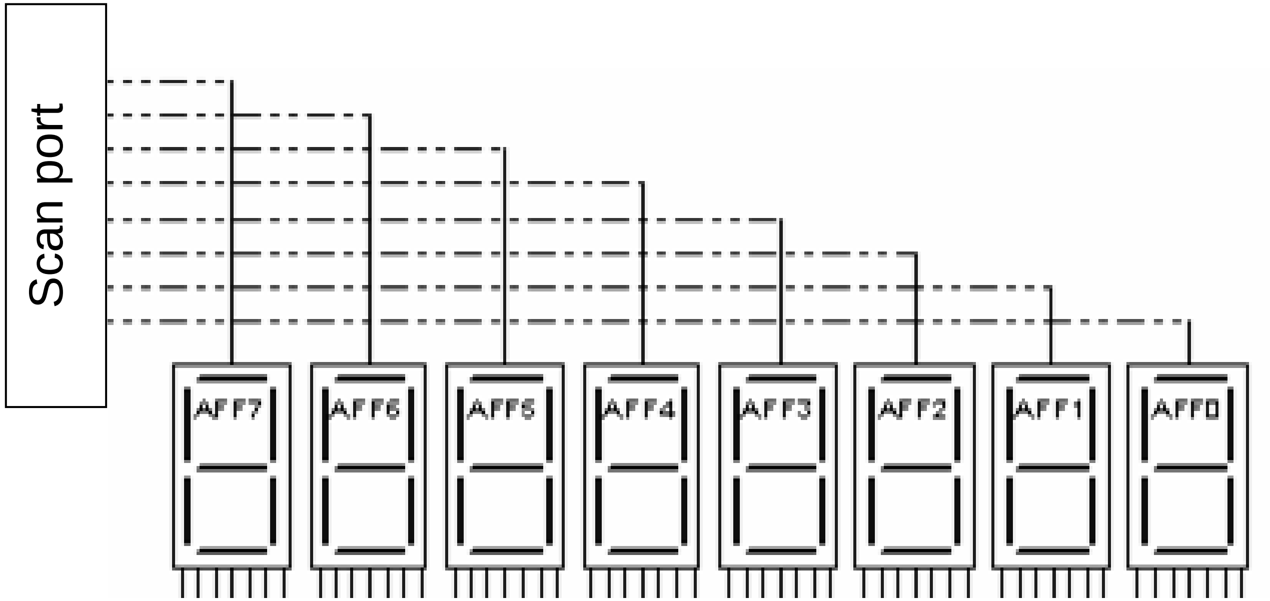
Delay functions

- Crude method: Use multiple loops
  - One loop to control micro increments (e.g. 1ms, 1us, etc.)
  - Micro increments must be converted into count
    - Count is controlled by the number of cycles AND frequency of device
  - Another to control macro increments
- Calculating delay in seconds: delay time = cycles/Clock
  - For a clock of 16MHz: 1ms = 16K/16M
- Function format:

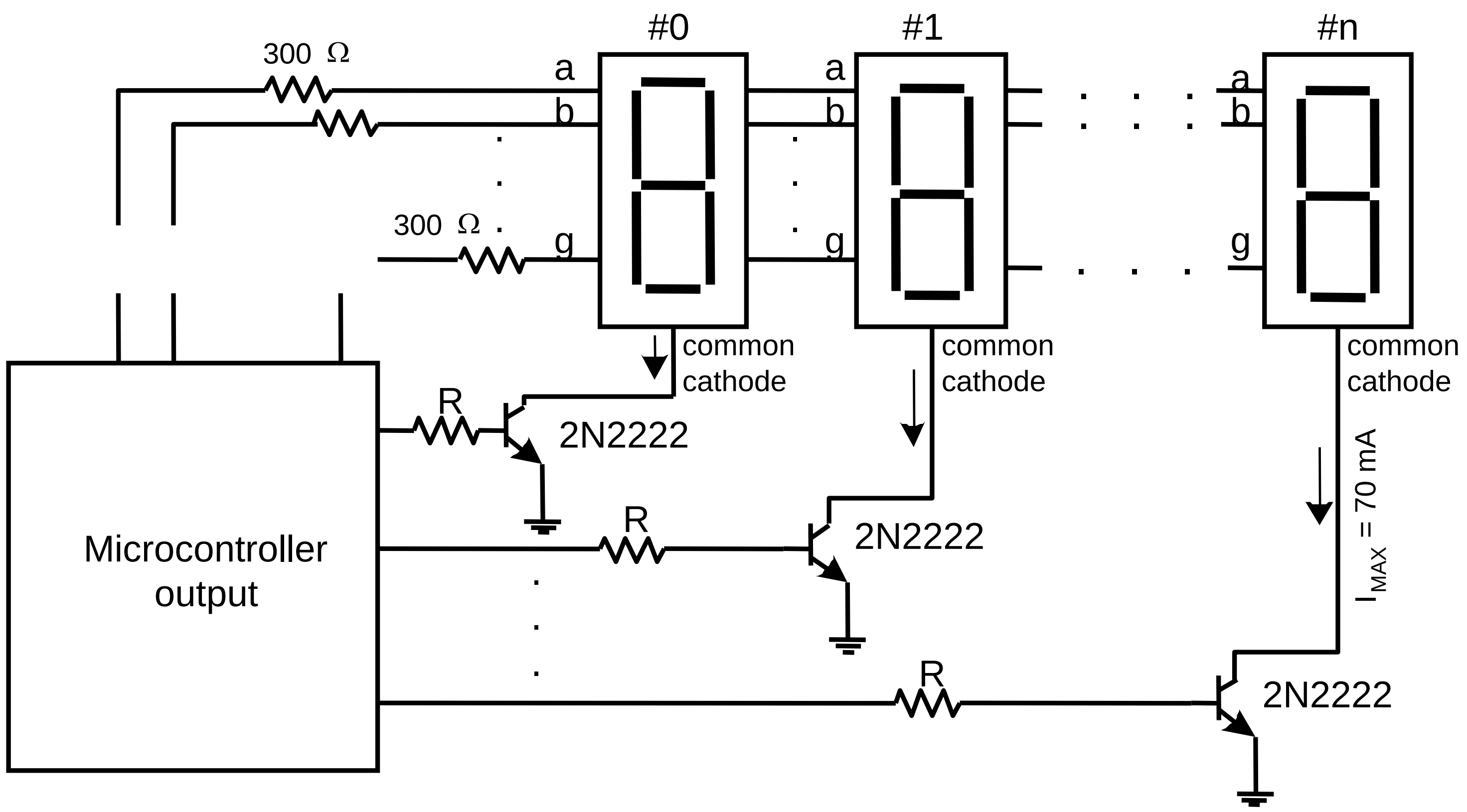
```
void delay1u(int macro_time)
{  for(int i=0; i< macro_time; i++)
         for(int j=0; j<micro_count ; j++)
               do_somethimg_useless;
}
```

Multiple 7SD

- For multiple 7-segmnets display: Commonly scanning is implemented
  - Another port used to select which chip to activate
  - A NPN transistor (2N2222) can be employed to help switch on the chip
  - NPN is connected to common cathode. If the cathode is allowed to connect to ground the chip will light up.
  - A small resistance 200Ω~1kΩ is needed to drive the transistor into saturation

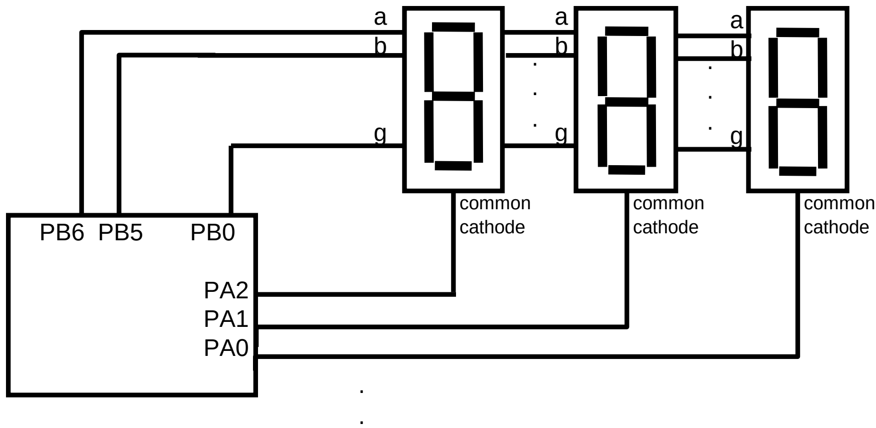


Multiple 7SDs Diagram



Coding Example: Multiple 7SD

- Given: 3x7SDs driven by PB6-PB0. 7SD GND is connected to PA2-PA0. Write a program that will display and repeat the pattern:
  123
  234
  345
  456
  567
  678
  789
  890
  901
  012
  Each line of the pattern should display for 1 sec.



Multiple 7SD Example Solution

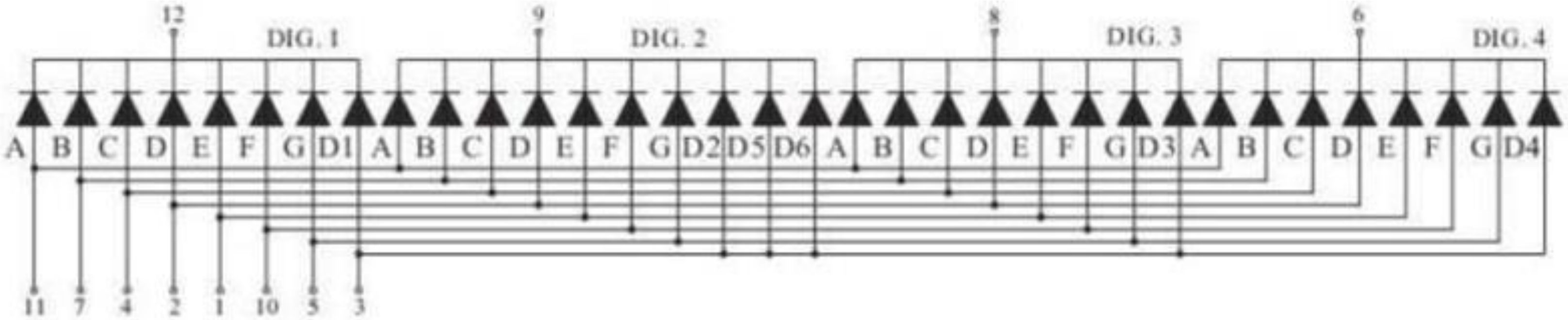
- Idea:
  - display each digit for a very short time (10ms-20ms).
  - Turn on a 7SD by outputting 0 for its corresponding pin in PORTA
    - Only one 7SD should be ON
    - All others use output 1 (OFF)
  - Repeat the line over for 1s.
    - Each 7SD displays for 10ms
    - For 1s repeat: Repeat 1s/delay = 1000ms/(30ms+30ms+30ms) ~ 30x
      - Must experimentally determine 1s for accurate count
  - Move to the next line
    - Start with digit 1
    - Display digit, digit+1, digit+2
    - For next line increment digit

Multiple 7SD Example Algorithm

```
Create an array for each digit: “1”,”2,”…,”9”,”0”,”1”,”2”
Set the PORTA,B for output
For i = 1 to 10:                                  //10 lines of patterns
        For j = 1 to 30:                          //Assume 30x30ms ~ 1s
                PORTB = array[i]
                PORTA = ~4              //4=b100; ~4= 011; PA2=011
                delay1ms(10)
                PORTB = array[i+1]
                PORTA = ~2              //2=b010; ~2=101;  PA1=101
                delay1ms(10)
                PORTB = array[i+2]
                PORTA = ~1              //1=b001; ~1=110; PA0=110
                delay1ms(10)
```

4x Seven Segment Display

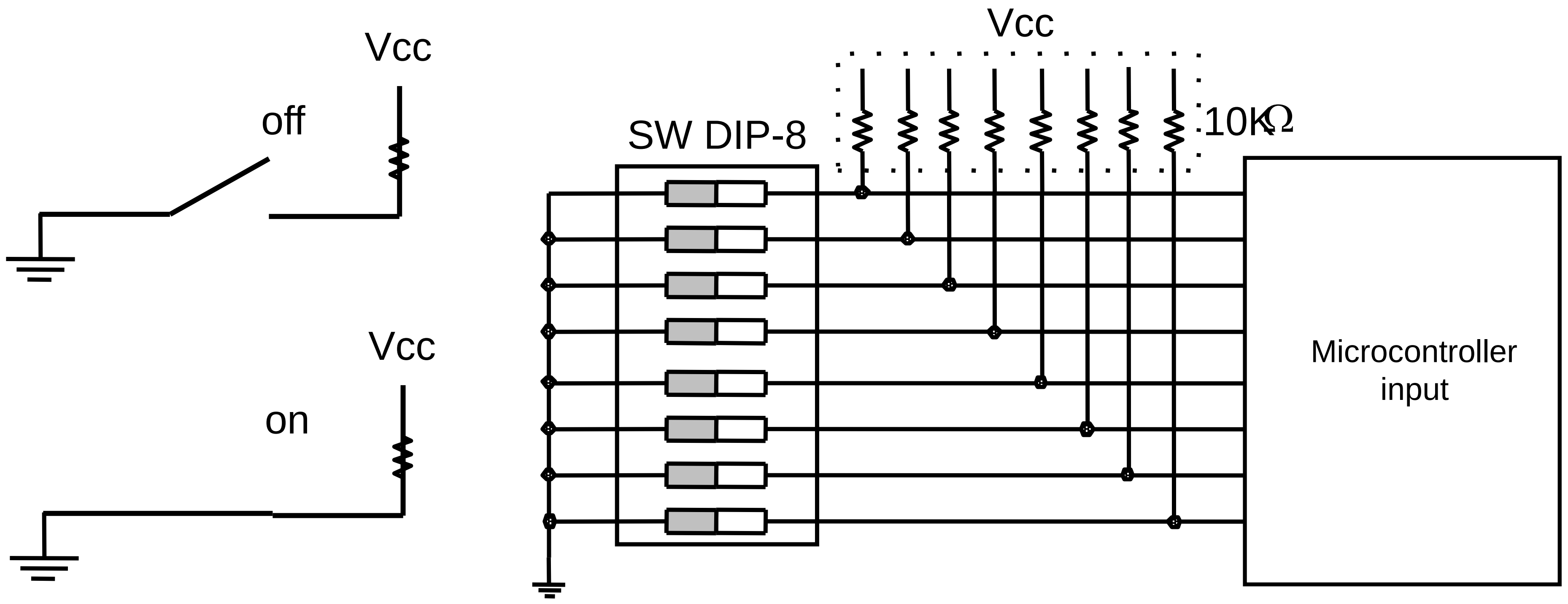
- 5643A: common cathode; 5643B: common anode;
- Segments a,b,c,d,e,f,g: pins 11,7,4,2,1,10,5
- “Enable” pins for 7SD1,2,3,4: pins 12, 9,8,6
  - Common cathode: GND
  - Common anode: VCC
- dot segment: pin 3




en_dig1
segA
segF
en_dig2
en_dig3
segB
12
11
10
9
8
7
dig1
dig2
dig3
dig4
a
f
b
g
e
c
d
dot
1
2
3
4
5
6
segE
segD
dot
segC
segG
en_dig4

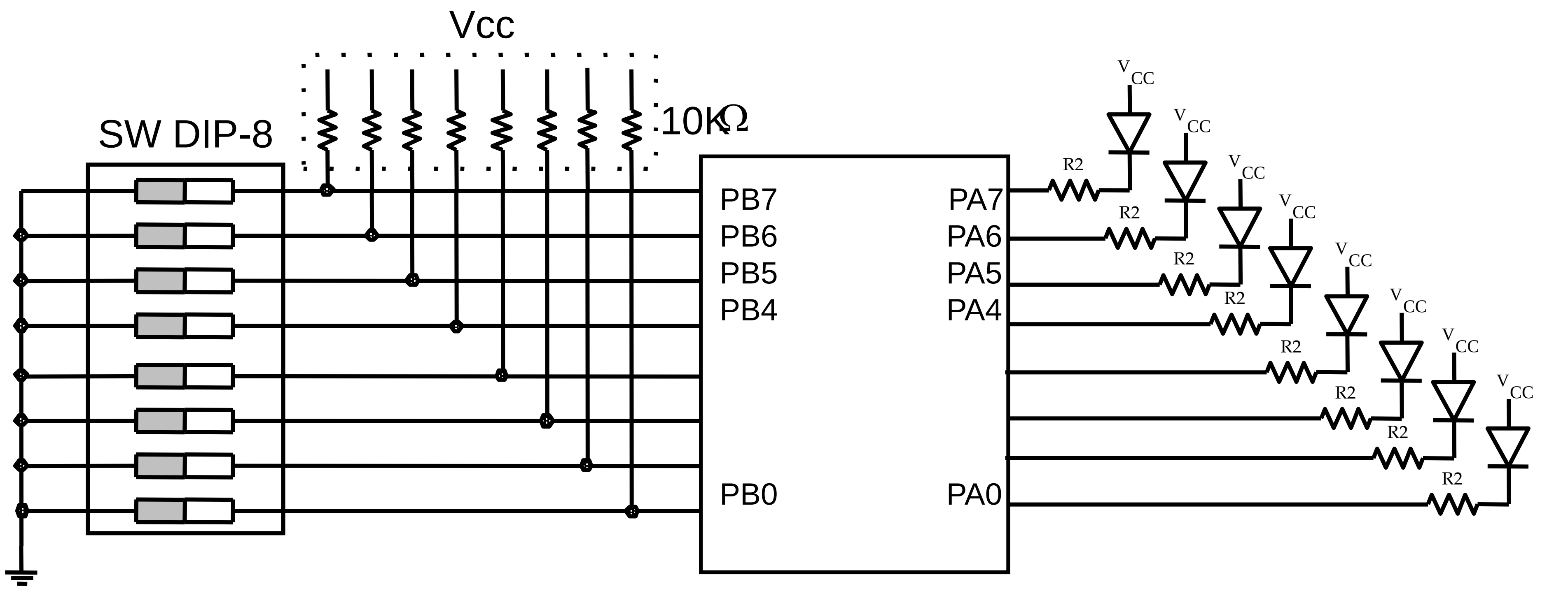
Interfacing DIP Switches

- Dual inline package (DIP): simple input device
- DIP switches are often used to provide setup information to MCU.
- For inputs = 5V use 10kΩ to provide 0.5mA input to MCU
- When in ON position current flows from input to MCU



8xDIP Switch to LEDs

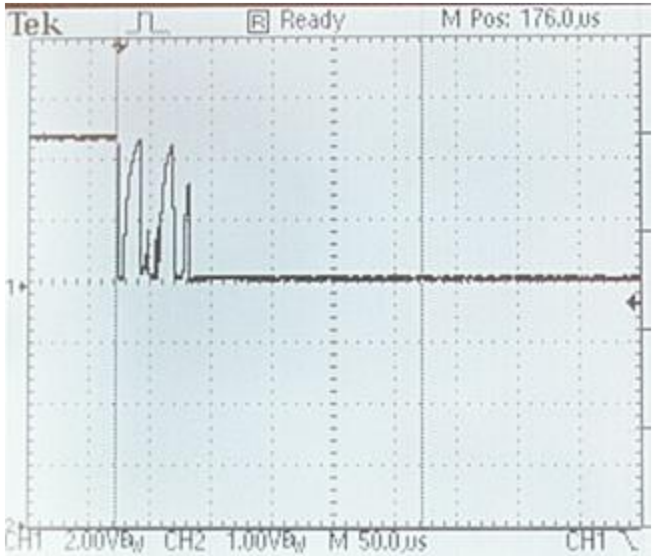
- An 8 input DIP switch is used to turn on 4 LEDs as showt in the diagram. When the switch is ON, turn the corresponding LED ON.



DIP, LED Example Code

- Configure PORTS for input/output:
  - PORTB must be input; PORTA must be an output port
- Read PORTB:
  - Use PINB
  - When switch x is ON: PBx = 0
- Turn LEDs ON:
  - When PAx=0: LEDx is ON
  - PORTA = PINB

```
DDRA = 0xFF;          //output
DDRB = 0;             //input
while(1)
        PORTA = PINB;     //BE (NOT) AWARE OF BOUNCE
```



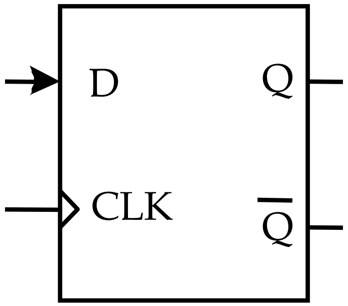
Bounce of a DIP switch
On falling edge 200μs

Push buttons

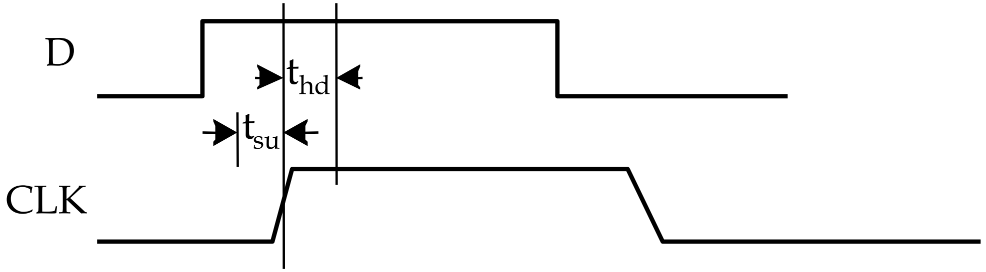
- Do not require advanced synchronization
- A push-button is switch
  - Can be mechanical, membrane, capacitors, or Hall-effect in construction.
- Mechanical switches are most popular.
  - Problem: contact bounce.
  - Closing a mechanical switch generates a series of pulses because the switch contacts do not come to rest immediately.
- Humans cannot type more than 50 keys in a second. Reading the switch more than 50 times a second will read the same key stroke too many times.
  - 1/50 = 0.02s = 20ms

Timing Compatibility

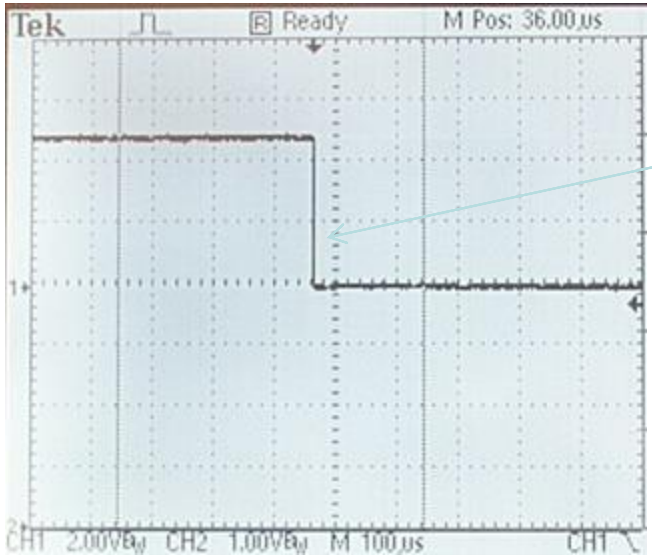
- There is no timing problem when driving a peripheral pin that does not contain latches or flip-flops.
- When driving a latch or flip-flop device, one needs to make sure that the data set up time (t_{SU}) and data hold time (t_{HD}) are both satisfied.
- Not a big issue with simple devices



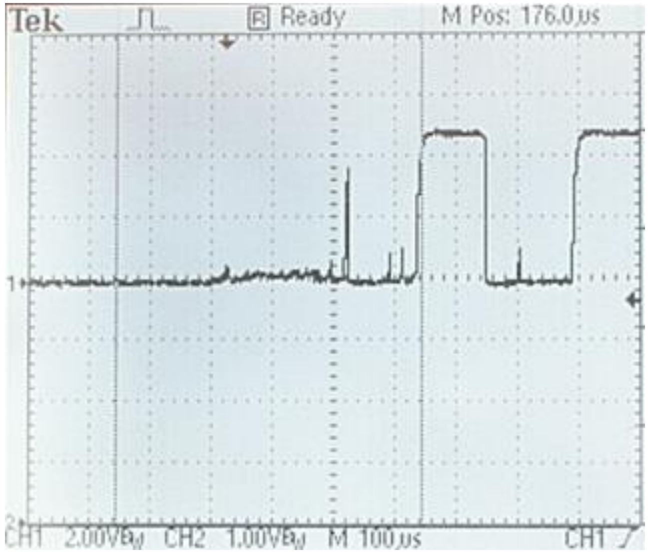
(a)



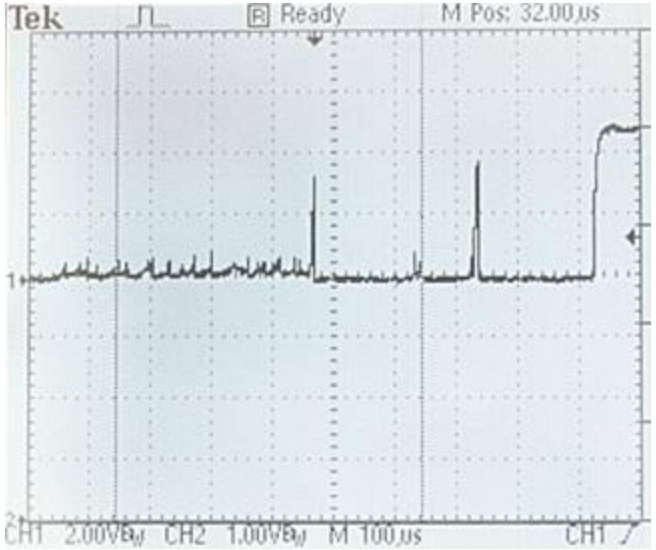
(b)



button is pushed



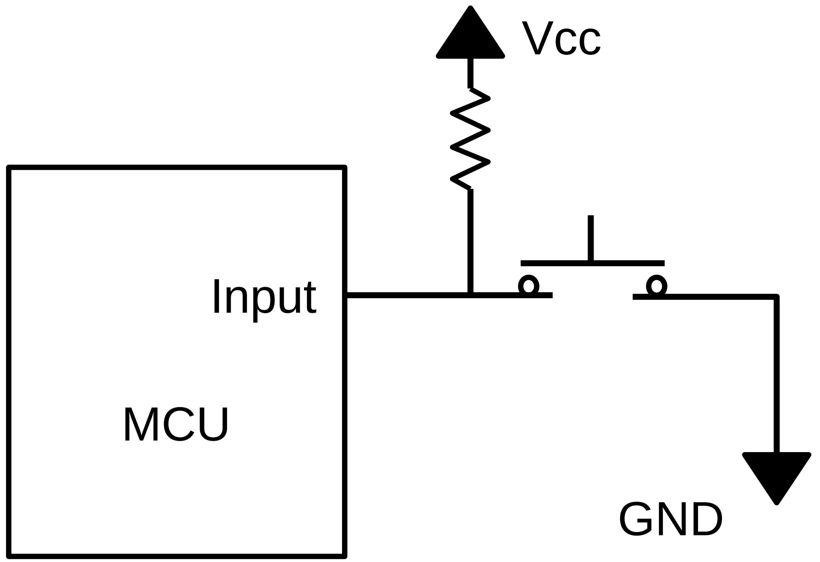
button is released 600 μs bounce



button is released 400 μs bounce

Push Button

- Port must be used as input
- Strategy:
  - Scan the port for a 0
  - Debounce button
- Hardware Debouncing Techniques
  - SR latches
  - Non-inverting CMOS gates
  - Integrating debouncer
- Software debouncing:
  - <200μs delay for DIP; <1ms delay for pushbutton
  - Adv: cheaper, controllable
  - Disadv.: consumes power (interrupt driven methods less power consumption).



Hardware Debouncing

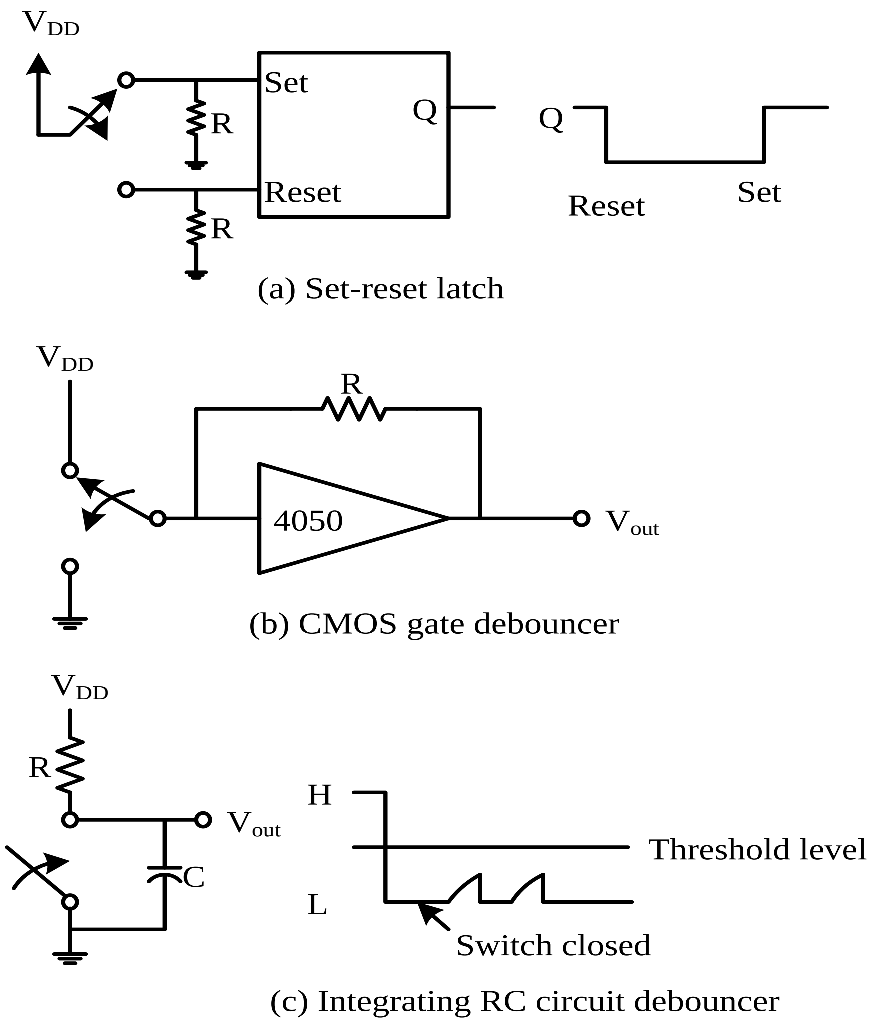
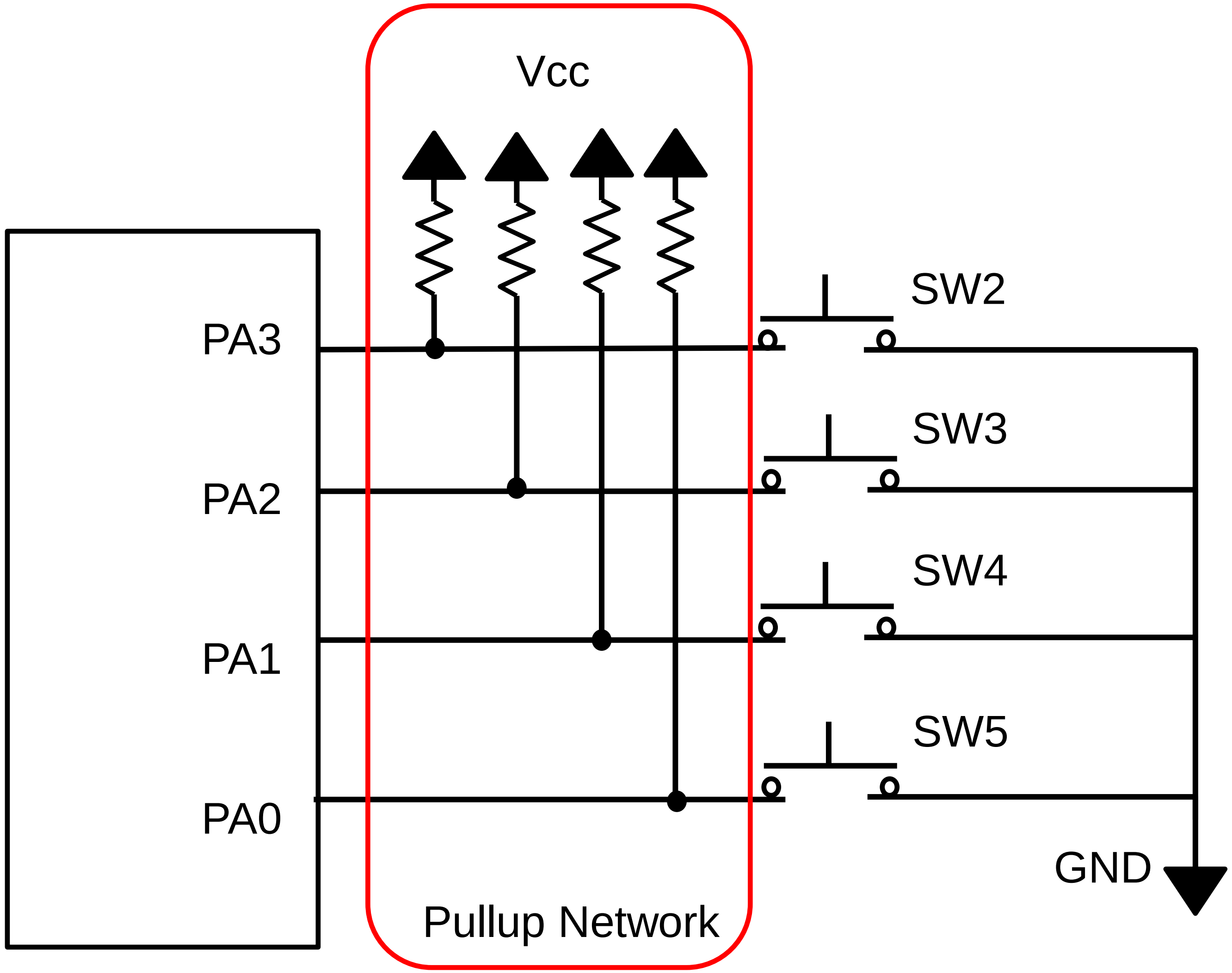


Figure 7.42 Hardware debouncing techniques

Push Button Example



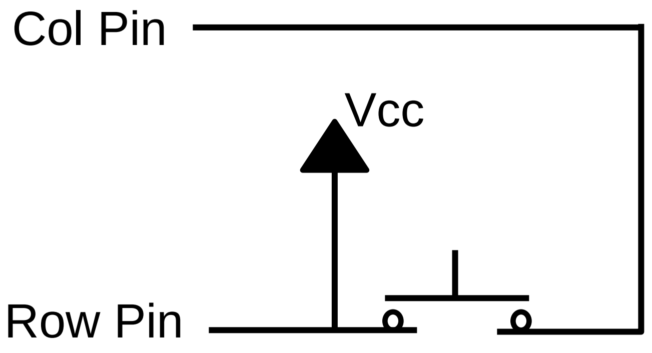
Coding for Push button

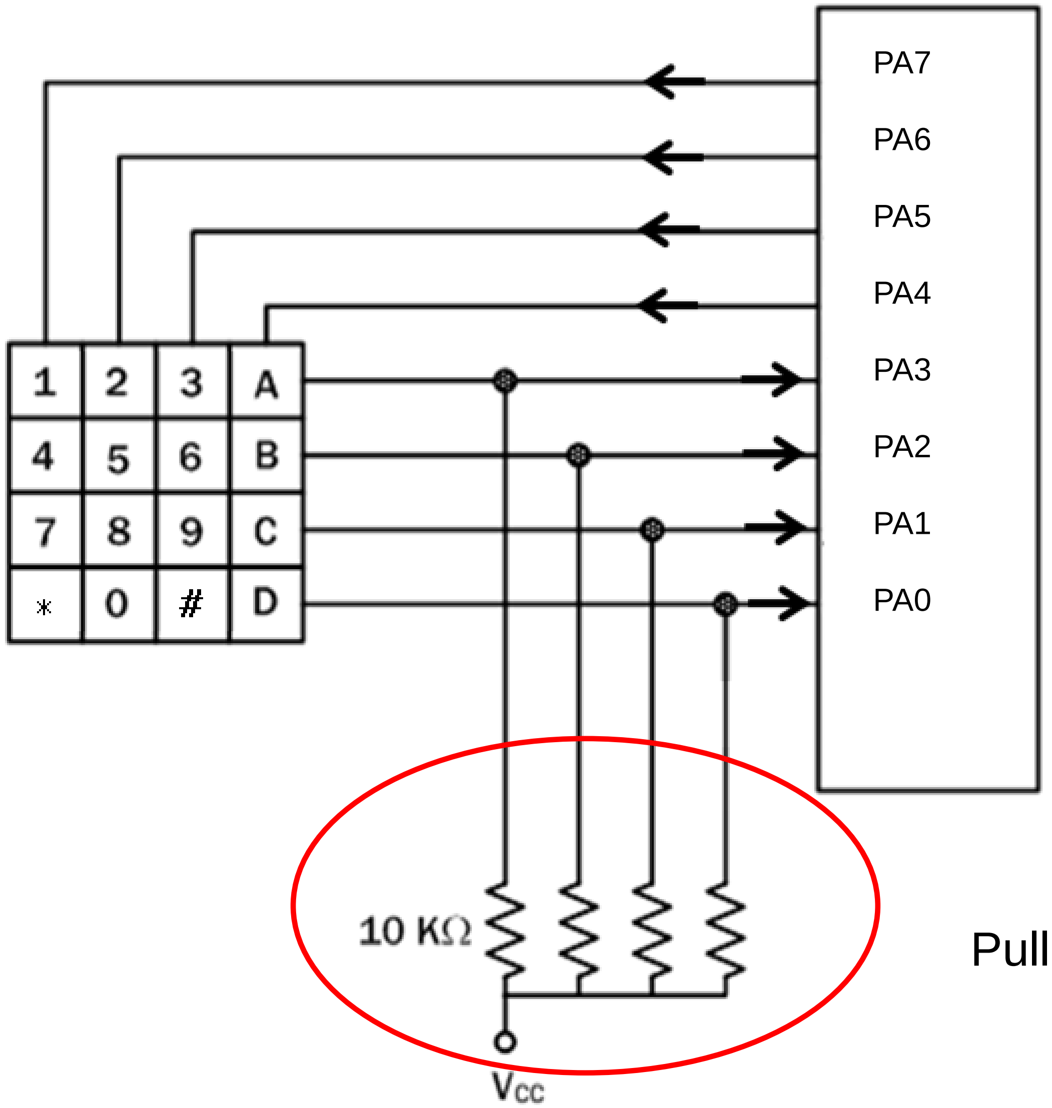
- Set port direction:
  - Read: DDRx = 0
- Optional: Set the port to PULL-UP (if device PULLUP is used)
  - Enable pullup on the port
- Read pin
  - Use PINx
  - PIN = 0 => button is ON
  - If activity is detected, delay 10ms and read again

```
DDRA = 0;
while(PINA==0x0F);          //wait until a button is pressed
delay1ms(10);               //wait for bounce to be over (debounce)
```

Keypad

- Array of pushbutton switches.
  - Typically 12-24 buttons
- Example: 16 keypad interface
  - Rows configured as input
  - Columns configured as output
  - Pullup network connected to rows
  - To scan a column, output configured to 0
  - When button is not pressed input is 1
  - When button is pressed circuit is connected and input 0





Pullup circuit

Strategy

1. Use the built in PULLUP Port circuit
2. Enable a single column (send a 0)
3. Read rows (one by one)
4. If key is pressed, delay by 10ms then read again
   - To overcome bouncing
   - Use row, col combination to determine key pressed
   - (col,row) = (0,0) is “1”
     (col,row) = (4,4) is “D”
   - STOP if key is pressed
5. Enable next column
6. Jump to 2

Code 4x4 Keypad

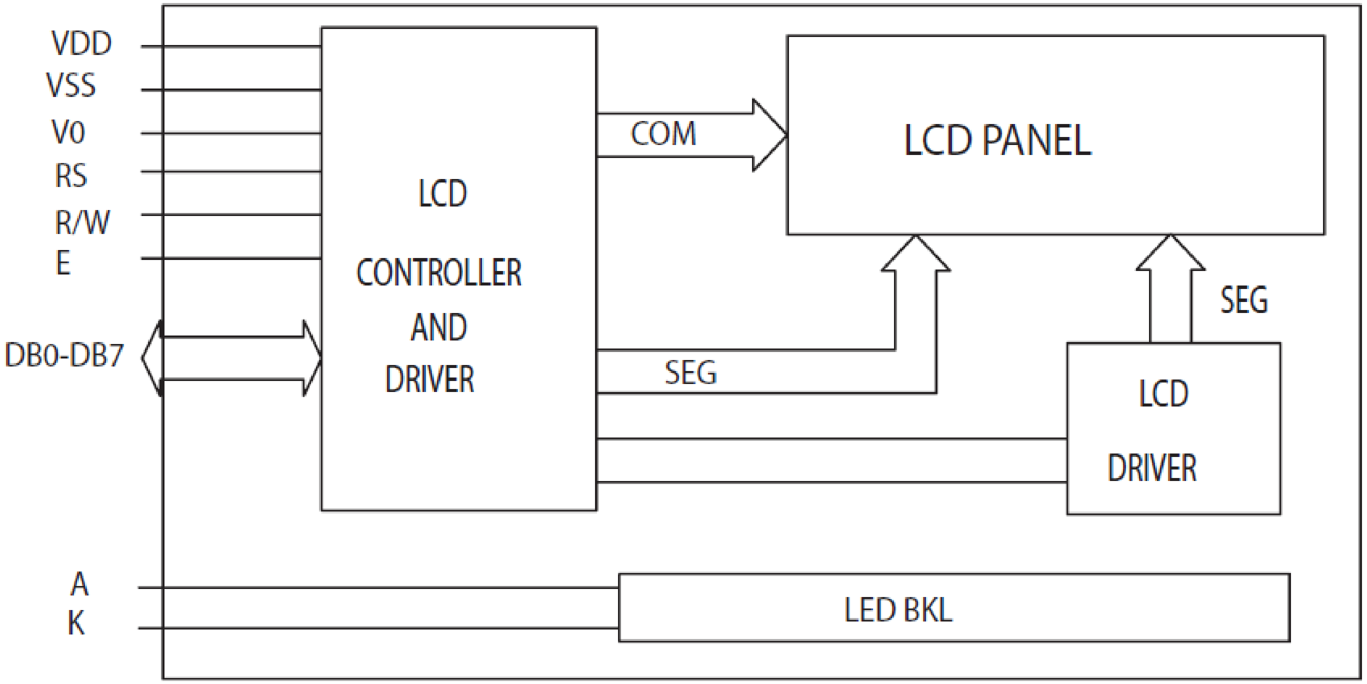
```
char keypad(void)
{       char col[] = {0x70, 0xB0, 0xD0, 0xE0}, char key[][] = {{“1”,”2”,”3”,”A”},…}
        int r;
        DDRA = 0xF0;                            //PA7-4: controls columns, PA3-0: input rows
        for (int c = 0; c< 4; c++)              // enable each column 1 at a time
        {       PORTA = col[c];
                if (PINA != 0x0F)
                {       delay1m(10);            //debounce
                        switch (PINA & 0x0F)
                        {       case(0b0111): r = 0; break;
                                case(0b1011): r = 1; break;
                                case(0b1101): r = 2; break;
                                case(0b1110): r = 3; break;
                                default:
                        }
                        return (key[r][c]);
                }
        }
}
```

LCD

- Most common type allows light to pass through when activated
  - activated when a low frequency bipolar signal in the range of 30 Hz to 1KHz is applied to it.
- Can display characters and graphics.
- Often sold in a module with LCDs and controller unit built in.

LCM1602A LCD Kit

- Display capability: 2 x 16
- DB0-7 can be used to:
  - Send characters to be display on LCD
  - Send commands to LCD (control cursor, fonts, etc.)
  - Receive status/configuration info from LCD



1-2 Lines LCD

Pin No.	symbol	I/O	Function
1	V_{SS}	-	Power supply (GND)
2	V_{CC}	-	Power supply (+5 V)
3	V_{EE}	-	Contrast adjust
4	RS	I	0 = instruction input, 1 = data input
5	$R/\overline{W}$	I	0 = write to LCD, 1 = read from LCD
6	E	I	Enable signal
7	DB0	I/O	Data bus line 0
8	DB1	I/O	Data bus line 1
9	DB2	I/O	Data bus line 2
10	DB3	I/O	Data bus line 3
11	DB4	I/O	Data bus line 4
12	DB5	I/O	Data bus line 5
13	DB6	I/O	Data bus line 6
14	DB7	I/O	Data bus line 7

Pins 15,16 control back lighting

3-4 Line LCD

Pin No.	symbol	I/O	Function
1	DB7	I/O	Data bus line 7
2	DB6	I/O	Data bus line 6
3	DB5	I/O	Data bus line 5
4	DB4	I/O	Data bus line 4
5	DB3	I/O	Data bus line 3
6	DB2	I/O	Data bus line 2
7	DB1	I/O	Data bus line 1
8	DB0	I/O	Data bus line 0
9	E1	I	Enable signal row 0 and 1
10	$R/\overline{W}$	I	0 = write to LCD, 1 = read from LCD
11	RS	I	0 = instruction input, 1 = data input
12	V_{EE}	-	Contrast adjust
13	V_{SS}	-	Power supply (GND)
14	V_{CC}	-	Power supply (+5 V)
15	E2	I	Enable signal row 2 and 3
16	N.C	-	

Instruction Format

Instruction	RS	R/$\overline{W}$	B7	B6	B5	B4	B3	B2	B1	B0	Description	Execution Time
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0)	1.64 ms
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position without changing DDRAM contents. Also returns display being shifted to the original position.	1.64 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D); specifies to shift the display (S). These operations are performed during data read/write.	40 µs
Display on/ off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display(D), cursor on/off (c), and blink of cursor position character (B).	40 µs
Cursor/ display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM contents remain unchanged.	40 µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).	40 µs
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.	40 µs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.	40 µs
Read busy flag and address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous operation).	0 µs
Write CGRAM or DDRAM	1	0	write data								Writes data to CGRAM or DDRAM	40 µs
Read from CGRAM or DDRAM	1	1	read data								Reads data from CGRAM or DDRAM	40 µs

Instruction Bits

Bit Name	Settings	
I/D	0 = decrement cursor position.	1 = increment cursor position
S	0 = no display shift.	1 = display shift
D	0 = display off	1 = display on
C	0 = cursor off	1 = cursor on
B	0 = cursor blink off	1 = cursor blink on
S/C	0 = move cursor	1 = shift display
R/L	0 = shift left	1 = shift right
DL	0 = 4-bit interface	1 = 8-bit interface
N	0 = 1/8 or 1/11 duty (1 line)	1 = 1/16 duty (2 lines)
F	0 = 5 × 8 dots	1 = 5 × 10 dots
BF	0 = can accept instruction	1 = internal operation in progress

Display Data RAM (DDRAM)

- 2x16 LCD Module: 2 lines, 16 characters per line
- Each character location has an address
- Write in that address a character to display it on the LCD

Display Size	Visible	
	Character Positions	DDRAM Addresses
2 * 16	00..15	0x00..0x0F + 0x40..0x4F
2 * 20	00..19	0x00..0x13 + 0x40..0x53
2 * 24	00..23	0x00..0x17 + 0x40..0x57
2 * 32	00..31	0x00..0x1F + 0x40..0x5F
2 * 40	00..39	0x00..0x27 + 0x40..0x67

ROM/RAM & Registers

- CGROM: Character generator ROM
  - generates 5x8 or 5x10 character patterns from a 8-bit code
- CGRAM: Character generator RAM
  - User can rewrite character patterns into CGRAM.
  - Up to eight 5x8 patterns or four 5 x10 patterns can be programmed.
- Registers: two 8-bit user accessible registers:
  - Instruction register (IR): for display control
  - Data register (DR): to write and read

RS	$R/\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.).
0	1	Read busy flag (DB7) and address counter (DB0 to DB6).
1	0	DR write as an internal operation (DR to DDRAM or CGRAM).
1	1	DR read as an internal operation (DDRAM or CGRAM to DR).

Commands

- Clear display
  - Writes 0x20 (space character) to all DDRAM locations.
  - Sets 0 to the address counter (return cursor to upper left corner of the LCD)
  - Sets increment mode
- Return Home
  - Sets address counter to 0.
  - DDRAM contents are not changed.
- Entry Mode Set
  - Sets incrementing or decrementing of the DDRAM address.
  - Controls the shifting (shifts if S bit = 1) of the display
- Display On/Off Control
  - Turns on/off display
  - Turns on/off cursor
  - Turns on/off cursor blinking

Cursor vs Shift

- Cursor can move to right or left
  - Supports different languages
- Display Shift
  - Entire display shifts

S/C	R/L	Operation
0	0	Shifts the cursor position to the left. (AC is decremented by 1)
0	1	Shifts the cursor position to the right. (AC is incremented by 1)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Function Set

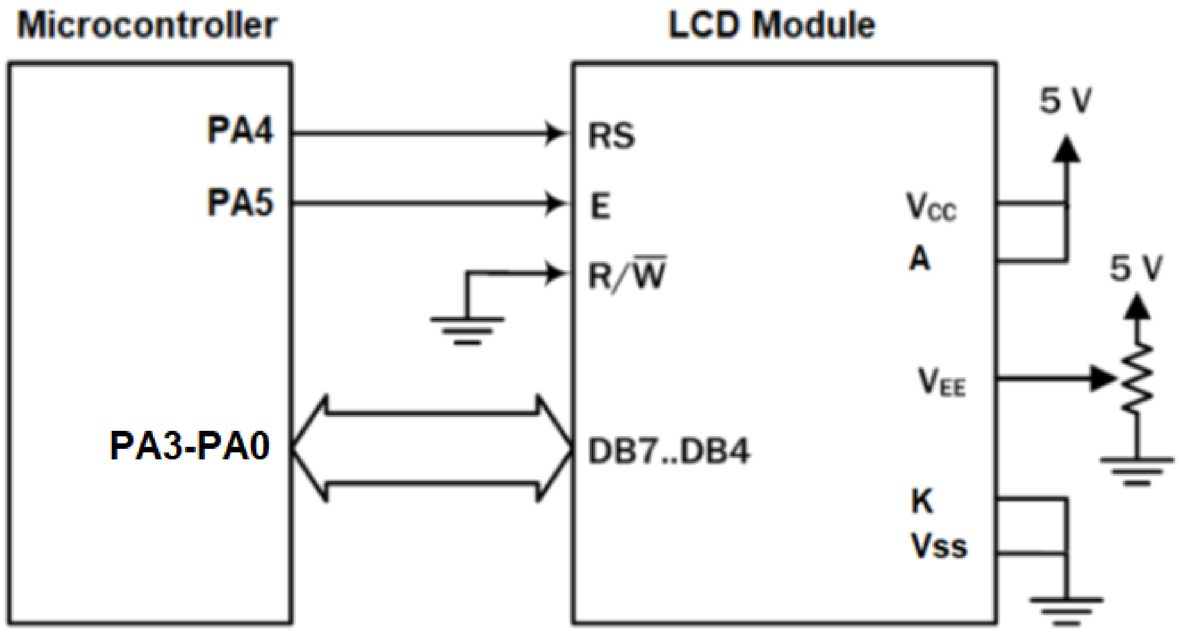
- Sets the interface length (DL bit) to be 4- or 8-bit.
  - Using the 4 bit mode is more complex
    - Data must be sent twice (upper 4 bits are sent first then lower 4 bits)
  - Save on number of used ports/pins
  - Chip starts in 8 bit mode
  - 4 bit mode data transfers on the upper data pins, D4-D7.
- Selects the number of lines (N bit) to be one or two lines.
- selects character font (F bit) to be 5x8 or 5x10

Address Regs & Flag

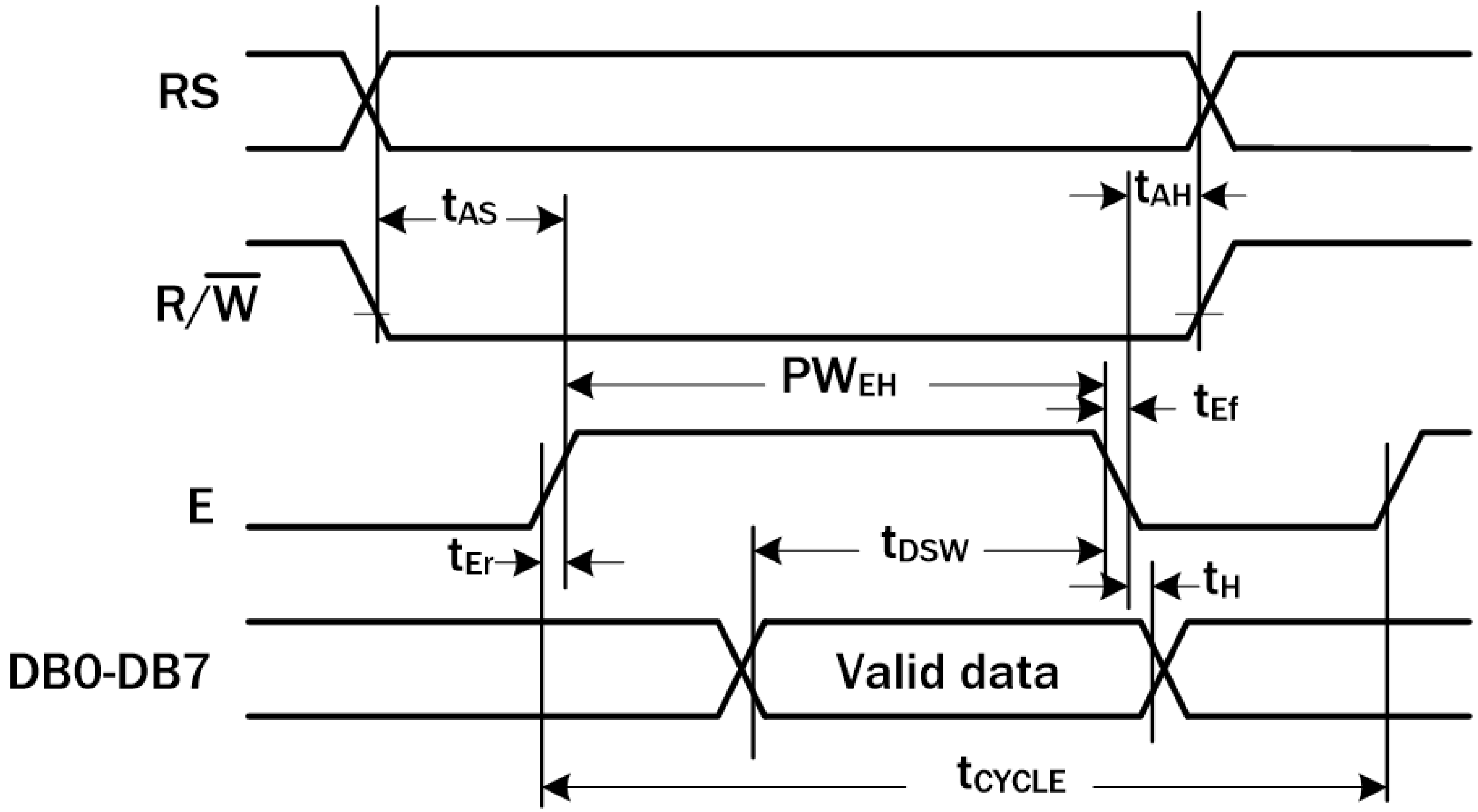
- Set CGRAM Address
  - This command contains the address to be written into the address counter.
- Set DDRAM Address
  - This command allows the user to set the starting address to display information.
- Read Busy Flag and Address
  - This command reads the busy flag and the address counter.
  - User can use this command to determine whether the LCD controller is ready to accept another command.
  - User can use this command to control where to start displaying information.

Setup

- LCD kit as an I/O device
- Use 4 bit-interface to save on pins



Timing: Write to LCD



Writing an Instruction to IR

- Process:
  1. IR instruction write: RS = 0
  2. Write: R/W = 0
     - Based on diagram on previous slide R/W is always 0 => ignore this step
  3. Enable input: E = 1
  4. Output port = instruction
     - Configure IO Port for output before
  5. Disable input: E = 0
- 4-bit interface requires sending the upper bits first, then the lower 4-bits

Writing a Character to LCD

- Process:
  1. DR Character write: RS = 1
  2. Write operation: R/W = 0
  3. Enable input: E = 1
  4. Output port = instruction
     - Configure IO Port for output before
  5. Disable input: E = 0
- 4-bit interface requires sending the upper bits first, then the lower 4-bits

Coding Examples

- Create the following functions:
  - `byte2LCD`:
    - Send a byte of data to the LCD
    - Bytes can be instructions or characters to display
    - Bytes are sent in 2 cycles: upper 4 bits, lower 4 bits
  - `setupLCD`: sends instructions to the LCD kit:
    - Uses `byte2LCD`
    - 4-bit data, 2-line display, 5x8 font
    - turn on display, display a blinking cursor
    - Set display to display left to right
    - Clear display, move cursor home
  - `str2LCD`: displays a string on LCD
    - Uses `byte2LCD`

Definitions

```
#define F_CPU 16000000L        // Specify 16MHz frequency
#include <avr/io.h>            // GPIO reg def
#include <util/delay.h>        // delay function

#define  LCD_DAT  PORTA        // Port A drives LCD data pins, E & RS
#define  LCD_DIR  DDRA         // Direction of LCD port
#define  LCD_E    0x20         // E signal
#define  LCD_RS1  0x10         // RS signal: 0 for instr, 1 for char
#define  LCD_RS0  0x00         // RS signal: 0 for instr, 1 for char
```

byte2LCD

```
void byte2LCD (char data, char IR_DR)
{       char temp;
        LCD_DAT = IR_DR;                // set instr or character
        temp = data >> 4;               // get upper 4 bits
        LCD_DAT = IR_DR|LCD_E;          // pull E signal to high
        _delay_ms(5000);
        temp = temp|IR_DR;
        temp = temp|LCD_E;
        LCD_DAT = temp;                 // output upper four bits, E, and RS
        _delay_ms(1);                   // wait until the send is complete

        LCD_DAT = IR_DR;                // set instr or character
        temp = data & 0x0F;             // keep lower 4 bits
        temp = temp|LCD_E;
        LCD_DAT = IR_DR|LCD_E;          // pull E to high
        temp = temp|IR_DR;
        LCD_DAT = temp;                 // output upper four bits, E, and RS
        _delay_ms(1);                   // wait until the command is complete
        LCD_DAT = 0;                    // pull E clock to low
}
```

setupLCD

```
void setupLCD(void)
{
        LCD_DIR = 0xFF;                 // configure LCD_DAT port for output
        byte2LCD(0x28, LCD_RS0);        // set 4-bit data, 2-line display, 5x8 font
        byte2LCD(0x0F, LCD_RS0);        // turn on display, cursor, blinking
        byte2LCD(0x06, LCD_RS0);        // move cursor right
        byte2LCD(0x01, LCD_RS0);        // clear screen, move cursor to home
        _delay_ms(1);                   // wait until complete
}
```

str2LCD

```
void str2LCD (char *ptr)
{
        while (*ptr) //ascii-Z strings (null-terminated)
        {
                byte2LCD(*ptr, LCD_RS1);
                ptr++;
        }
}
```

Example Program

```
int main (void)
{
        char *msg1 = "hello world!";
        char *msg2 = "LCD is working!";

        setupLCD();
        str2LCD(msg1);
        byte2LCD(0xC0,LCD_RS0);        // move cursor to 2nd row, 1st column
        str2LCD(msg2);
        while(1);
}
```