STM32F3 Microcontroller

GPIO Applications

27/02/2014

TECNOLÓGICO DE MONTERREY®

Dr. Cuauhtémoc Carbajal

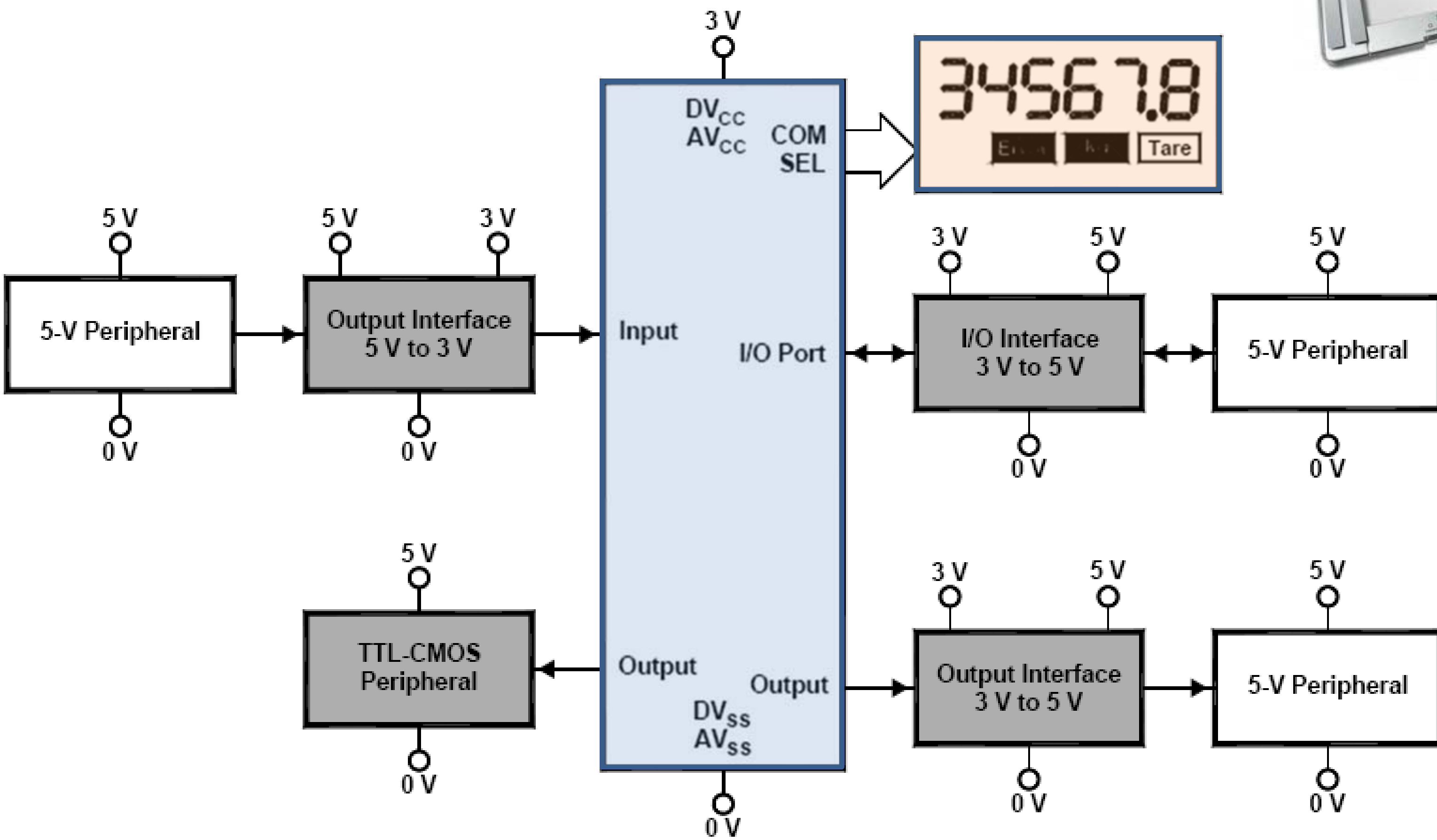
Agenda

- 3V-5V interfacing
- STM32F3 Electrical Characteristics
- I/O Device Categories
- GPIO Interfacing to External Devices

The need for interfacing between 3V and 5V systems

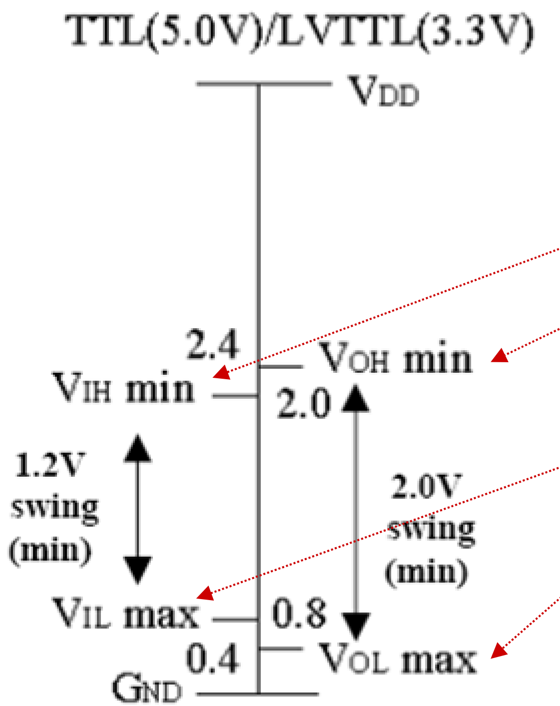
- Many reasons exist to introduce 3V systems, notably the lower power consumption for mobile applications and the introduction of parts that use technologies with such fine geometries that 5V is simply not allowed any more.
- There is a gradual transition from 5V to 3V, since not always are all required components available, or the system is rather complex so that 3V is introduced in part of a system.
- Because many devices still are 5V devices, and most modern sensors, displays, and flash cards are 3V-only, many makers find that they need to perform level shifting/conversion.
- We obviously want a reliable signal transfer from the 5V system to the 3V system and vice versa. This implies that the output voltages should be such that the input levels are satisfied.

The need for interfacing…



Interfaces Between a 3V Microcontroller and 5V Systems

Noise margins

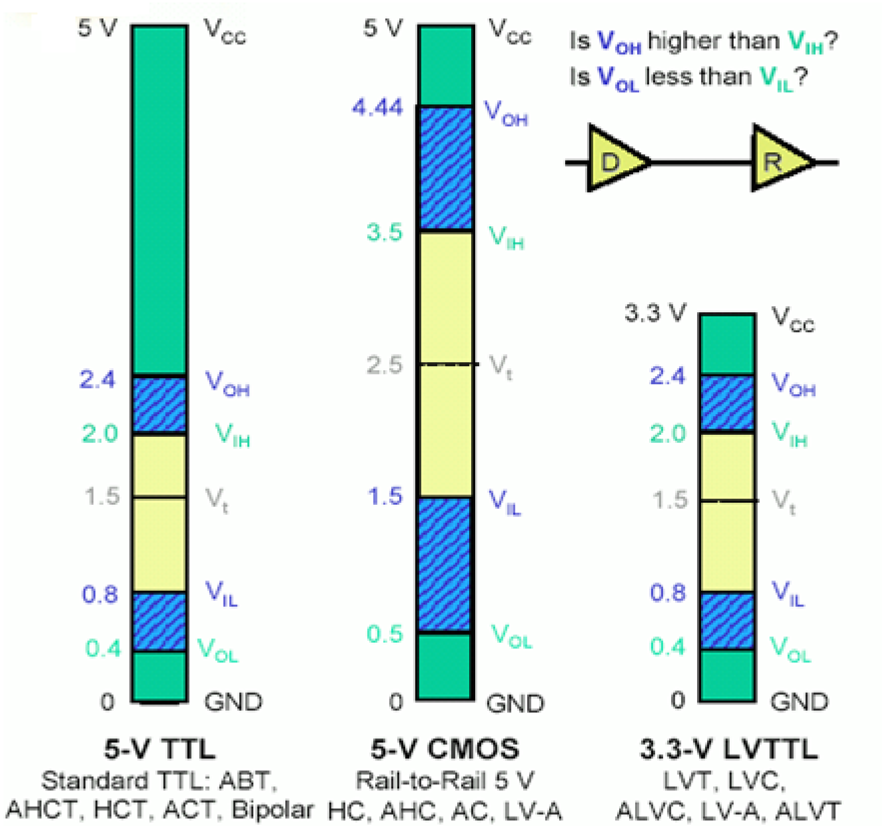


Important characteristics are:

- V_{IHmin} min value input recognized as a ‘1’
- V_{OHmin} min value of output generated as a ‘1’
- V_{ILmax} max value of input recognized as a ‘0’
- V_{OLmax} max value of output generated as a ‘0’
- Values outside the given range are not allowed.

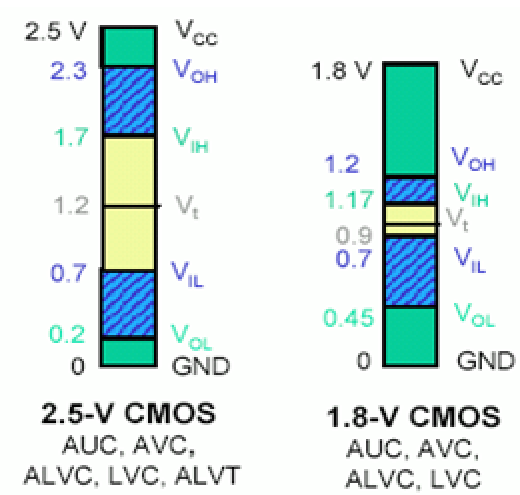
TTL and CMOS Switching Levels

- Digital circuits normally come in two versions:
  - TTL levels: V_{IL} = 0.8V, V_{IH} = 2.0V
  - CMOS levels: V_{IL}= 0.3*V_{CC}, V_{IH} = 0.7*V_{CC}.



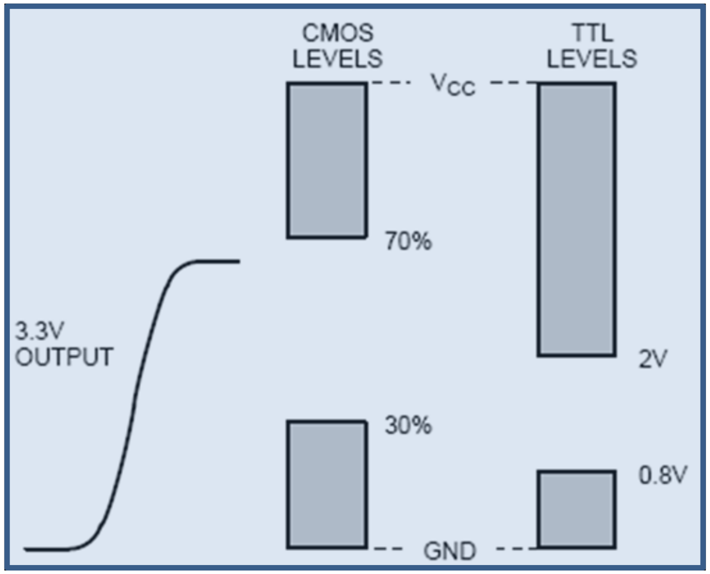
R / D	5TTL	5CMOS	3LVTTL	2.5CMOS	1.8CMOS
5TTL	Yes	No	Yes *	Yes*	Yes*
5 CMOS	Yes	Yes	Yes*	Yes*	Yes*
3 LVTTL	Yes	No	Yes	Yes*	Yes*
2.5 CMOS	Yes	No	Yes	Yes	Yes*
1.8 CMOS	No	No	No	No	Yes

* Requires V_{IH} Tolerance



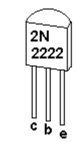
Level Shifting

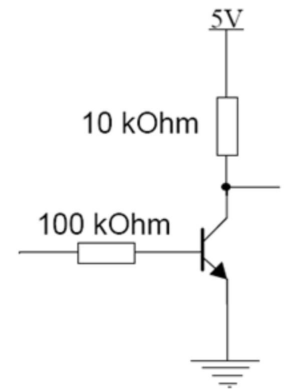
- 5V to 3V
  - All 5V families have an output voltage swing that is large enough to drive 3V reliably. Outputs may be as high as 3.5V for many TTL output stages, to the full 5V for many CMOS outputs. Therefore, as far as switching levels are concerned, there are no problems in interfacing from 5V to a 3V system.
- 3V to 5V
  - All 3V logic families deliver practically the full output voltage swing of 3V, so they can drive TTL switching levels without problems.
  - However, a 3V system cannot reliably drive a 5V one that has CMOS input levels, even when using pull-up resistors.

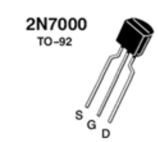


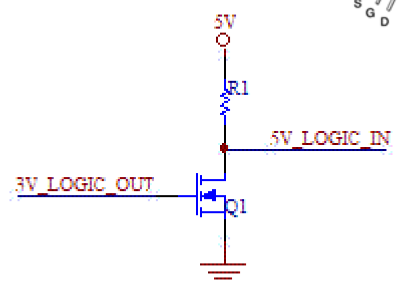
Voltage level conversion methods

- Discrete
  - You can do voltage level conversion using discrete bipolar transistors.
  - The circuit shown converts from a voltage swing of 0-3V to a voltage swing of 0-5V.
  - The resistor values may have to be modified depending on the switching speed required. At the same time, the resistor connected to the collector should be as large as possible in low power applications, since static current will be drawn when the output from the circuit is low.
  - Also note that this circuit inverts, i.e. it will drive the output low when the input is high and vice versa.
  - You can use the same circuit with an NMOS, but in this case you do not need a resistor in series with the gate. Make sure that you select a transistor with an appropriate threshold voltage.





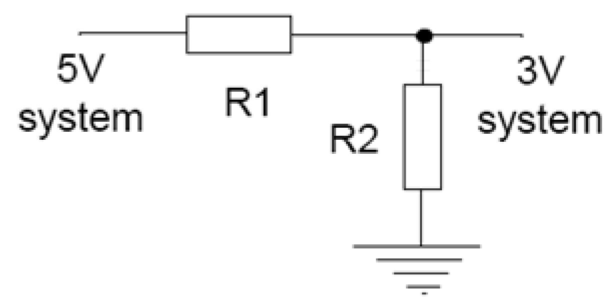




http://www.daycounter.com/Circuits/Level-Translators/Level-Translators.phtml

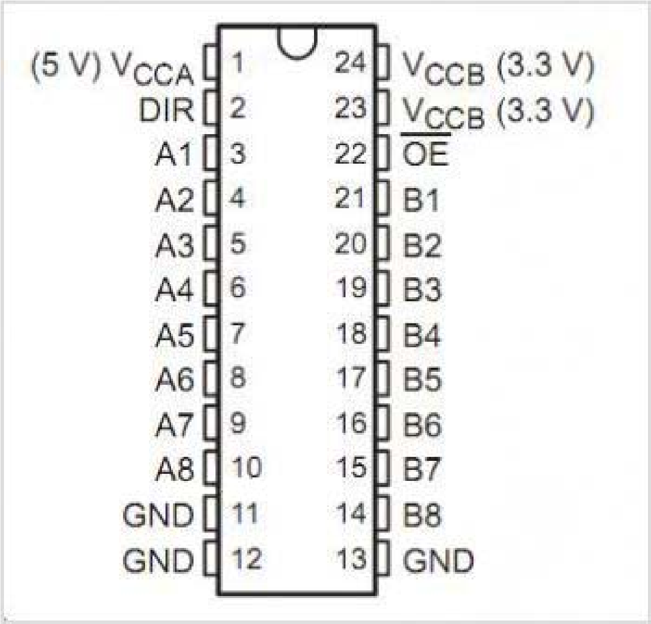
Voltage level conversion methods

- Passive voltage divider
  - If you are only concerned with avoiding violation of the absolute maximum ratings of the 3V circuit, you can use a resistor voltage divider to divide down the 5V signal to 3V. With an appropriate choice of resistors, this will work fine, but it will draw static current all the time.
  - Typical resistor values for the figure below can be R1=22kΩ, R2=33kΩ. If the 5V device has a low enough threshold voltage that it will function with a 3V input voltage, this can be a good approach for bi-directional signals, as the voltage divider only divides down the voltage in one direction.
  - Note: Can work on bi-directional signals if 5V system has low enough threshold voltage (V_{IH5V}<V_{OH3V})



Voltage level conversion methods

- Dual VCC level shifters
  - The 74LVC4245 and 74ALVC164245 (8 and 16 bits resp.) are CMOS transceivers fed from both 3V and 5V supplies. The level shifting is done internally and the parts have full output voltage swings at both sides, making them ideal for level shifting purposes, especially when driving 5V CMOS levels.
  - Dual VCC level shifters are superior alternatives to the sometimes used input pull-up resistors, blocking diodes and other circuits that normally degrade speed and/or noise margins.
  - Only problem is that they are only good in one direction which can be a problem for some specialty bi-directional interfaces and also makes wiring a little hairy.



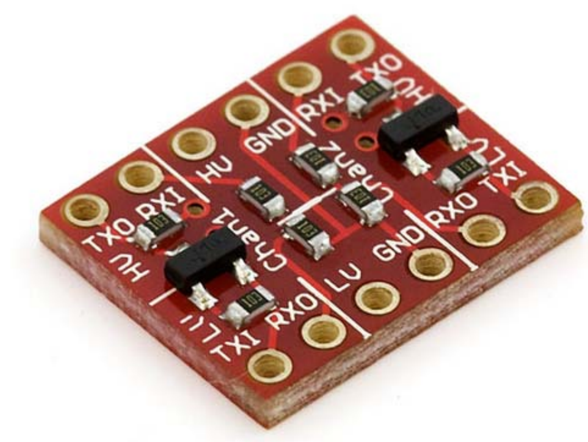
FUNCTION TABLE

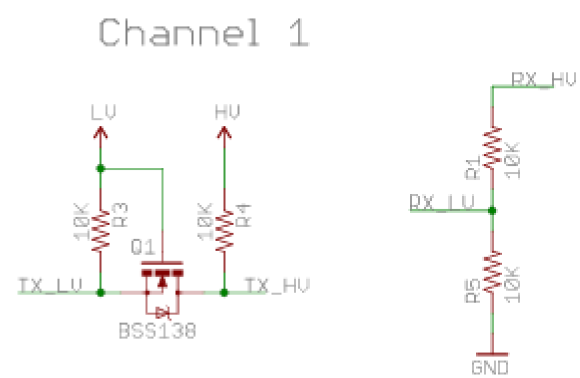
INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

swra071.pdf

Voltage level conversion methods

- Logic Level Converter BOB-08745
  - It safely steps down 5V signals to 3.3V and steps up 3.3V to 5V. This level converter also works with 2.8V and 1.8V devices. Each level converter has the capability of converting 4 pins on the high side to 4 pins on the low side. Two inputs and two outputs are provided for each side.
  - Can be used with normal serial, I2C, SPI, and any other digital signal. It does not work with an analog signal.





https://www.sparkfun.com/products/8745

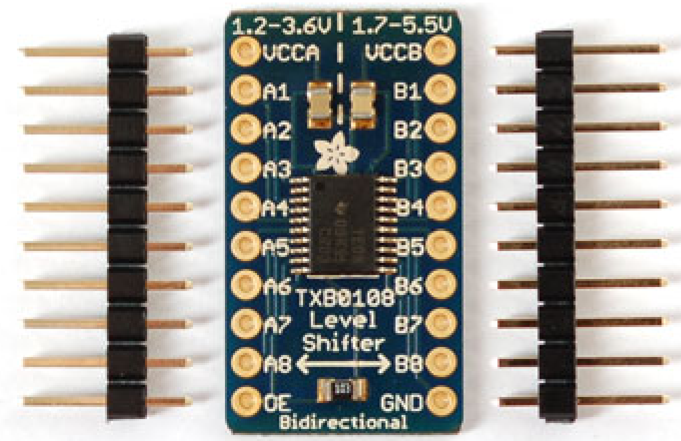
http://www.adafruit.com/datasheets/BSS138.pdf

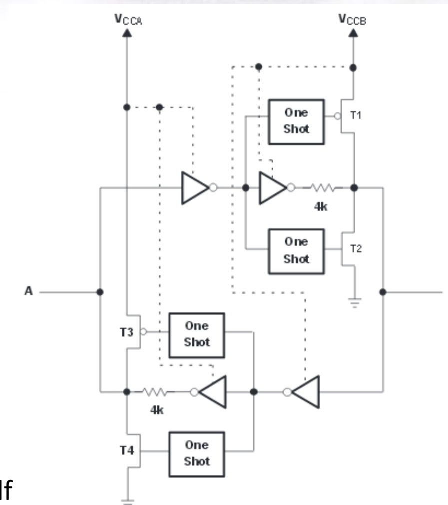
Voltage level conversion methods

- 8-channel Bi-directional Logic Level Converter - TXB0108
  - This chip perform bidirectional level shifting from pretty much any voltage to any voltage and will **auto-detect** the direction. Only thing that doesn't work well with this chip is i2c (because it uses strong pullups which confuse auto-direction sensor).
  - If you need to use pullups, you can but they should be at least 50K ohm - the ones internal to the STM32F3 are about 100K ohm so those are OK!

3V side

5V side



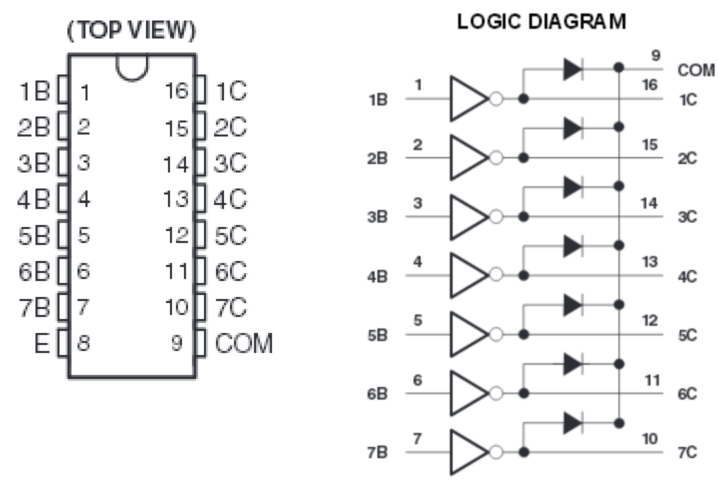


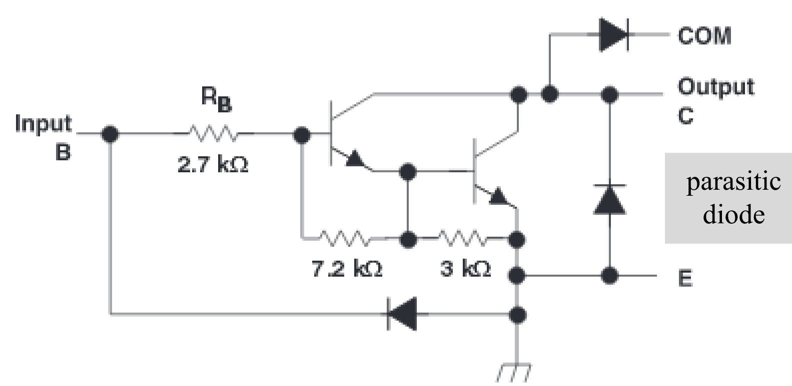
http://www.adafruit.com/products/395

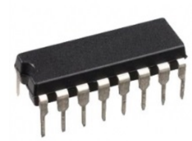
http://www.ti.com/lit/ds/symlink/txb0108.pdf

Voltage level conversion methods

- ULN2003A are high-voltage high -current Darlington transistor arrays. It consists of seven NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads.
- The collector-current rating of a single Darlington pair is 500 mA.
- The Darlington pairs can be paralleled for higher current capability.

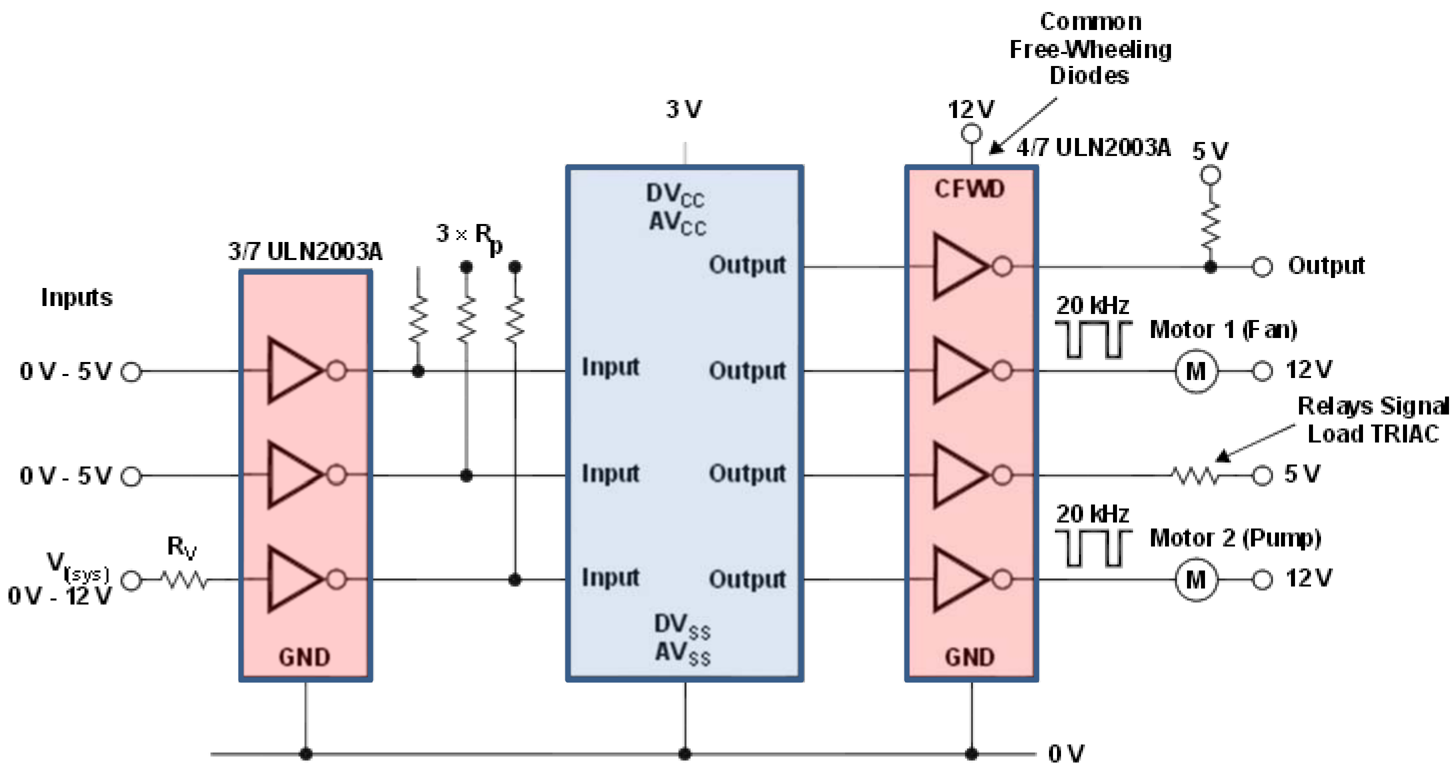






Voltage level conversion methods

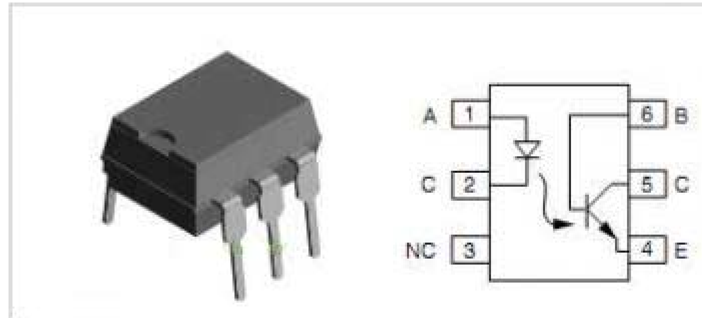
- ULN2003A applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers.



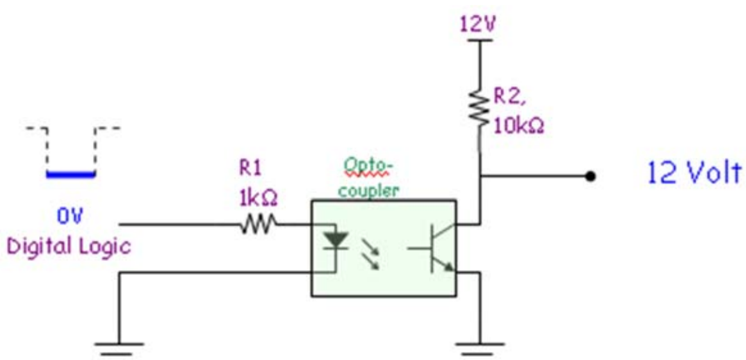
Interfaces With High-Current Output Buffers ULN2003

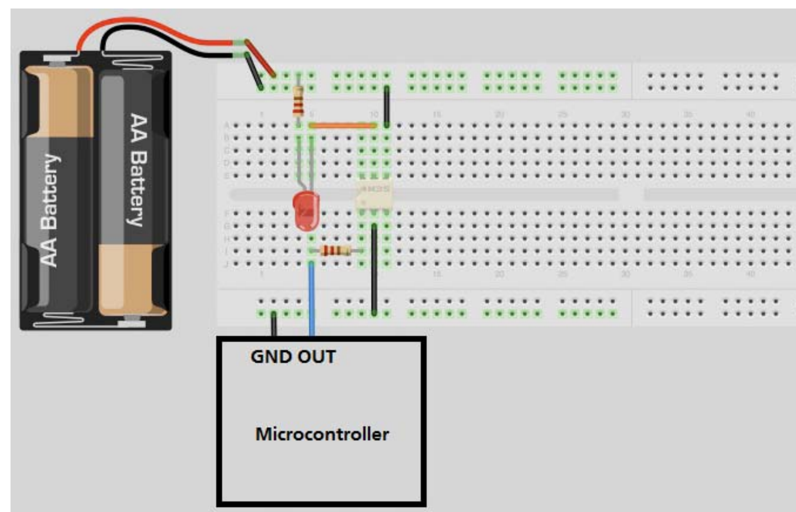
Voltage level conversion methods

- Optocoupler
  - It can be used to control a circuit that is completely isolated from your uC.
  - In this case, imagine that the LED and battery pack are a hacked toy which you're turning on and off with the uC.



4N28





http://www.siongboon.com/projects/2006-06-19_switch/

STM32F3
I/O structure

PIN	Str	PIN	Str	PIN	Str	PIN	Str	PIN	Str	PIN	Str
PA0	TTa	PB0	TTa	PC0	TTa	PD0	FT	PE0	FT	PF0	FT
PA1	TTa	PB1	TTa	PC1	TTa	PD1	FT	PE1	FT	PF1	FT
PA2	TTa	PB2	TTa	PC2	TTa	PD2	FT	PE2	FT	PF2	TTa
PA3	TTa	PB3	FT	PC3	TTa	PD3	FT	PE3	FT	PF3	
PA4	TTa	PB4	FT	PC4	TTa	PD4	FT	PE4	FT	PF4	TTa
PA5	TTa	PB5	FT	PC5	TTa	PD5	FT	PE5	FT	PF5	
PA6	TTa	PB6	FTf	PC6	FT	PD6	FT	PE6	FT	PF6	FTf
PA7	TTa	PB7	FTf	PC7	FT	PD7	FT	PE7	TTa	PF7	
PA8	FT	PB8	FTf	PC8	FT	PD8	TTa	PE8	TTa	PF8	
PA9	FTf	PB9	FTf	PC9	FT	PD9	TTa	PE9	TTa	PF9	FT
PA10	FTf	PB10	TTa	PC10	FT	PD10	TTa	PE10	TTa	PF10	FT
PA11	FT	PB11	TTa	PC11	FT	PD11	TTa	PE11	TTa	PF11	
PA12	FT	PB12	TTa	PC12	FT	PD12	TTa	PE12	TTa	PF12	
PA13	FT	PB13	TTa	PC13	TC	PD13	TTa	PE13	TTa	PF13	
PA14	FTf	PB14	TTa	PC14	TC	PD14	TTa	PE14	TTa	PF14	
PA15	FTf	PB15	TTa	PC15	TC	PD15	TTa	PE15	TTa	PF15	

Free I/O

5V tolerant I/O

Name	Abbreviation	Definition
I/O structure	FT	5 V tolerant I/O
	FTf	5 V tolerant I/O, FM+ capable
	TTa	3.3 V tolerant I/O directly connected to ADC
	TC	Standard 3.3V I/O
	B	Dedicated BOOT0 pin
	RST	Bidirectional reset pin with embedded weak pull-up resistor
Notes	Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset	

Fast mode: up to 400 kHz
Fast mode plus: up to 1 MHz

STM32F3 Absolute maximum ratings

- Voltage Characteristics

Symbol	Ratings	Min	Max	Unit
$V_{DD}–V_{SS}$	External main supply voltage (including $V_{DDA,}$ V_{BAT} and V_{DD})	-0.3	4.0	V
$V_{DD}–V_{DDA}$	Allowed voltage difference for $V_{DD} > V_{DDA}$		0.4	
V_{IN}	Input voltage on FT and FTf pins	$V_{SS} – 0.3$	$V_{DD} + 4.0$	
	Input voltage on TTa pins	$V_{SS} – 0.3$	4.0	
	Input voltage on any other pin	$V_{SS} – 0.3$	4.0	

STM32F3DISCOVERY BOARD:
VSS : 0 volts
VDD: 3 volts

STM32F3 Absolute maximum ratings

- Current characteristics

Symbol	Ratings	Max.	Unit
I_{VDD}	Total current into V_{DD} and VDDSDx power lines (source)(2)	TBD	mA
I_{VSS}	Total current out of V_{SS} and VSSSD ground lines (sink)(2)	TBD	
$I_{IO(PIN)}$	Output current sunk by any I/O and control pin	25	
	Output current source by any I/O and control pin	– 25	
$\Sigma I_{IO(PIN)}$	Total output current sunk by sum of all IOs and control pins	75	
	Total output current sourced by sum of all IOs and control pins	75	
$I_{INJ(PIN)}$	Injected current on FT, FTf and B pins(3)	-5/+0	
	Injected current on TC and RST pin(4)	± 5	
	Injected current on TTa pins(5)	± 5	
$\Sigma I_{INJ(PIN)}$	Total injected current (sum of all I/O and control pins)(6)	± 25	

I/O port characteristics

- General input/output characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Standard I/O input low level voltage		–0.3	-	$0.3V_{DD}+0.07$	V
	TTa I/O input low level voltage		–0.3	-	$0.3V_{DD}+0.07$	
	FT and FTf[1] I/O input low level voltage		–0.3	-	$0.475V_{DD}-0.2$	
	BOOT0 input low level voltage		0	-	$0.3V_{DD}–0.3$	
V_{IH}	Standard I/O input high level voltage		$0.445V_{DD}+0.398$	-	$V_{DD}+0.3$	
	TTa I/O input high level voltage		$0.445V_{DD}+0.398$	-	$V_{DD}+0.3$	
	FT and FTf[1] I/O input high level voltage		$0.5V_{DD}+0.2$	-	5.5	
	BOOT0 input high level voltage		$0.2V_{DD}+0.95$	-	5.5	

I/O port characteristics

- Output voltage characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
V_{OL}(1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	CMOS port(2) $I_{IO} = +8$ mA $2.7\ V < V_{DD} < 3.6\ V$	-	0.4	V
V_{OH}(3)	Output high level voltage for an I/O pin when 8 pins are sourced at same time		$V_{DD}-0.4$	-	
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	TTL port(2) $I_{IO} =+ 8$mA $2.7\ V < V_{DD} < 3.6\ V$	-	0.4	
V_{OH} (3)	Output high level voltage for an I/O pin when 8 pins are sourced at same time		2.4	-	
V_{OL}(1)(4)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	$I_{IO} = +20$ mA $2.7\ V < V_{DD} < 3.6\ V$	-	1.3	
V_{OH}(3)(4)	Output high level voltage for an I/O pin when 8 pins are sourced at same time		$V_{DD}-1.3$	-	
V_{OL}(1)(4)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	$I_{IO} = +6$ mA $2\ V < V_{DD} < 2.7\ V$	-	0.4	
V_{OH}(3)(4)	Output high level voltage for an I/O pin when 8 pins are sourced at same time		$V_{DD}-0.4$	-	
V_{OLFM+}	Output low level voltage for an FTf I/O pin in FM+ mode	$I_{IO} = +20$ mA $2.7\ V < V_{DD} < 3.6\ V$	-	0.4	

I/O Device Categories

- Input devices
  - Sensors, User-input
- Output devices
  - Actuators, Displays
- Complex I/O devices (printers, faxes, coprocessors, etc.)

Analog I/O issues

- Voltage levels
- Current draw
- Sampling frequency
- Noise

Digital I/O issues

- Voltage levels
- Synchronization
- Throughput
- Noise

Input Examples

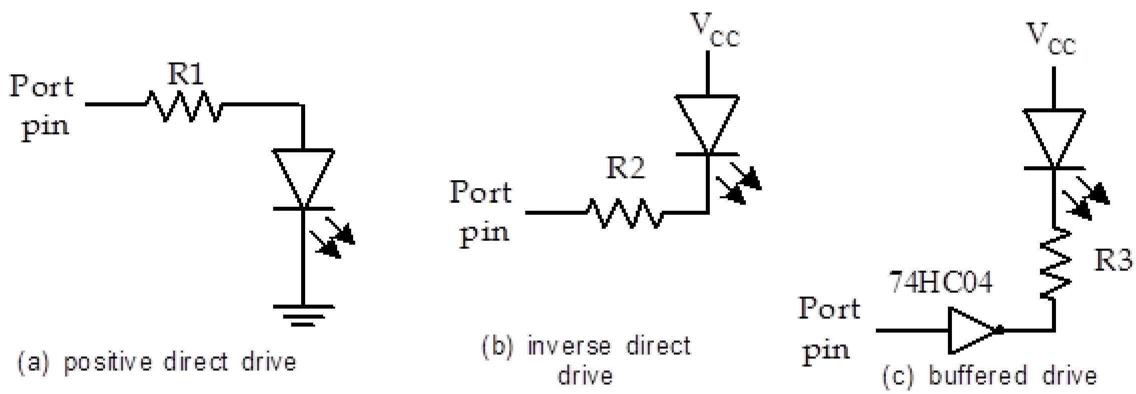
- Sensors
  - light
  - force
  - sound
  - position
  - orientation
  - proximity
  - tactile
  - temperature
  - pressure
  - humidity
  - speed
  - acceleration
  - displacement
- User input
  - keyboards
  - joysticks
  - mouse
  - keypad
  - switches
  - touchpad
  - dial
  - slider

Output Examples

- Actuators
  – motors
  – solenoids
  – relays
  – heaters
  – lights
  – piezoelectric materials (buzzers, linear actuator)
  – speakers
- Displays
  – LED displays
  – LCD displays
  – CRT displays
  – indicator lights
  – indicator gauges

Interfacing with LED Devices

- The figure below suggests three methods for interfacing with LEDs.
- Circuit (a) and (b) are recommended for LEDs that need only small current to light.
- Circuit (c) is recommended for LEDs that need larger current to light.



$V_{OL,\,min} = 0.15V$

An LED connected to a CMOS inverter through a current-limiting resistor

- LED

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Luminous Intensity	Iv	If=20mA	20	45		mcd
Forward Voltage	Vf	If=20mA		1.8	2.2	V
Peak Wavelength	λ P	If=20mA		660		nm

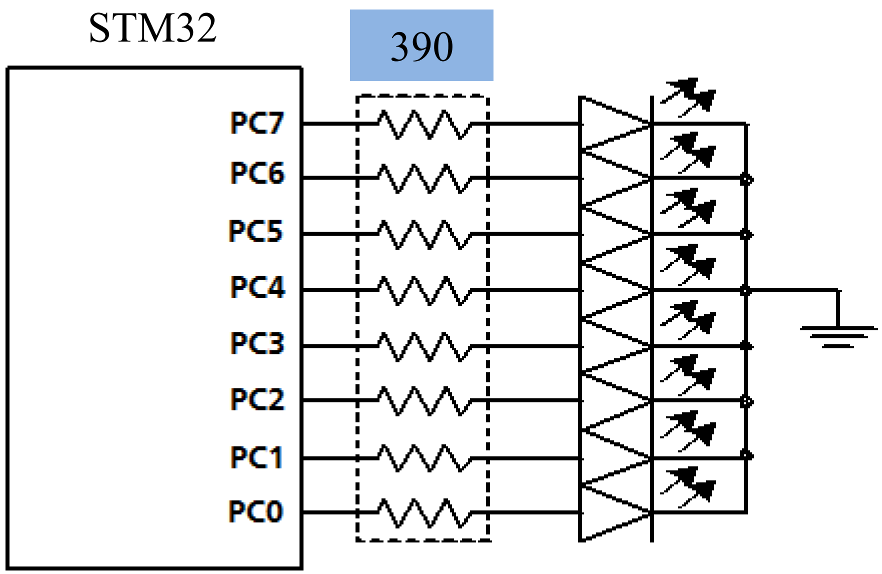
http://www.farnell.com/datasheets/553533.pdf

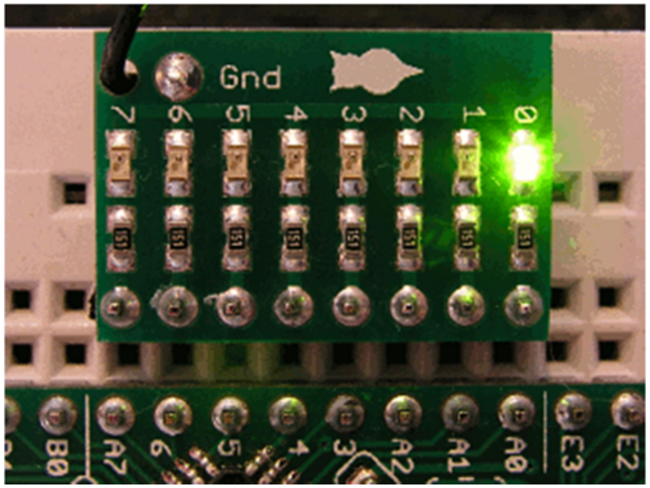
- 74HC04

Symbol	Parameter	Conditions	25 °C			–40 °C to +85 °C		–40 °C to +125 °C		Unit
			Min	Typ	Max	Min	Max	Min	Max	
V_{OL}	LOW-level output voltage	$V_I = V_{IH}$ or V_{IL}								
		$I_O = 20\ \mu A$; $V_{CC} = 2.0\ V$	-	0	0.1	-	0.1	-	0.1	V
		$I_O = 20\ \mu A$; $V_{CC} = 4.5\ V$	-	0	0.1	-	0.1	-	0.1	V
		$I_O = 20\ \mu A$; $V_{CC} = 6.0\ V$	-	0	0.1	-	0.1	-	0.1	V
		$I_O = 4.0\ mA$; $V_{CC} = 4.5\ V$	-	0.15	0.26	-	0.33	-	0.4	V
		$I_O = 5.2\ mA$; $V_{CC} = 6.0\ V$	-	0.16	0.26	-	0.33	-	0.4	V

http://www.farnell.com/datasheets/1645614.pdf

Example: Use PC[7:0] to drive eight LEDs using the circuit shown in the Figure below. Light each LED for half a second in turn and repeat assuming the GPIOC has a 72-MHz clock.





PART	COLOR	LUMINOUS INTENSITY (mcd)			at I_F (mA)	WAVELENGTH (nm)			at I_F (mA)	FORWARD VOLTAGE (V)			at I_F (mA)	TECHNOLOGY
		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		
TLLR4400	Red	0.63	1.2	-	2	612	-	625	2	-	1.9	2.4	2	GaAsP on GaP

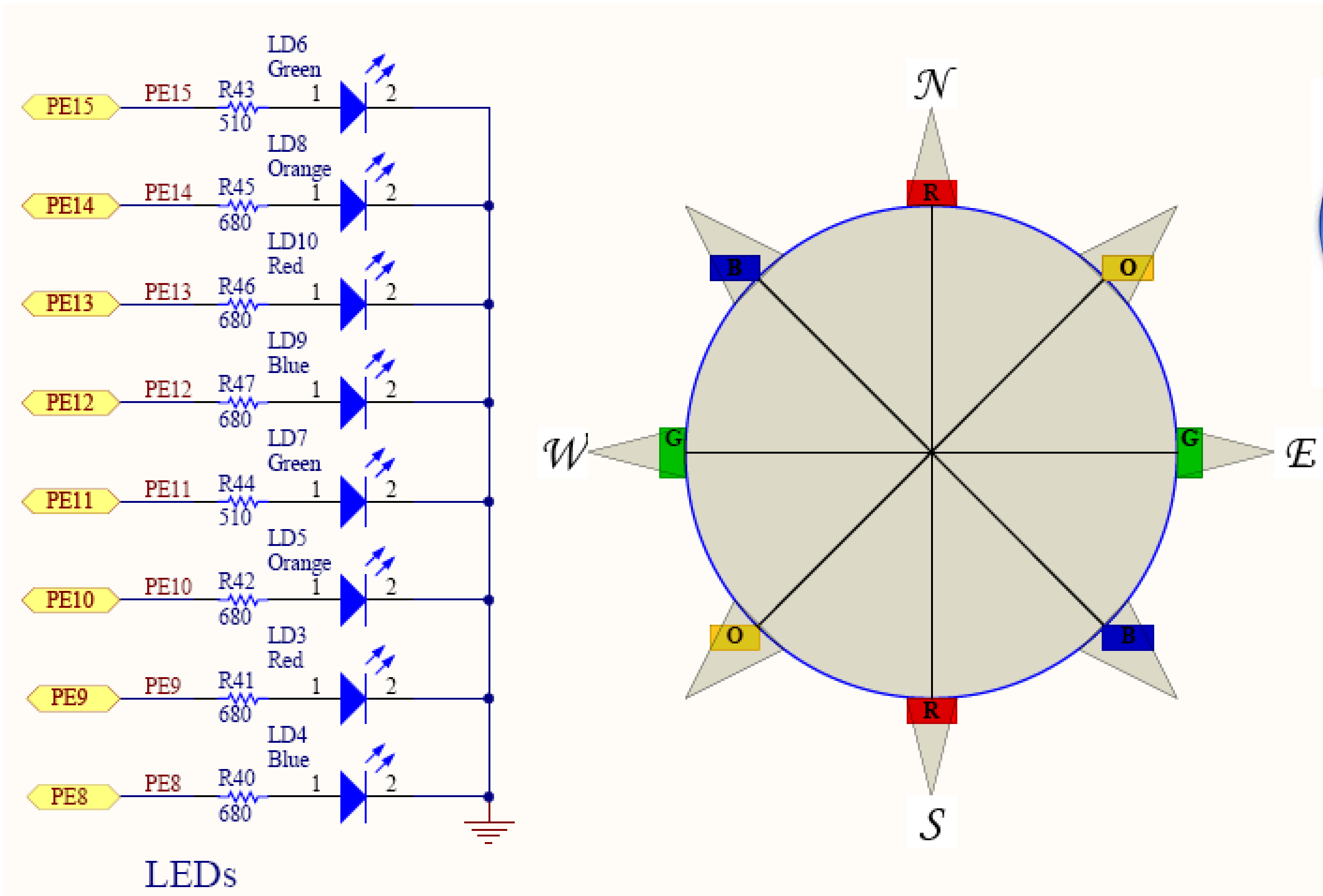
To turn on one LED at a time for half a second in turn, one should output the value \$80, \$40, \$20, \$10, \$08,\$04,\$02, and \$01 and stay for half a second in each value.

The C language version of the program is as follows:

```c
#include "stm32f30x.h"

void delaybyms(unsigned int j);

int main(void) {
  unsigned char led_tab[] = {0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01,
                             0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
  char i=0;

  RCC->AHBENR  |= RCC_AHBENR_GPIOCEN; // Enable GPIOC clock

  // PC[7:0] configuration
  GPIOC->MODER   = GPIOC->MODER   & 0xFFFF0000 | 0x00005555; // 0b01: Output
  GPIOC->OTYPER  = GPIOC->OTYPER  & 0xFFFFFF00;              // 0b0 : PP (R)
  GPIOC->OSPEEDR = GPIOC->OSPEEDR & 0xFFFF0000 | 0x0000FFFF; // 0b11: 50MHz
  GPIOC->PUPDR   = GPIOC->PUPDR   & 0xFFFF0000;              // 0b00: no PU/PD (R)

  while (1) {
    for (i = 0; i < 16; i++) {
      GPIOC->ODR = GPIOC->ODR & 0xFFFFFF00 | led_tab[i];
      msdelay(500);
    }
  }
}

void delaybyms(unsigned int j) {
    unsigned int k,l;
    for(k=0;k<j;k++)
    for(l=0;l<1427;l++);
}
```

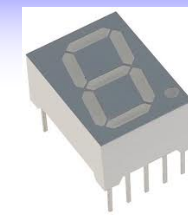
gpio_test.c

STM32F3DISCOVERY BOARD: SCHEMATIC DETAIL

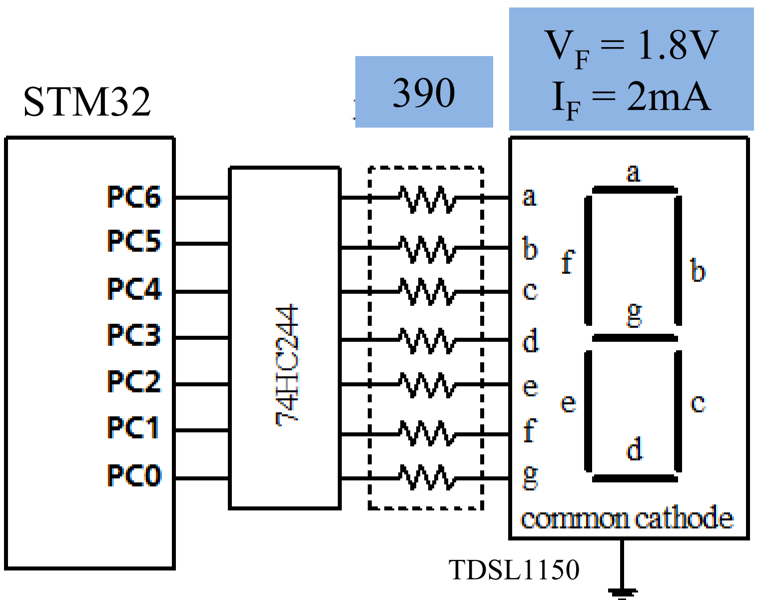


http://homepage.cem.itesm.mx/carbajal/Microcontrollers/RESOURCES/STM32F3DISCOVERY/stm32f3discovery_sch/MB1035.pdf

Driving a Single Seven-Segment Display



- A common cathode seven-segment display is driven by the 74HC244 via resistors.
- The output high voltage of the 74HC244 is close to 3V with a 3V power supply.
- The segment patterns for 0 to 9 are shown in the table below.



Driving a single seven-segment display

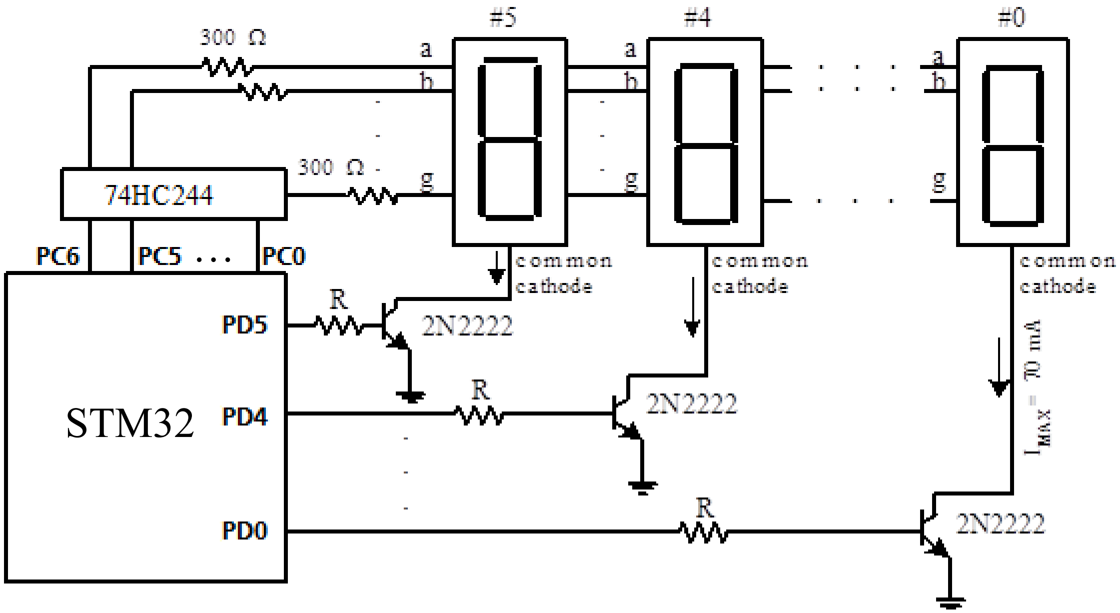
BCD digit	Segments a	b	c	d	e	f	g	Corresponding Hex Number
0	1	1	1	1	1	1	0	$7E
1	0	1	1	0	0	0	0	$30
2	1	1	0	1	1	0	1	$6D
3	1	1	1	1	0	0	1	$79
4	0	1	1	0	0	1	1	$33
5	1	0	1	1	0	1	1	$5B
6	1	0	1	1	1	1	1	$5F
7	1	1	1	0	0	0	0	$70
8	1	1	1	1	1	1	1	$7F
9	1	1	1	1	0	1	1	$7B

BCD to seven-segment decoder

PART	COLOR	LUMINOUS INTENSITY (μcd)			at I_F (mA)	WAVELENGTH (nm)			at I_F (mA)	FORWARD VOLTAGE (V)			at I_F (mA)	CIRCUITRY
		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		
TDSL1150	Red	180	260	-	2	612	-	625	2	-	1.8	2.4	2	Common anode
TDSL1160	Red	180	260	-	2	612	-	625	2	-	1.8	2.4	2	Common cathode

Driving Multiple Seven-Segment Displays

- Time multiplexing technique is often used to drive multiple displays in order to save I/O pins.
- One parallel port is used to drive the segment pattern and the other port turns on one display at a time. Each display is turned on and then off many times within a second. The persistence of vision make us feel that all displays are turned on simultaneously.



Port C and Port D together drive six seven-segment display

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
$V_{CE\ (sat)}$*	Collector-emitter Saturation Voltage	I_C = 150 mA I_B = 15 mA I_C = 500 mA I_B = 50 mA			0.4 1.6	V V
$V_{BE\ (sat)}$*	Base-emitter Saturation Voltage	I_C = 150 mA I_B = 15 mA I_C = 500 mA I_B = 50 mA			1.3 2.6	V V
		for **2N2219** and **2N2222** I_C = 0.1 mA V_{CE} = 10 V I_C = 1 mA V_{CE} = 10 V I_C = 10 mA V_{CE} = 10 V I_C = 150 mA V_{CE} = 10 V I_C = 500 mA V_{CE} = 10 V I_C = 150 mA V_{CE} = 1 V	 35 50 75 100 30 50		 300	
f_T	Transition Frequency	I_C = 20 mA V_{CE} = 20 V f = 100 MHz	250			MHz

Example: Write a sequence of instructions to display **4** on the seven-segment display **#4** in the figure above.

Solution: To display the digit 4 on the display #4, we need to:

- Output the hex value $33 to port C
- Set the PD4 pin to 1
- Clear pins PD5 and PD3...PD0 to 0

In C language:

```
// Configure PC[6:0] & PD[5:0] as GP output + PP
GPIOC->ODR = GPIOD->ODR & 0xFFFFFF80 | 0x33;
GPIOD->ODR = GPIOD->ODR & 0xFFFFFFC0 | 0x10;
```

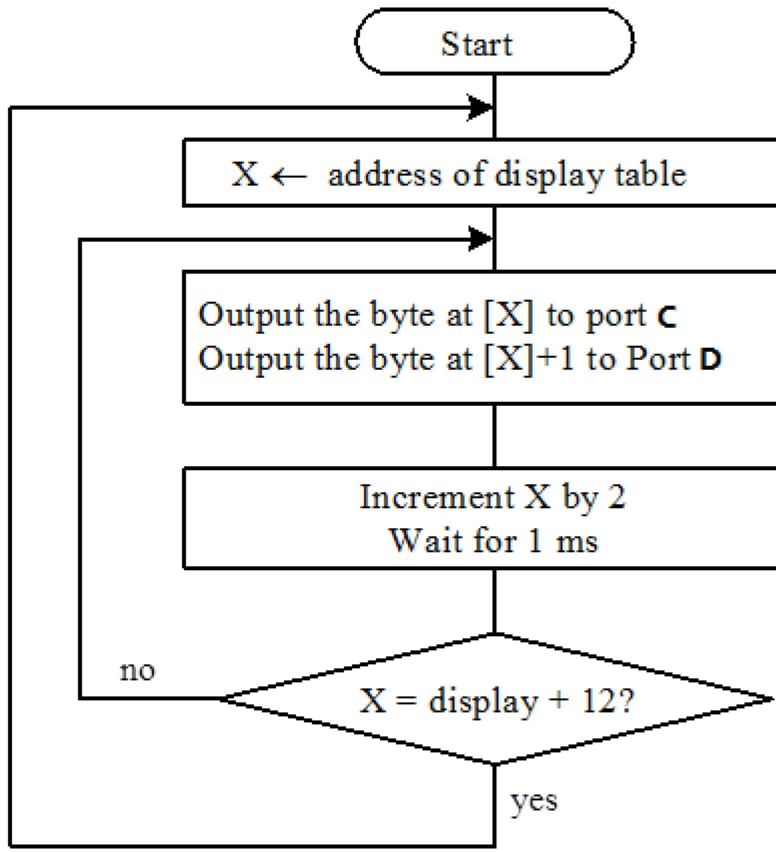
Example: Write a program to display 123456 on the six seven-segment displays shown in the figure below.

Solution: Display 123456 on display #5, #4, #3, #2, #1, and #0, respectively.

- The values to be output to Port C and Port D to display one digit at a time is shown in the table below.

seven-segment display	displayed BCD digit	PortB	PortK
#5	1	$30	$20
#4	2	$6D	$10
#3	3	$79	$08
#2	4	$33	$04
#1	5	$5B	$02
#0	6	$5F	$01

Table of display patterns for this example



Time-multiplexed seven-segment display algorithm

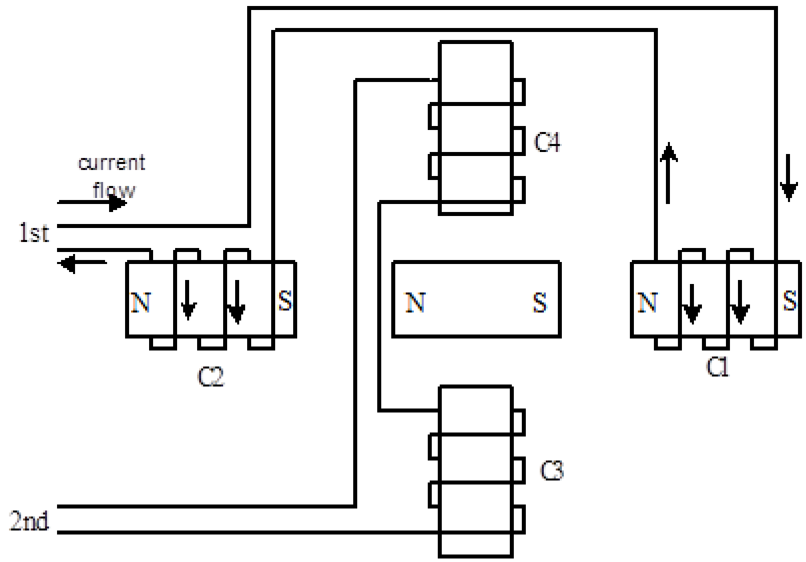
```c
int main (void)
{
  char disp_tab[6][2] = {{0x30,0x20},{0x6D,0x10},{0x79,0x08},
                         {0x33,0x04},{0x5B,0x02},{0x5F,0x01}};
  char i;
  // configure PC[6:0] for output
  // configure PD[5:0] for output
  while (1) {
    for (i = 0; i < 6; i++) {
      // output the segment pattern
      GPIOC->ODR = GPIOC->ODR & 0xFFFFFF80 | disp_tab[i][0];
      // turn on the display
      GPIOD->ODR = GPIOD->ODR & 0xFFFFFFC0 | disp_tab[i][1];
      delaybyms(1);
    }
  }
}
```

Stepper Motor Control (1 of 7)

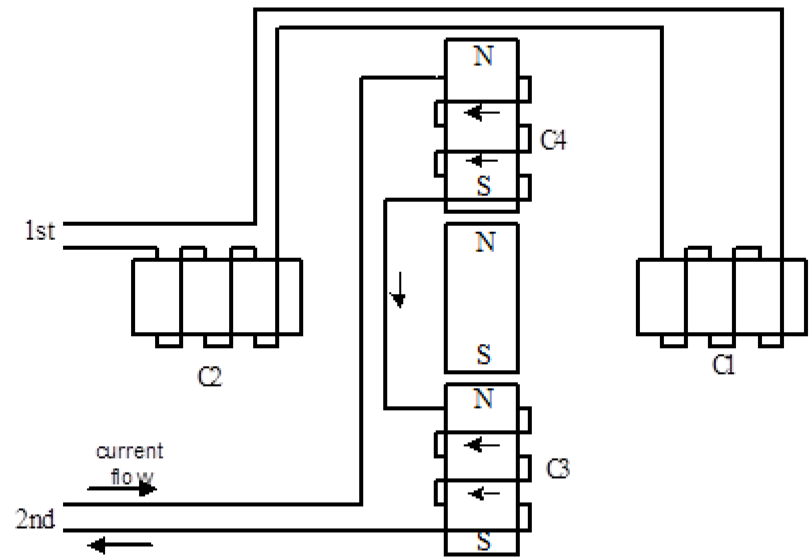
- It is digital in nature and provides high degree of control.
- In its simplest form, a stepper motor has a permanent magnet rotor and a stator consisting of two coils. The rotor aligns with the stator coil that is energized.
- By changing the coil that is energized, the rotor is turned.
- Next four figures illustrate how the rotor rotates clockwise in full step.

http://homepage.cem.itesm.mx/carbajal/Microcontrollers/ASSIGNMENTS/labs/pas.swf

Stepper Motor Control (2 of 7)



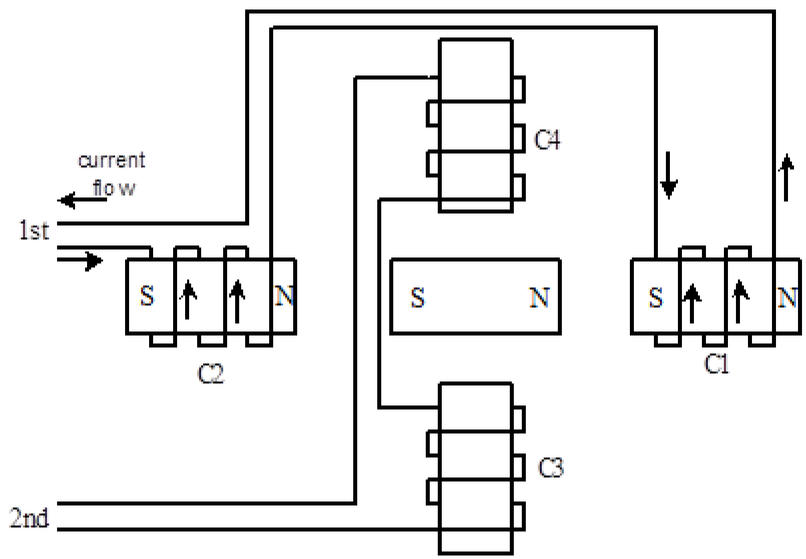
Stepper motor full step 1



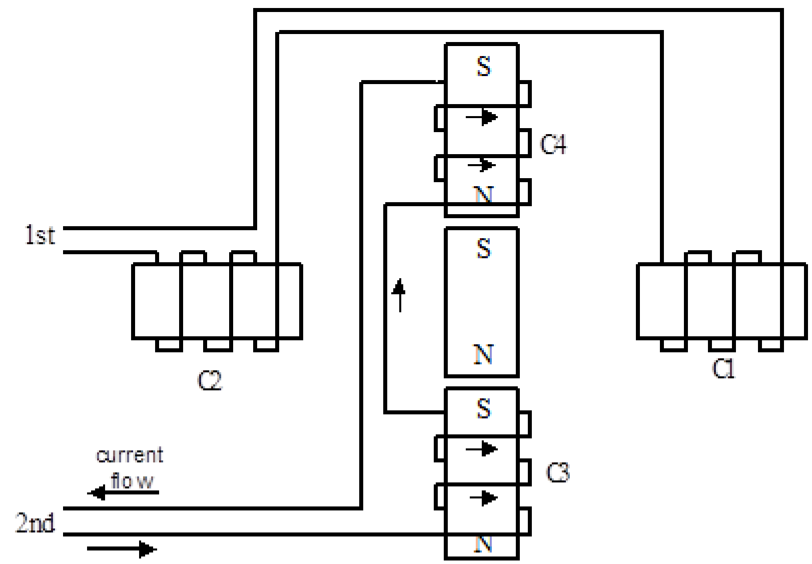
Stepper motor full step 2

Stepper motor clockwise rotation in full step (1 of 2)

Stepper Motor Control (3 of 7)



Stepper motor full step 3

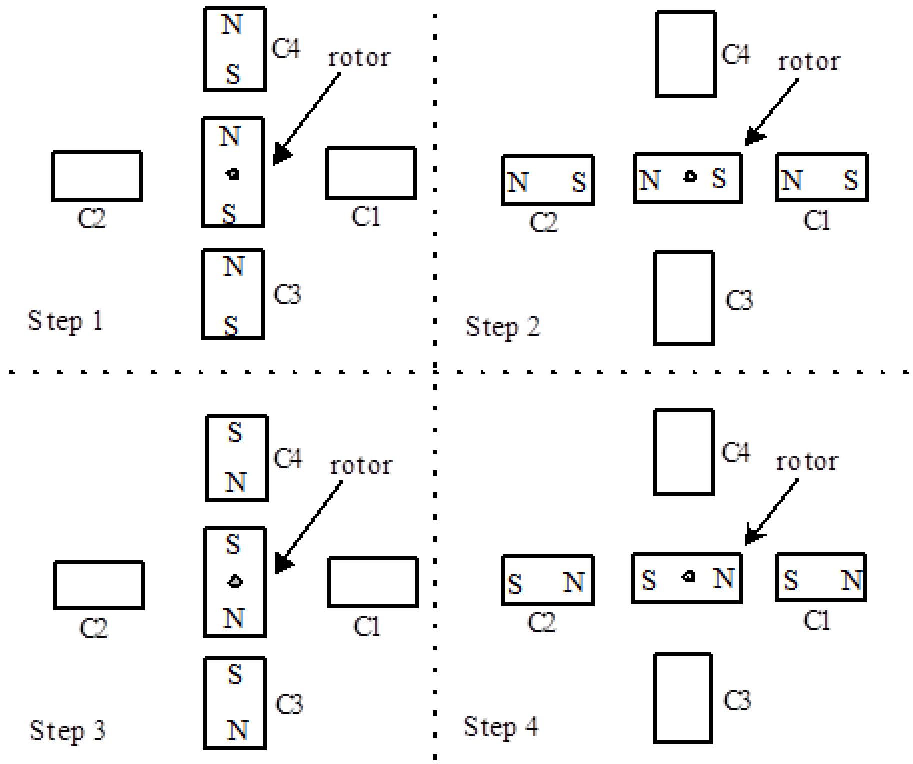


Stepper motor full step 4

Stepper motor clockwise rotation in full step (2 of 2)

Stepper Motor Control (4 of 7)

- Next figure illustrates how the rotor rotates counter-clockwise in full step.

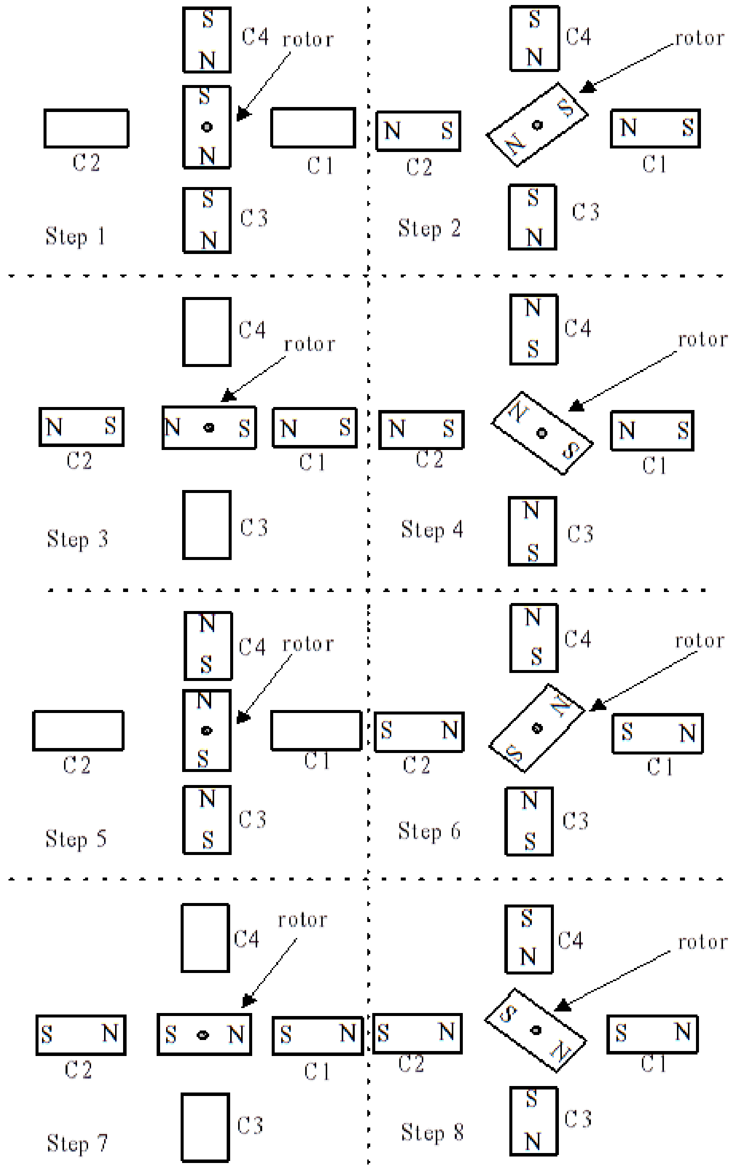


Full-step counter-clockwise operation of step motor

Stepper Motor Control (5 of 7)

- In a four-pole stepper motor shown before, a full step is 90 degrees.
- The stepper motor may also operate with half step. A *half step* occurs when the rotor (in a four-pole step) is moved to eight discrete positions (45º).
- To operate the stepper motor in half steps, sometimes both coils may have to be on at the same time. When two coils in close proximity are energized, there is a resultant magnetic field whose center will depend on the relative strengths of the two magnetic fields.
- The next figure illustrates the half-stepping sequence.

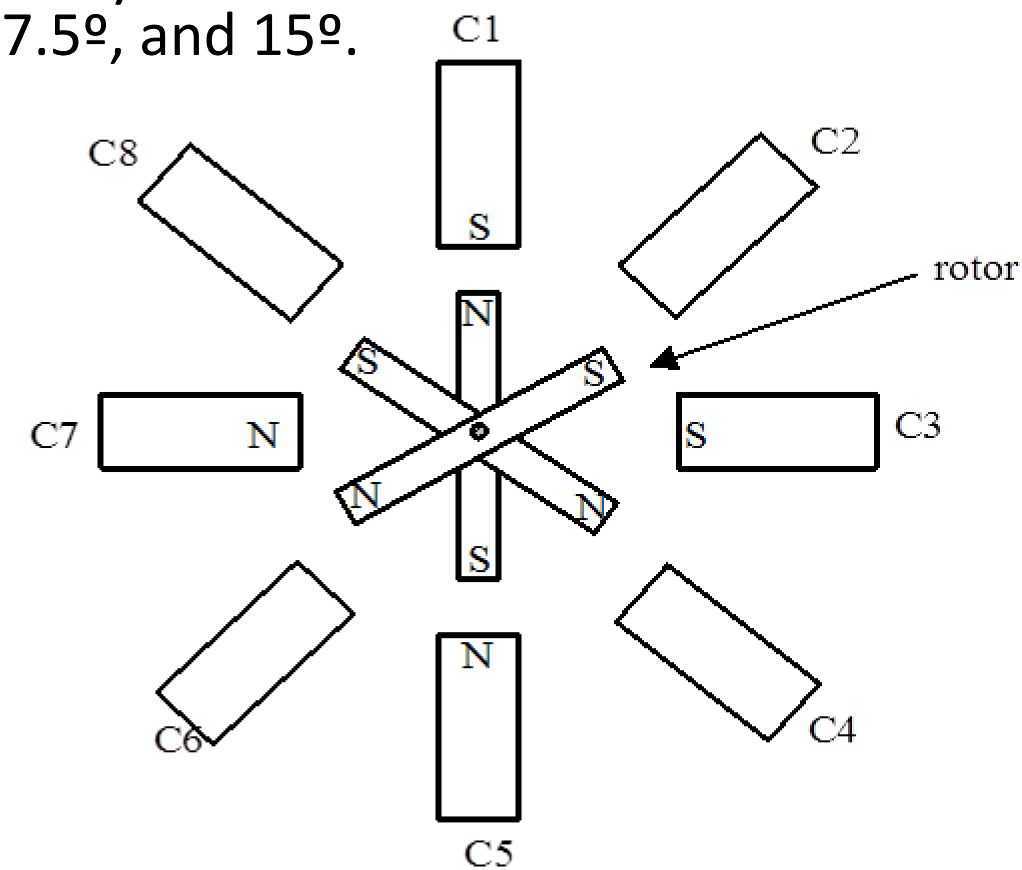
Stepper Motor Control (6 of 7)



Half-step operation of the stepper motor

Stepper Motor Control (7 of 7)

The actual stator of a real motor has more segments than previously indicated. One example is shown in the next figure. The step sizes of the stepper motors may vary from approximately 0.72º to 90º. The most common step sizes are 1.8º, 7.5º, and 15º.

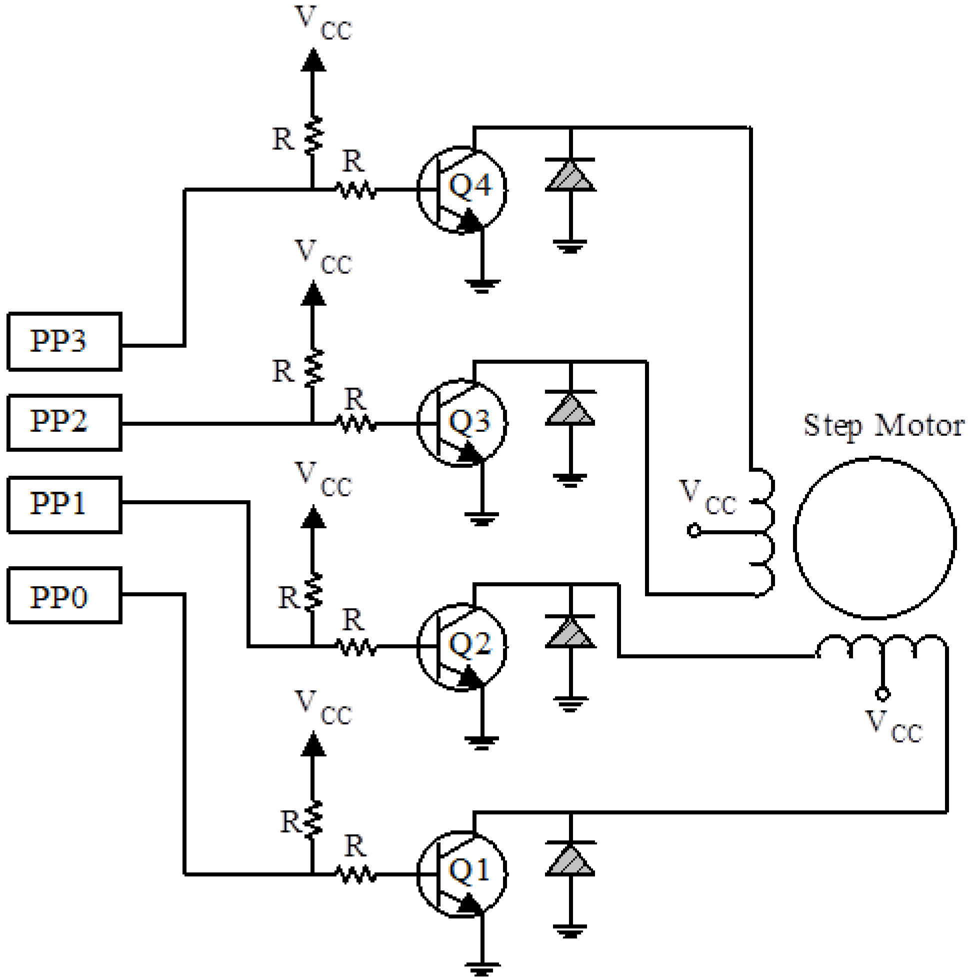


Actual internal construction of stepper motor

Stepper Motor Drivers (1 of 4)

- Driving a step motor involves applying a series of voltages to the coils of the motor.
- A subset of coils is energized at a time to cause the motor to rotate one step. The pattern of coils energized must be followed exactly for the motor to work correctly.
- A microcontroller can easily time the duration that the coil is energized, and control the speed of the stepper motor in a precise manner.
- The circuit in the figure below shows how the transistors are used to switch the current to each of the four coils of the stepper motor.
- The diodes in figure below are called ***fly back diodes*** and are used to protect the transistors from reverse bias.
- The transistor loads are the windings in the stepper motor. The windings are inductors, storing energy in a magnetic field.
- When the current is cut off, the inductor dispenses its stored energy in the form of an electric current.
- This current attempts to flow through the transistor, reversely biasing its collector-emitter pair. The diodes are placed to prevent this current from going through the transistors.

Stepper Motor Drivers (2 of 4)



Driving the stepper motor

Stepper Motor Drivers (3 of 4)

- The normal full-step sequence should be used for high-torque applications.
- For lower-torque applications the half-step mode is used.
- The microcontroller outputs the voltage pattern in the sequence shown in these tables.
- The tables are circular. The values may be output in the order as shown in the table, which will rotate the motor clockwise; or in the reverse order, which will rotate the motor counterclockwise.
- A delay of about 5 to 15ms is required between two steps to prevent motor from missing steps.

Full-step sequence for clockwise rotation

Step	Q1 PP0	Q2 PP1	Q3 PP2	Q4 PP3	value
1	on	off	on	off	1010
2	on	off	off	on	1001
3	off	on	off	on	0101
4	off	on	on	off	0110
1	on	off	on	off	1010

Half-step sequence for clockwise rotation

Step	Q1 PP0	Q2 PP1	Q3 PP2	Q4 PP3	value
1	on	off	on	off	1010
2	on	off	off	off	1000
3	on	off	off	on	1001
4	off	off	off	on	0001
5	off	on	off	on	0101
6	off	on	off	off	0100
7	off	on	on	off	0110
8	off	off	on	off	0010
1	on	off	on	off	1010

Stepper Motor Drivers (4 of 4)

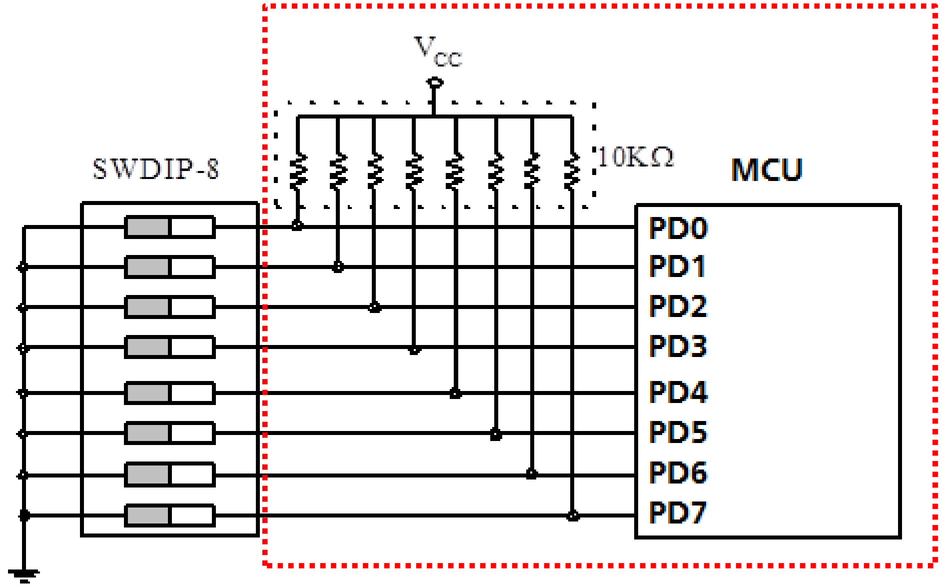
Example: Assuming that pins PP3...PP0 are used to drive the four transistor in the figure above, write a subroutine to rotate the stepper motor clockwise one cycle using the half-step sequence.

```
#include <stm32f30x.h>

const unsigned char sequence[] = {0x05,0x01,0x09,0x08,0x0A,0x02,0x04,0x06};

int main( void ) {
  unsigned int i,j;

  // Configure GPIOC[3:0] as output, PP, high speed, no pullup/pulldown
  RCC->AHBENR    |= 0x00080000; // Activating GPIOC clock source
  GPIOC->MODER   |= 0x00000055; // Configuring GPIOC[3:0] as outputs
  GPIOC->OSPEEDR &= 0xFFFFFFAA; // Selecting low-speed on GPIOC[3:0]
  while (1) {
    for (i=0;i<8;i++) {
      GPIOC->ODR = sequence[i]; // Sequence number output
      for ( j=0;j<10000;j++ );  // Delay loop
    }
  }
}
```

Interfacing with DIP Switches (1 of 2)

- Switches are often grouped together. It is most common to have four or eight switches in a DIP package.
- DIP switches are often used to provide setup information to the microcontroller. After power is turned on, the microcontroller reads the settings of the DIP switches and performs accordingly.

STM32F3
$R_{PULL\text{-}UP}=R_{PULL\text{-}DOWN}=40k\Omega$



Connecting a set of eight DIP switches to port D of the MCU

Interfacing with DIP Switches (2 of 2)

- **Example** Write a sequence of instructions to read the value from an eight-switch DIP connected to GPIOD of the STM32 into accumulator A.
- Solution

In C language

```
void main () {
  char  xx;
  RCC->AHBENR |= 1 << 20;        // Enable GPIOD clock
  GPIOD->MODER &= 0xFFFF0000;    // PD[7:0] as input (reset value)
  GPIOD->PUPDR |= 0x00005555;    // Pull-up resistors
  xx = GPIOD->IDR & 0x000000FF; // clean value
}
```

Hardware Debouncing Techniques

- SR latches
- Non-inverting CMOS gates
- Integrating debouncer

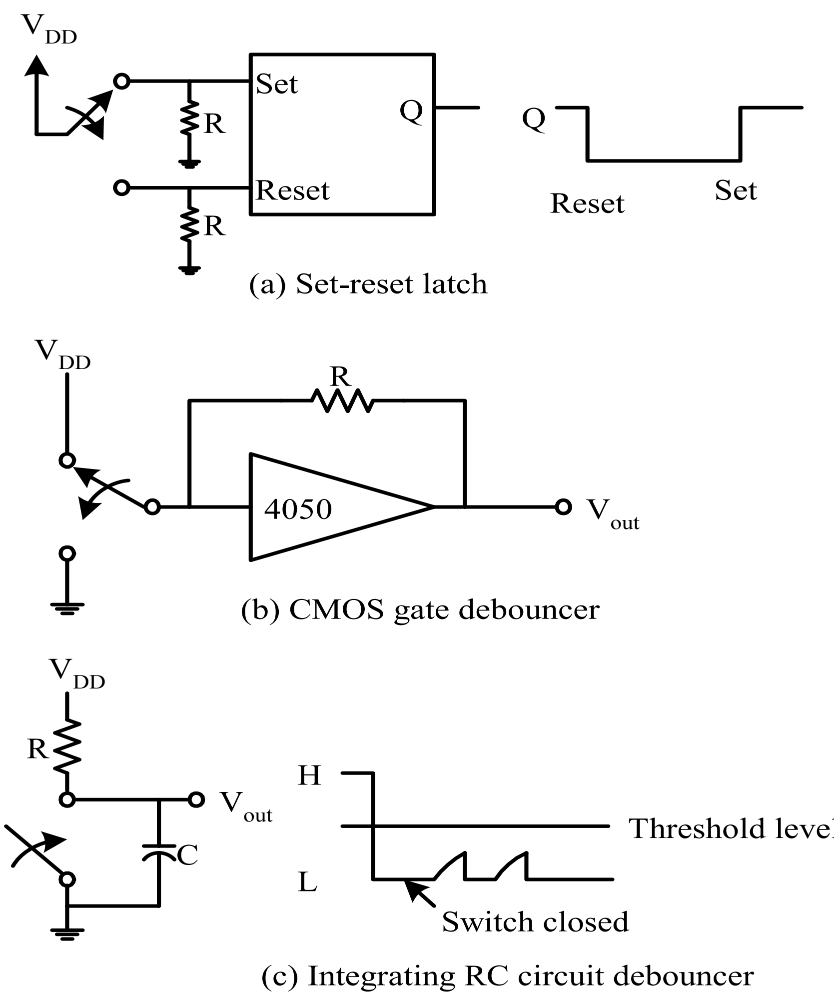
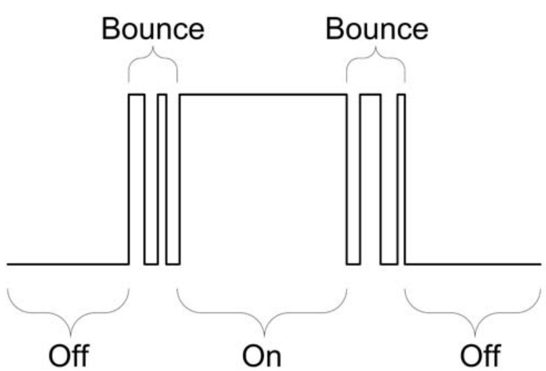


Figure 7.42 Hardware debouncing techniques

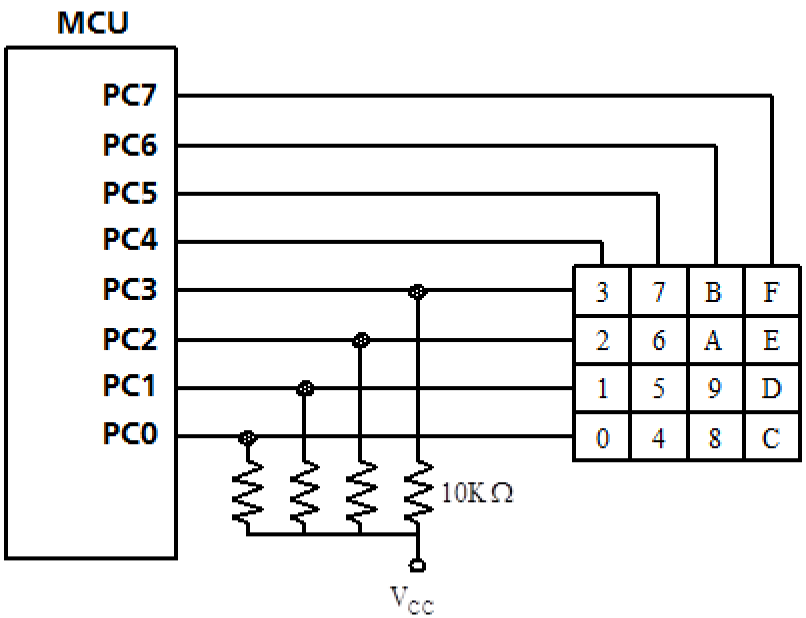
Software Debouncing Technique

- The most popular and simple one has been the **wait and see** method.
  - In this method, the program simply waits for about 10 ms and reexamines the same key again to see if it is still pressed.



Interfacing the MCU to a Keypad

- A keypad usually consists of 12 to 24 keys and is adequate for many applications.
- Like a keyboard, a keypad also needs debouncing.
- A 16-key keypad can be easily interfaced to one of the MCU parallel ports.
- A circuit that interfaces a 16-key keypad is shown in the figure below (left). Note: pins PC[7:4] each control four keys.



Sixteen-key keypad connected to the MCU

PC7	PC6	PC5	PC4	Selected keys
1	1	1	0	0, 1, 2, and 3
1	1	0	1	4, 5, 6, and 7
1	0	1	1	8, 9, A, and B
0	1	1	1	C, D, E, and F

Sixteen-key keypad row selections

```c
#define         keypad          GPIOC->IDR                      // keypad port
#define         keypad_dir      GPIOC->MODER                    // keypad port direction register
// ************************************************************************
// rmask is row mask, cmask is column mask, row is the row being scanned, col is the
// column being scanned
// ************************************************************************
char getkey (void)
{
        char rmask, cmask, row, col;
        char temp, keycode;
        keypad_dir = (keypad_dir &0xFFFF0000) | 0x00005500; // Configure lower bits[3:0] as input, bits[7:4] as output
        while (1) {
                rmask    = 0xEF;
                for (row = 0; row < 4; row++){
                        cmask    = 0x01;
                        GPIOC->ODR &= rmask; // select the current row
                        for (col = 0; col < 4; col++){
                                if (!(keypad & cmask)){            // key switch detected pressed
                                        delayby10ms(1);
                                        if(!(keypad & cmask)){  // check the same key again
                                                keycode = row * 4 + col;
                                                if (keycode < 10)
                                                        return (0x30 + keycode);
                                                else
                                                        return (0x37 + keycode);
                                        }
                                }
                                cmask = cmask << 1;
                        }
                        rmask = (rmask << 1) | 0x0F;
                }
        }
}
```

Adapted from “The HCS12/9S12: An Introduction to Software and Hardware Interfacing”, HUANG (2010)

Example: Write a C program to read a character from the keypad shown in the figure above.
This program will perform keypad scanning, debouncing, and ASCII code lookup.
Solution:

```
void delaybyms (unsigned int); // prototype

char get_key (void) {
  RCC->AHBENR |= 1 << 19; // Enable GPIOC clock
  GPIOC->MODER |= 0x5500; // configure PC[7:4] for output and PC[3:0] for input

  while (1) {

    GPIOC->ODR = 0xE0;      // prepare to scan the row controlled by PC4

    if (!(GPIOC->IDR & 0x01)) {
      delaybyms (10);
      if (!(GPIOC->IDR & 0x01))
        return 0x30; // return ASCII code of 0
    }
    if (!(GPIOC->IDR & 0x02)) {
      delaybyms (10):
      if (!(GPIOC->IDR & 0x02))
        return 0x31; // return ASCII code of 1
    }
```

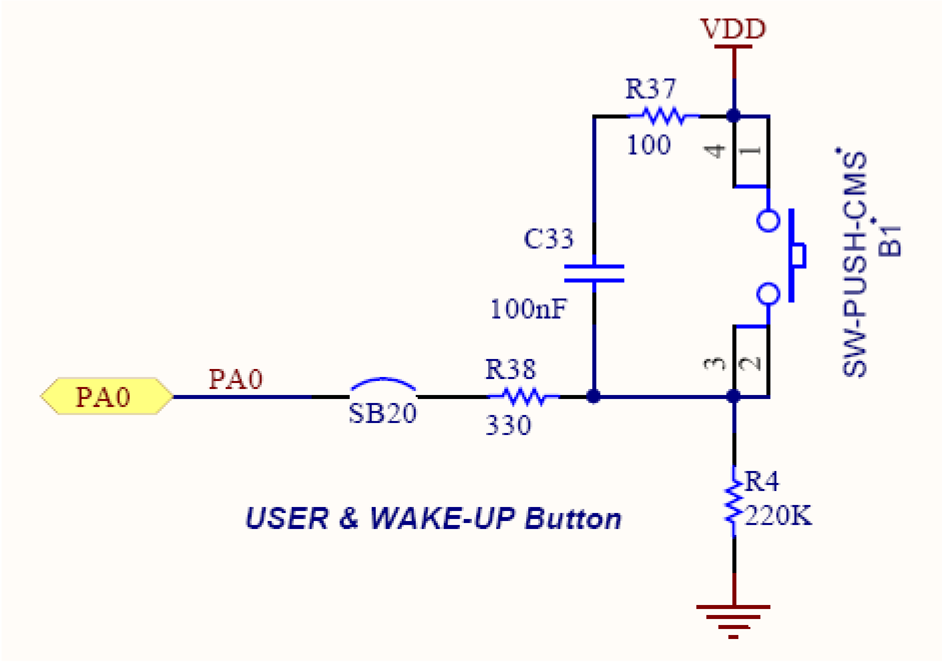
```
if (!(GPIOC->IDR & 0x04)) {
  delaytbyms (10);
  if (!(PORTC & 0x04))
    return 0x32; // return ASCII code of 2
}
if (!(GPIOC->IDR & 0x08)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x08))
    return 0x33; // return ASCII code of 3
}

GPIOC->ODR = 0xD0; // set PC5 to low to scan second row

if (!(GPIOC->IDR & 0x01)) {
  delayby0ms (10);
  if (!(GPIOC->IDR & 0x01))
    return 0x34; // return ASCII code of 4
}
if (!(GPIOC->IDR & 0x02)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x02))
    return 0x35; // return ASCII code of 5
}
```

```
if (!(GPIOC->IDR & 0x04)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x04))
    return 0x36; // return ASCII code of 6
}
if (!(GPIOC->IDR & 0x08)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x08))
    return 0x37; // return ASCII code of 7
}

GPIOC->ODR = 0xB0; // set PC6 to low to scan the third row

if (!(GPIOC->IDR & 0x01)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x01))
    return 0x38; // return ASCII code of 8
}
if (!(GPIOC->IDR & 0x02)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x02))
    return 0x39; // return ASCII code of 8
}
```

```c
if (!(GPIOC->IDR & 0x04)) {
  wait_10ms ( );
  if (!(GPIOC->IDR & 0x04))
    return 0x41;  // return ASCII code of A
}
if (!(GPIOC->IDR & 0x08)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x08))
    return 0x42;  // return ASCII code of B
}

GPIOC->ODR = 0x70;  // set PC7 to low to scan the fourth row

if (!(GPIOC->IDR & 0x01)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x01))
    return 0x43;  // return ASCII code of C
}
if (!(GPIOC->IDR & 0x02)) {
  delaybyms (10);
  if (!(GPIOC->IDR & 0x02))
    return 0x44;  // return ASCII code of D
}
```

```
    if (!(GPIOC->IDR & 0x04)) {
      wait_10ms ( );
      if (!(GPIOC->IDR & 0x04))
        return 0x45;  // return ASCII code of E
    }
    if (!(GPIOC->IDR & 0x08)) {
      delaybyms (10);
      if (!(GPIOC->IDR & 0x08))
        return 0x46;  // return ASCII code of F
    }
  }
}
```

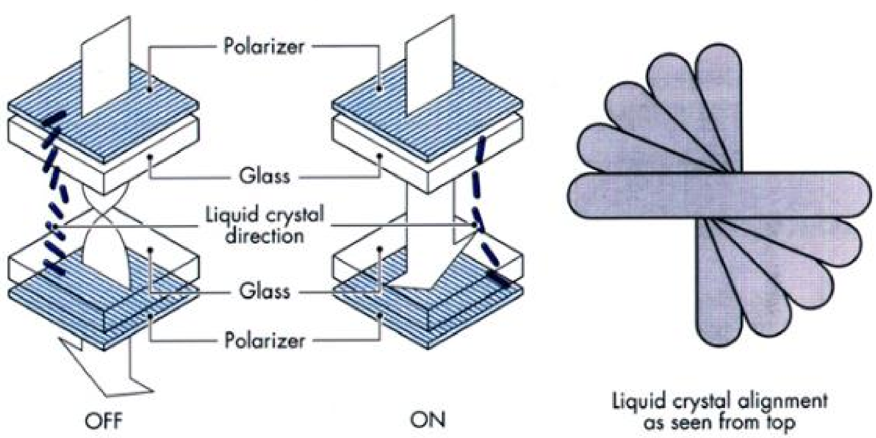
Debouncing User Switch



Liquid Crystal Display (LCD)

- The most common type of LCD allows the light to pass through when activated.
- An LCD segment is activated when a low frequency bipolar signal in the range of 30 Hz to 1KHz is applied to it.
- LCD can display characters and graphics.
- LCDs are often sold in a module with LCDs and controller unit built in.
- The Hitachi HD44780 is the most popular LCD controller being used today.

Field Effect of LCD

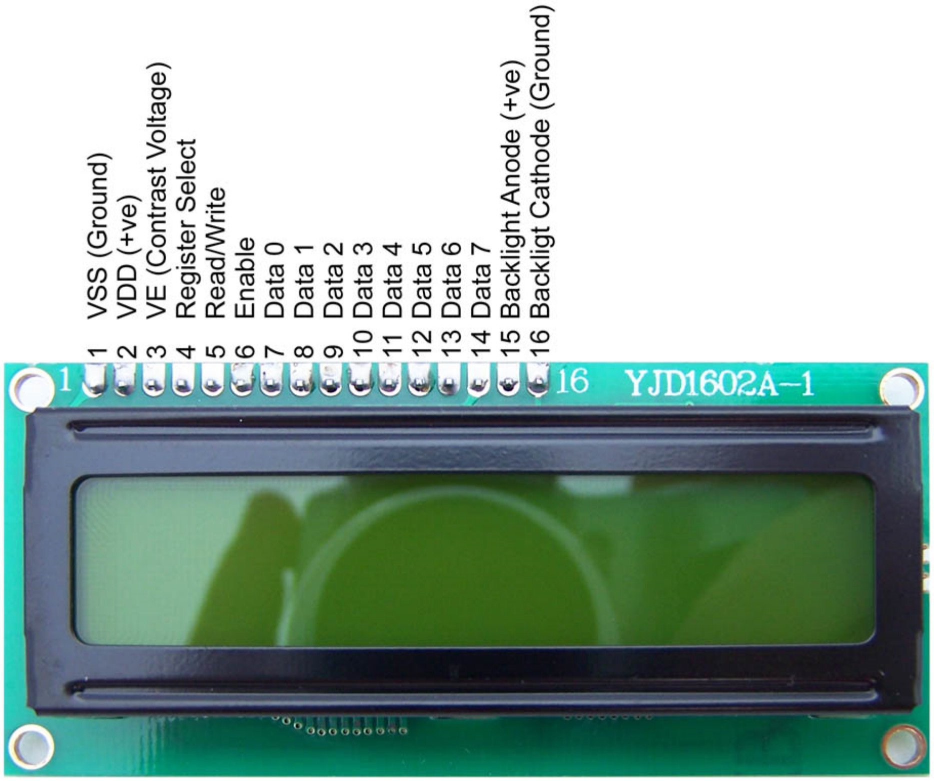


Basic construction of an LCD

https://www.youtube.com/watch?v=O3aITfU_UvE

A HD44780-Based LCD Kit

- Display capability: 4 x 20
- Uses the HD44780 as the controller as shown in the figure below.
- Pins DB7~DB0 (data port) are used to exchange data with the CPU.
- E input should be connected to one of the address decoder output or I/O pin.
- The RS signal selects instruction register (0) or data register (1).
- The VEE signal allows the user to adjust the LCD contrast.
- The HD44780 can be configured to display 1-line, 2-line, and 4-line information.



HD44780-based LCD kit

HD44780 Commands (1 of 3)

Instruction	Code										Description	Execution time
	RS	$R/\overline{W}$	B7	B6	B5	B4	B3	B2	B1	B0		
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 (address 0). Also returns display being shifted to the original position. DDRAM contents remain unchanged.	1.64 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Set 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-(S/C), shift direction (R/L). DDRAM contents remains 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 is sent and received after this setting.	40 µs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data is 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 is being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).	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

HD4478U instruction set

HD44780 Commands (2 of 3)

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 = 5x8 dots	1 = 5 x 10 dots
BF	0 = can accept instruction	1 = internal operation in progress

LCD instruction bit names

HD44780 Commands (3 of 3)

- The HD44780 has a display data RAM (DDRAM) to store data to be displayed on the LCD.

DDRAM address usage for a 2-line LCD

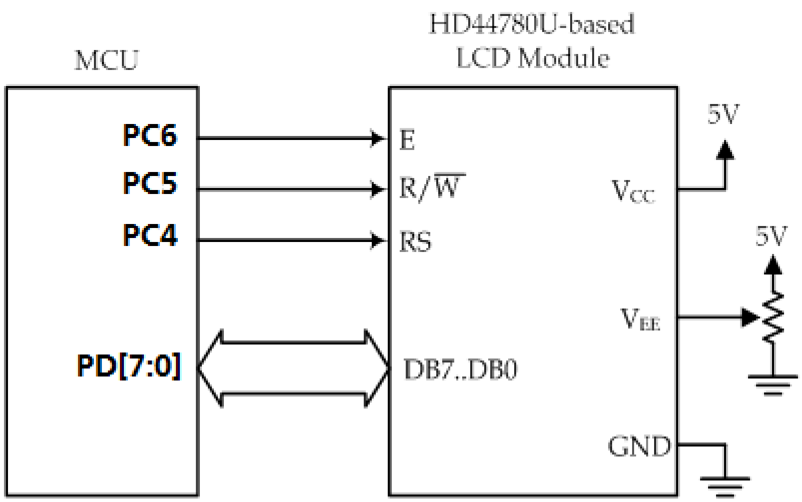
00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
40	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**4A**	**4B**	**4C**	**4D**	**4E**	**4F**

DDRAM address usage for a 1-line LCD

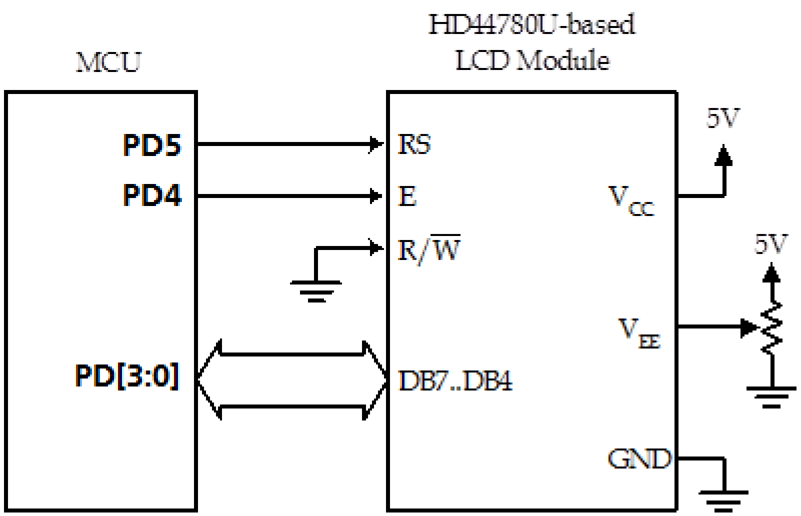
00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**

Interfacing the HD44780 with the MCU

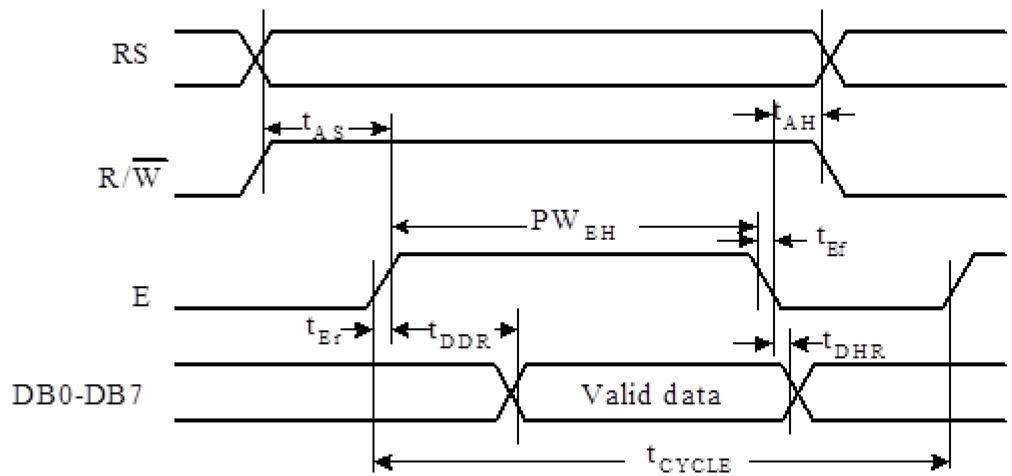
- One can treat the LCD kit as an I/O device and use an I/O port and several other I/O pins as control signals.
- The interface can be 4 bits or 8 bits.
- To read or write the LCD successfully, one must satisfy the timing requirements of the LCD. The timing diagrams for read and write are shown in the next two figures.



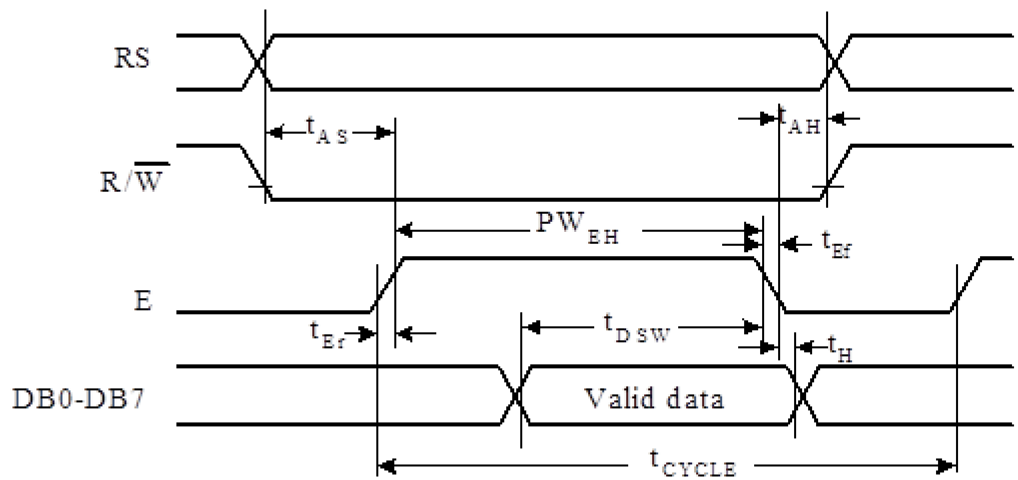
LCD interface example (8-bit bus)



LCD interface example (4-bit bus)



HD4478U LCD controller read timing diagram



HD4478U LCD controller write timing diagram

HDD4478U bus timing parameters (2 MHz operation)

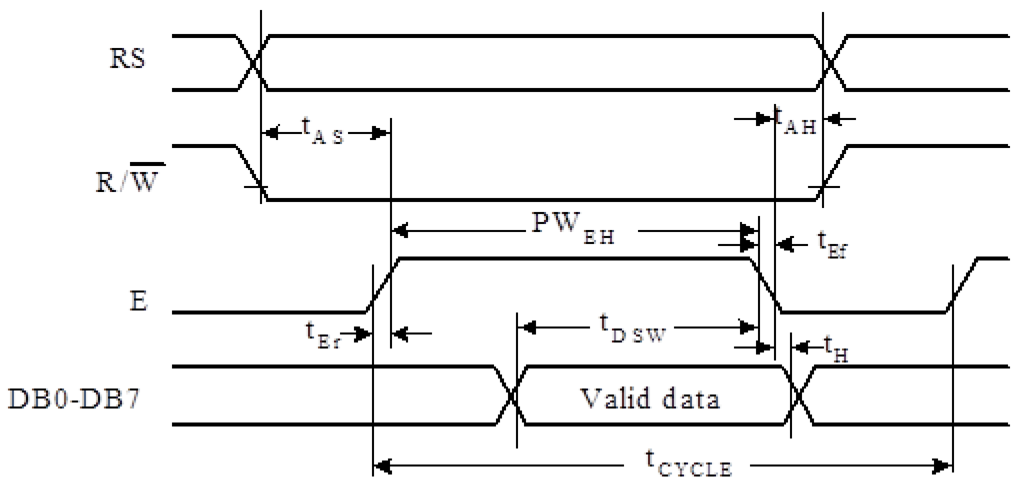
Symbol	Meaning	Min	Typ	Max.	Unit
t_{CYCLE}	Enable cycle time	500	-	-	ns
PW_{EH}	Enable pulse width (high level)	230	-	-	ns
t_{Er}, t_{Ef}	Enable rise and decay time	-	-	20	ns
t_{AS}	Address setup time, RS, $R/\overline{W}$, E	40	-	-	ns
t_{DDR}	Data delay time	-	-	160	ns
t_{DSW}	Data setup time	80	-	-	ns
t_{H}	Data hold time (write)	10	-	-	ns
t_{DHR}	Data hold time (read)	5	-	-	ns
t_{AH}	Address hold time	10	-	-	ns

- Write a function to send a command to the LCD kit
  - Most LCD commands are completed in 40 ms.
  - If the function waits for 40 ms after performing the specified operation, then most commands will be completed when the function returns.

```
#define  lcdPort   GPIOD->ODR // Port D drives LCD data pins
#define  lcdE      0x40       // E signal (PC6)
#define  lcdRW     0x20       // R/W signal (PC5) Connected to ground
#define  lcdRS     0x10       // RS signal (PC4)
#define  lcdCtl    GPIOC->ODR // LCD control port direction
```

Send Command to LCD

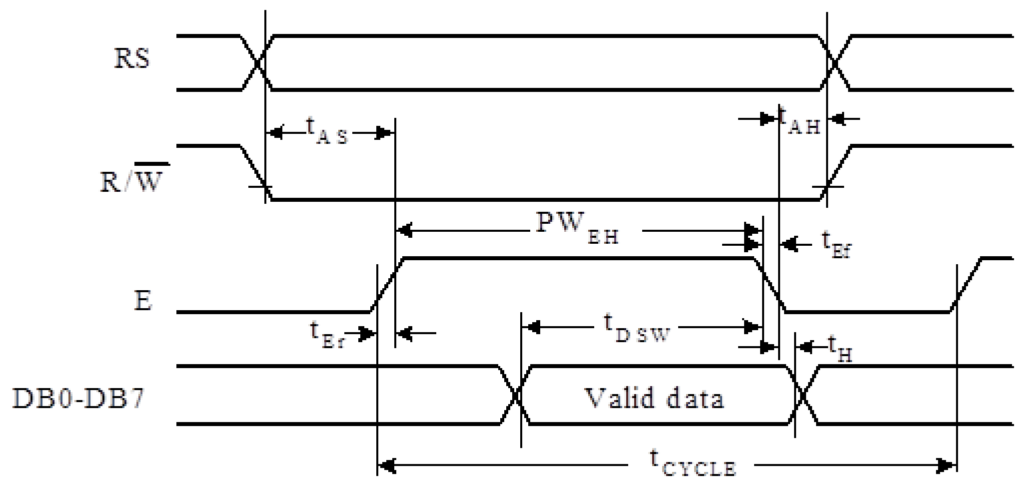
```
void cmd2lcd (char cmd) {
  char temp;
  char xa, xb;
  lcdCtl &= ~(lcdRS+lcdRW); // select instruction register & pull R/W low
  lcdCtl |= lcdE;           // pull E signal to high
  lcdPort = cmd;            // output command
  xa = 1;                   // dummy statements to lengthen E
  xb = 2;                   //       "
  lcdCtl &= ~lcdE;          // pull E signal to low
  lcdCtl |= lcdRW;          // pull R/W to high
  delayby50us(1);           // wait until the command is complete
}
```



HD4478U LCD controller write timing diagram

Send data to LCD

```
void putc2lcd(char cx) {
  char temp;
  char xa, xb;
  lcdCtl |= lcdRS;          // select LCD data register and pull R/W high
  lcdCtl &= ~lcdRW;         // pull R/W to low
  lcdCtl |= lcdE;           // pull E signal to high
  lcdPort = cx;             // output data byte
  xa = 1;                   // create enough width for E
  xb = 2;                   // create enough width for E
  lcdCtl &= ~lcdE;          // pull E to low
  lcdCtl |= lcdRW;          // pull R/W signal to high
  delayby50us(1);
}
```



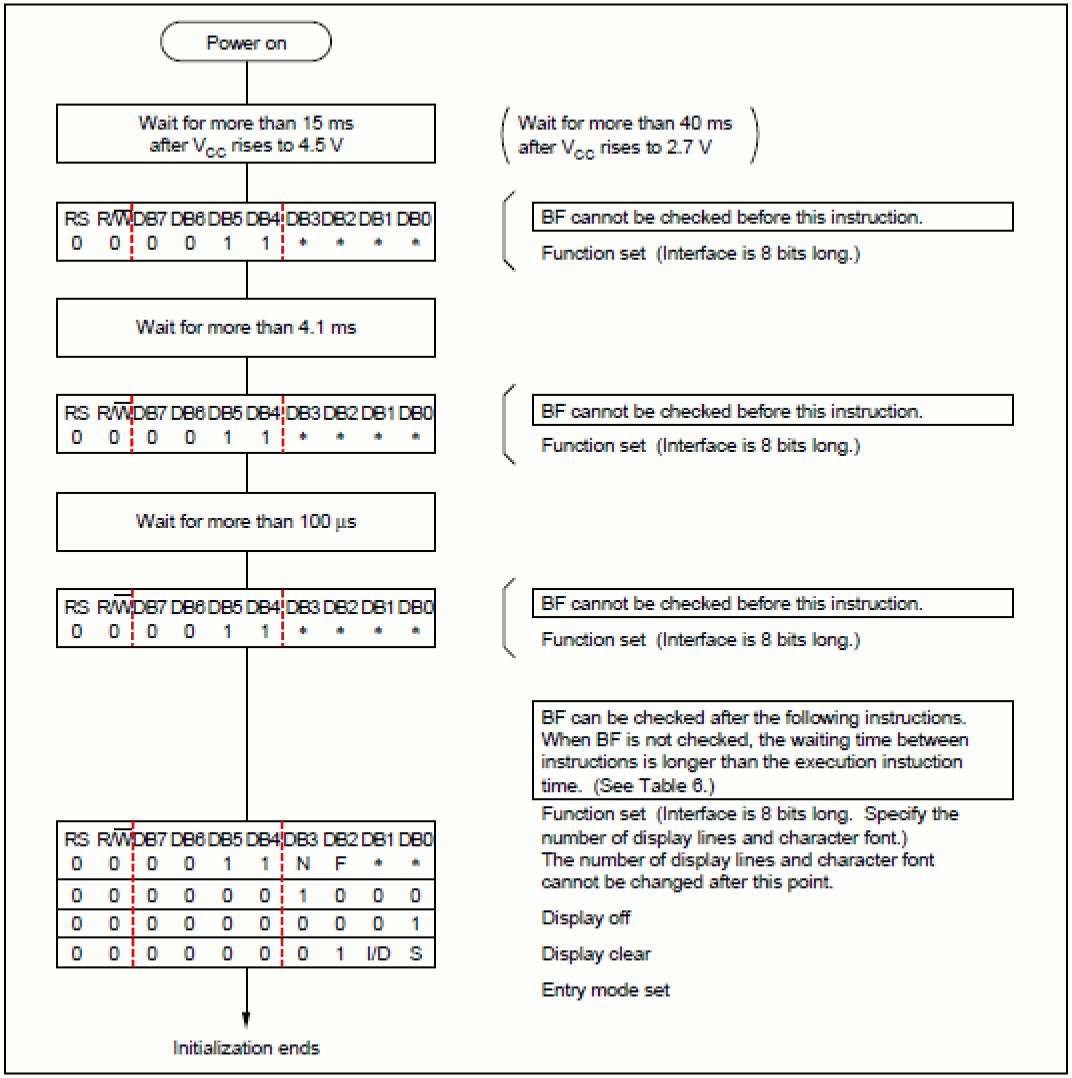
HD4478U LCD controller write timing diagram

Data String to LCD

- Function to output a string terminated by a NULL character

```
void puts2lcd (char *ptr) {
  while (*ptr) {
    putc2lcd(*ptr);
    ptr++;
  }
}
```

8-bit Initialization



LCD Initialization

- The function to configure LCD sends four commands to the LCD kit
  - Entry mode set
  - Display on/off
  - Function set
  - Clear display

```
void initlcd(void) {
  // configure lcdPort port (GPIOD) as output
  GPIOD->MODER   |= 0x00005555;
  // configure LCD control pins (PC6, PC5, & PC4) as outputs
  GPIOC->MODER   |= 0x00001500;
  delayby100ms(5); // wait for LCD to become ready
  cmd2lcd (0x38);  // set 8-bit data, 2-line display, 5x8 font
  cmd2lcd (0x0F);  // turn on display, cursor, blinking
  cmd2lcd (0x06);  // move cursor right
  cmd2lcd (0x01);  // clear screen, move cursor to home
  delayby1ms (2);  // wait until "clear display" command is complete
}
```

Example Write an C program to test the previous four subroutines by displaying the following messages on two lines:

hello world!

I am ready!

```
int main (void) {
  char *msg1 = "hello world!";
  char *msg2 = "I am ready!";
  initlcd();
  cmd2lcd(0x80); // move cursor to the 1st column of row 1
  puts2lcd(msg1);
  cmd2lcd(0xC0); // move cursor to 2nd row, 1st column
  puts2lcd(msg2);
}
```

4-bit Initialization