

Physical World / Computer Interaction

Lecture #10

Nuno Alves

Decoders, Logic Gates and LCD displays

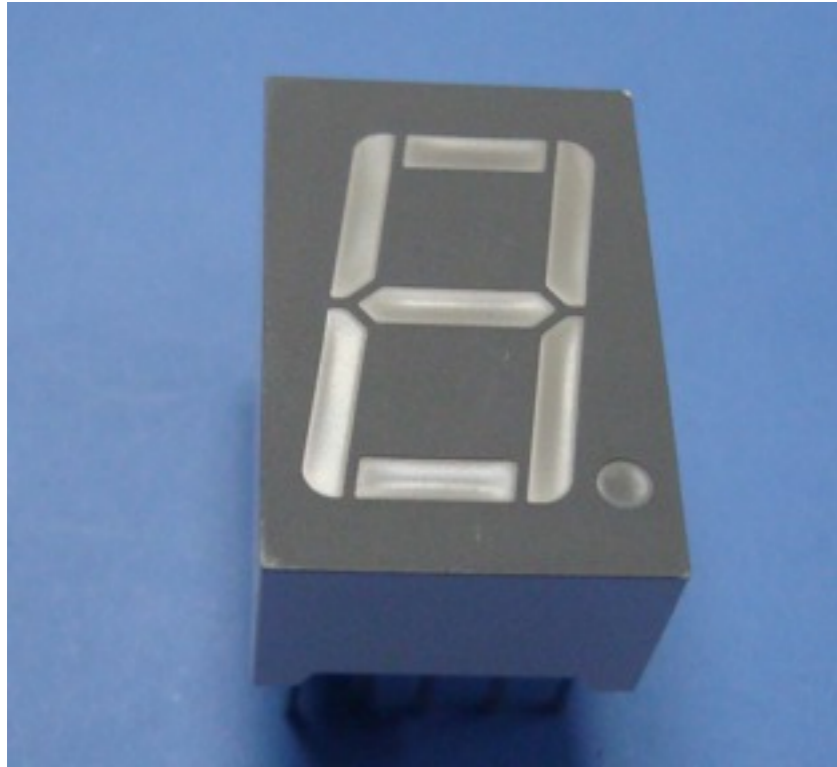


December 08, 2010

Interfacing with Seven Segment Displays



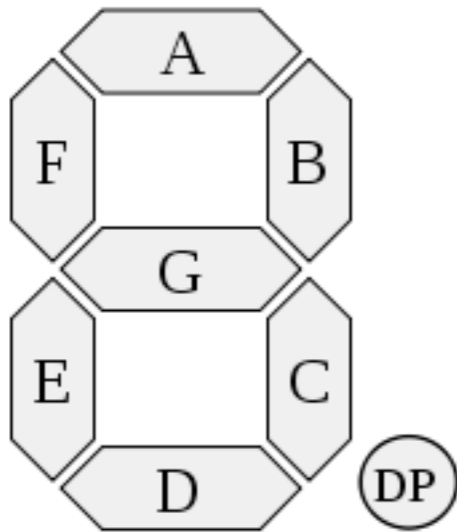
7-Segment LED Displays



- One of the old, but simpler methods of displaying numeric information is to use one or more 7-Segment numeric displays connected to your board.
- A seven segment display is composed of LED seven elements.
- Individually on or off, they can be combined to produce simplified representations of the arabic numerals.
- In most applications, the seven segments are of nearly uniform shape and size.
- The numbers are usually arranged in an slanted arrangement, which aids readability.

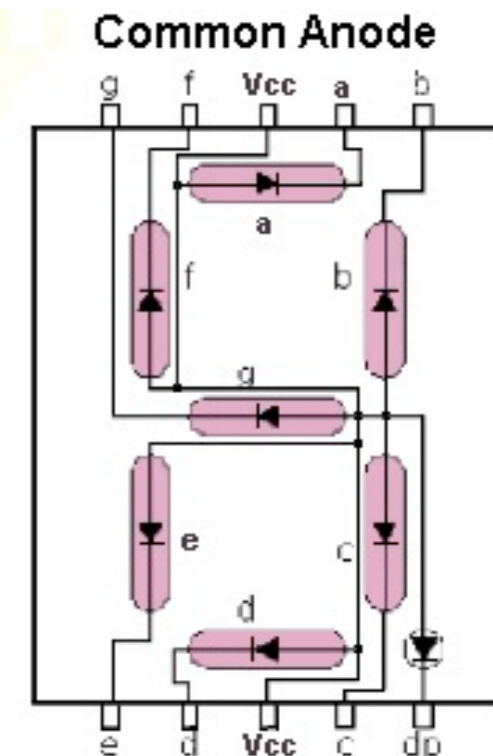
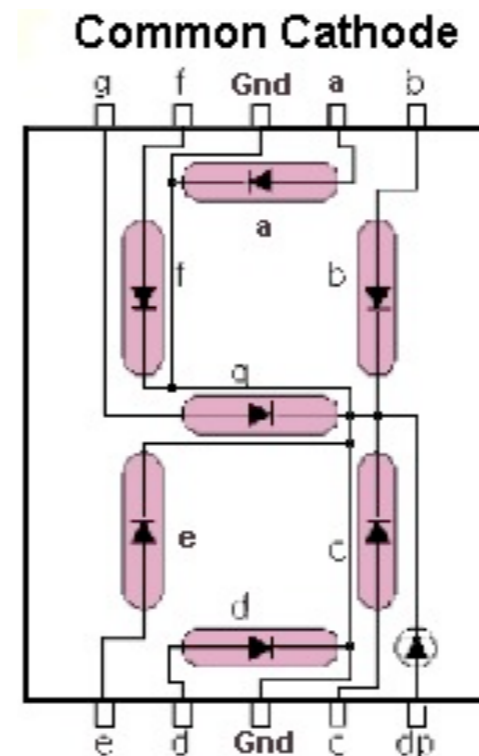


7-Segment LED Displays



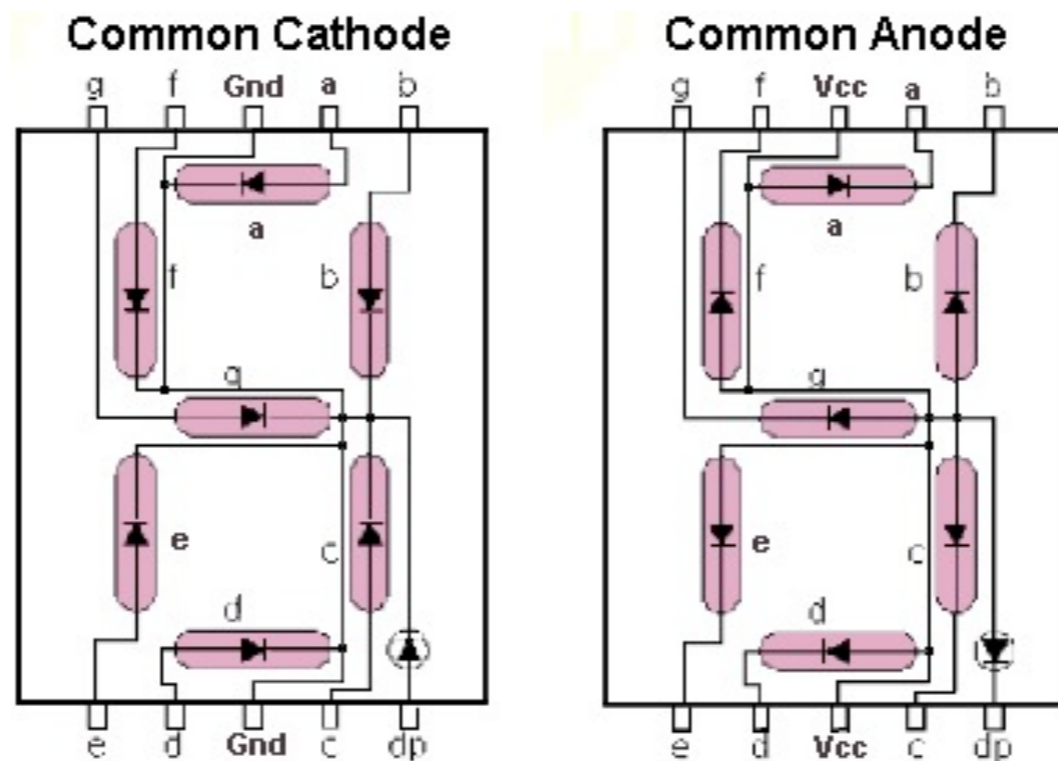
- The segments of a 7-segment display are referred to by the letters A to G.
- Some displays have the optional decimal point, used for the display of non-integer numbers.
- Circuit connections vary from different manufacturers.

Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	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



Exercise: Exploring 7-Segment LED Displays

(A) Determine the functionality of the 7-Segment display I've handed out. What are the Input/Output (IO) pins? Does it have a common anode or cathode? Which pin is it? Please add a small resistor (500-1000 Ω) to the power supply.



(B) Implement a circuit with a LED display connected to an arduino. Have the LED display the digit that was typed on the keyboard.



```
int pinA=4; int pinB=7; int pinC=9; int pinD=10;
int pinE=5; int pinF=6; int pinG=8;
int readKey=-1;
```

```
void setup()
```

```
{
  pinMode(pinA, OUTPUT);   pinMode(pinB, OUTPUT);
  pinMode(pinC, OUTPUT);   pinMode(pinD, OUTPUT);
  pinMode(pinE, OUTPUT);   pinMode(pinF, OUTPUT);
  pinMode(pinG, OUTPUT);
  Serial.begin(9600);
```

```
  digitalWrite(pinA, HIGH); digitalWrite(pinB, HIGH);
  digitalWrite(pinC, HIGH); digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH); digitalWrite(pinF, HIGH);
  digitalWrite(pinG, HIGH);
}
```

```
void loop()
```

```
{
  while (Serial.available()>0) readKey = Serial.read();
```

```
  if (readKey=='0')
```

```
  {
    digitalWrite(pinA, LOW);   digitalWrite(pinB, LOW);
    digitalWrite(pinC, LOW);   digitalWrite(pinD, LOW);
    digitalWrite(pinE, LOW);   digitalWrite(pinF, LOW);
    digitalWrite(pinG, HIGH);
  }
```

```
  if (readKey=='1')
```

```
  {
    digitalWrite(pinA, HIGH);  digitalWrite(pinB, LOW);
    digitalWrite(pinC, LOW);   digitalWrite(pinD, HIGH);
    digitalWrite(pinE, HIGH);  digitalWrite(pinF, HIGH);
    digitalWrite(pinG, HIGH);
  }
```

```
}
```

One solution...

Improve the
readability of this code
with using arrays



Arduino Variables Review

I. Types of variables:

1. boolean : 1 bit (true or false)
2. char : 8 bit (-128 to 127)
3. byte or unsigned char: 8 bit (0 to 255)
4. int : 16 bit (-32,768 to 32,767)
5. word or unsigned int : 16 bit (0 to 65535)
6. long: 32 bit (-2,147,483,648 to 2,147,483,648)
7. unsigned long : 32 bit (0 to $2^{32} - 1$)
8. float : 32 bits (3.4028235E+38 to -3.4028235E+38)
9. double : in arduino, double and floats have the same precision



```

int pinId[7]={4,7,9,10,5,6,8};
int readKey=-1;

void setup()
{
  for (int i=0; i<7; i++)
  { pinMode(pinId[i],OUTPUT); }
  Serial.begin(9600);

  for (int i=0; i<7; i++)
  digitalWrite(pinId[i],HIGH);
}

void loop()
{
  while (Serial.available()>0)
  readKey = Serial.read();
  if (readKey=='0')
  {
    char data={B11000000};
    for (int i=0; i<7; i++)
    digitalWrite(pinId[i],bitRead(data,i));
  }

  if (readKey=='1')
  {
    char data={B11111001};
    for (int i=0; i<7; i++)
    digitalWrite(pinId[i],bitRead(data,i));
  }
  readKey=-1;
}

```

A cleaner solution

bitRead() - Reads a bit of a number

Syntax - `bitRead(x, n)`

Parameters:

x: the number from which to read

n: which bit to read, starting at 0 for the rightmost bit

bitWrite() - Writes a bit of a number

Syntax - `bitWrite(x, n,b)`

Parameters:

x: the number from which to read

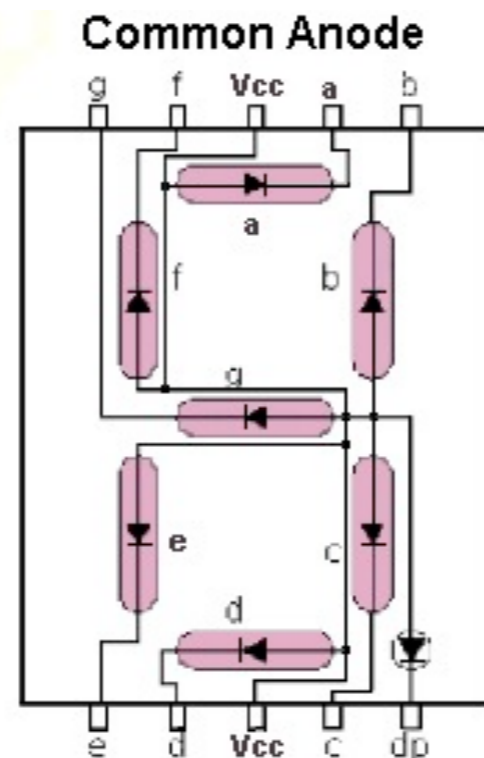
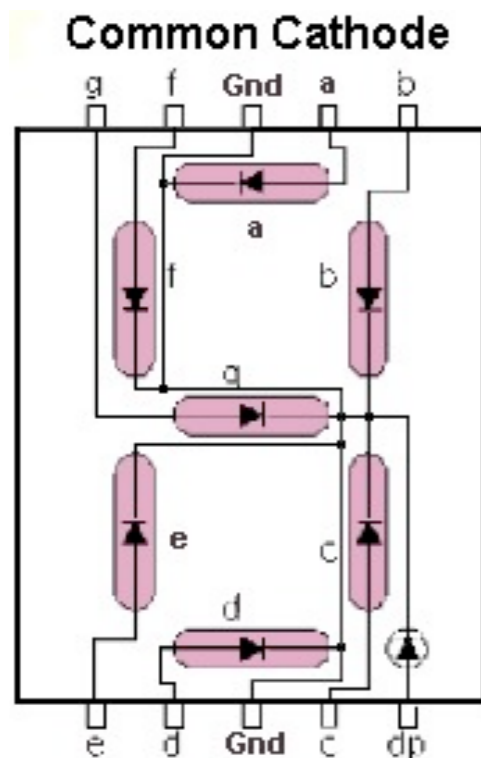
n: which bit to read, starting at 0 for the rightmost bit

b: boolean value



Decoder

- What is I want the arduino to cycle through all possible number combinations?
- There are 8 control pins, so I could connect one control pin to one of the arduino digital input/output pins.
- For example, if we want to display the number 1, then turn ON segment B and C, and turn OFF all the other segments.
- This is BAD. We are wasting a lot of arduino IO pins.

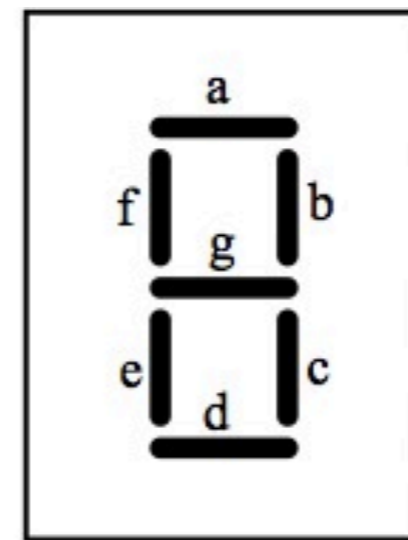
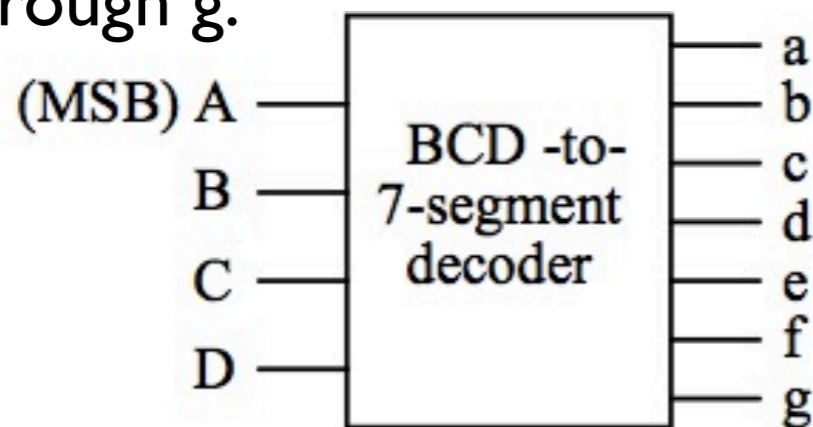


- Using boolean/binary logic, with 4 arduino pins we can control at most 16 segments ($2^4 = 16$).
- With 3 arduino pins we can control at most 8 (2^3) segments.



BCD-to-7-segment decoder

- Its purpose is to decode BCD inputs (the binary codes corresponding to the decimal values 0 - 9) in order to light the appropriate segments on a 7-segment display.
- The decoder needs 7 outputs in order to control the 7 segments, which are labeled a through g.



7-segment display

The decoder should function as follows:

if $ABCD = 0000$, the display should light the digit 0 (segments a, b, c, d, e, f)

if $ABCD = 0001$, the display should light the digit 1 (segments b, c)

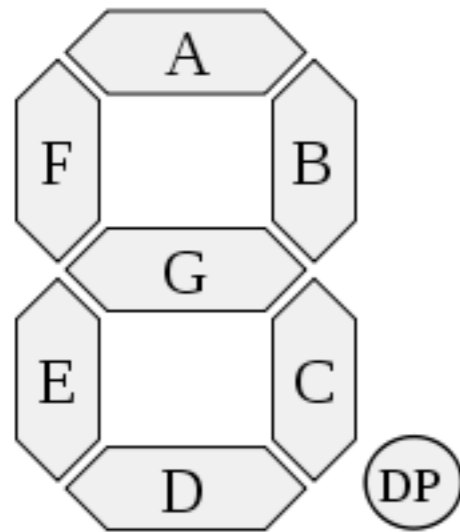
if $ABCD = 0010$, the display should light the digit 2 (segments a, b, d, e, g)

.

if $ABCD = 1001$, the display should light the digit 9 (segments a, b, c, f, g)



BCD-to-7-segment decoder



$$a = A + C + B \cdot D + \bar{B} \cdot \bar{D}$$

$$b = \bar{B} + C \cdot D + \bar{C} \cdot \bar{D}$$

We need to get a boolean logic expression for all outputs... above are a and b, found using the method of Karnaugh Maps. Search the web for some tutorials if you are interested.

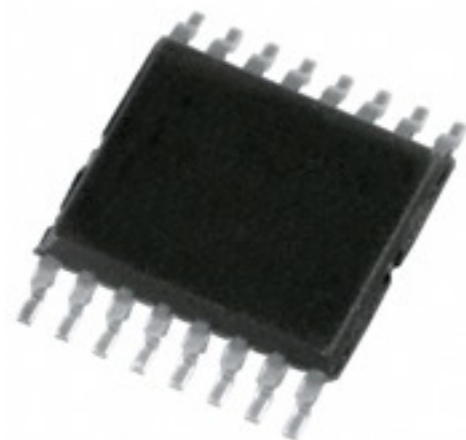
	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
	1	0	1	0	x	x	x	x	x	x	x
	1	0	1	1	x	x	x	x	x	x	x
	1	1	0	0	x	x	x	x	x	x	x
	1	1	0	1	x	x	x	x	x	x	x
	1	1	1	0	x	x	x	x	x	x	x
	1	1	1	1	x	x	x	x	x	x	x

↑
binary representation

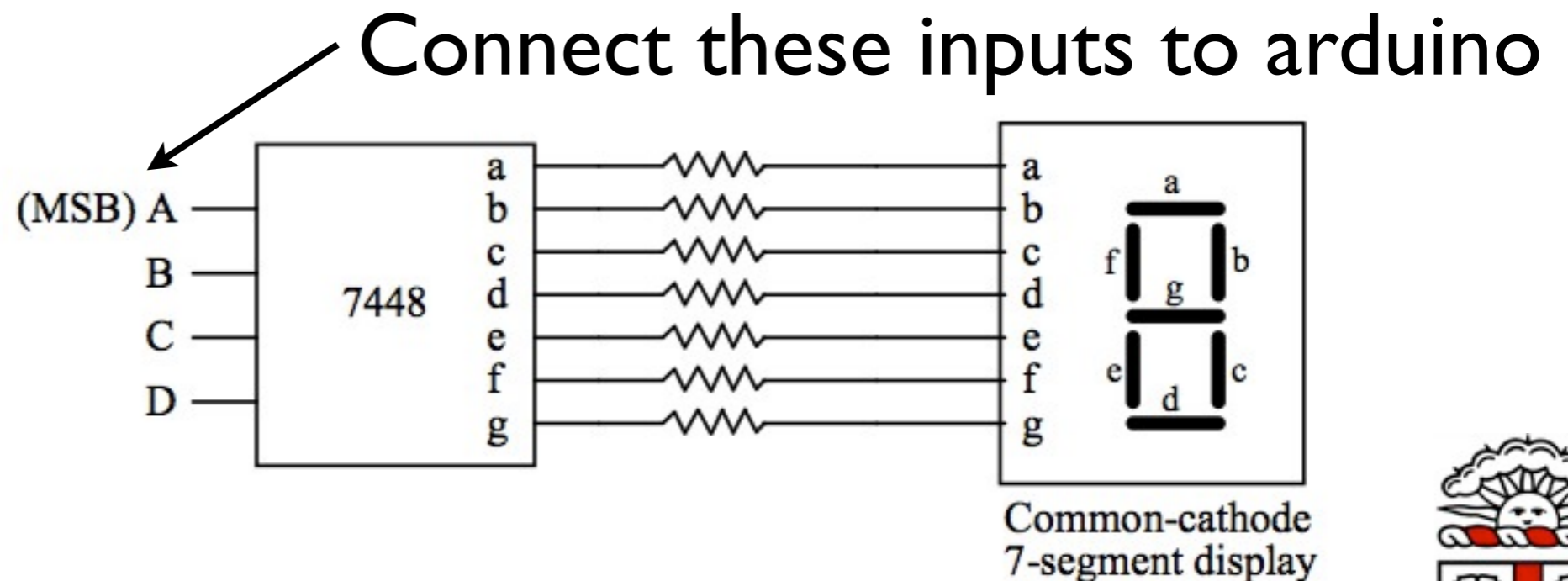


What is the point of decoders?

- If you want to reduce the number of pins needed to control something, you need to use decoders.
- You can make one from scratch. It is not too hard, but its time consuming.
- If you need a standard decoder, for example to control the 7 segment display, you can buy one.



IC Decoder
#7448
(\$0.15)

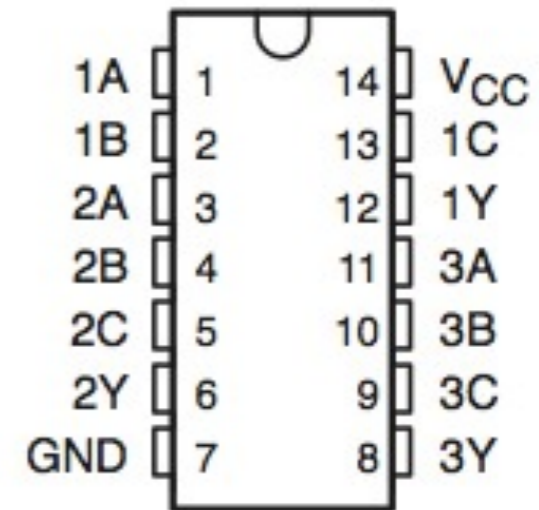
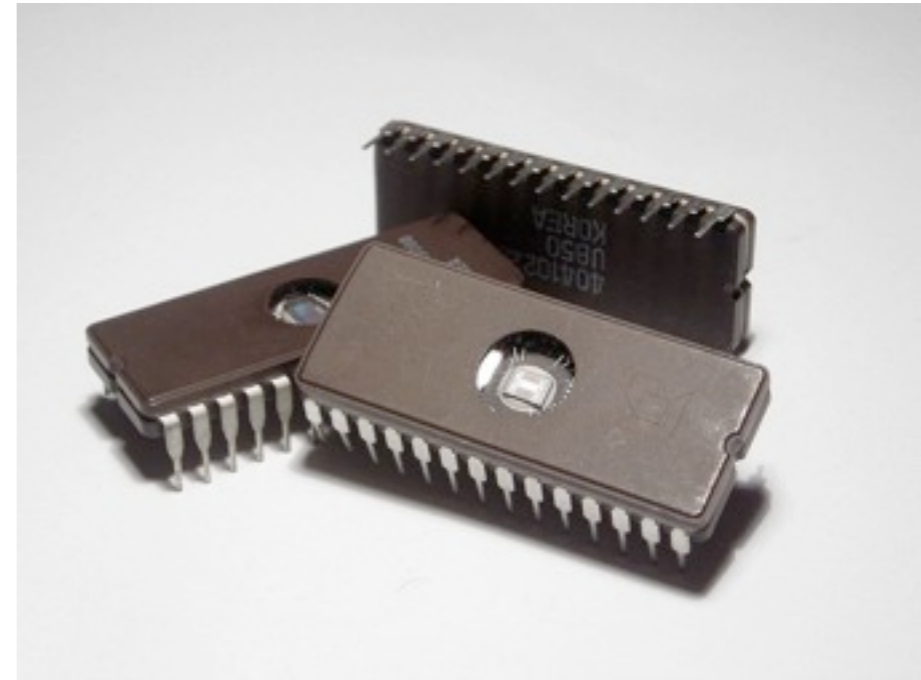


Logic Gates



Logic Gates

- Digital logic : ON / OFF of voltage signals.
- Logic gates are devices that have inputs which draw virtually no current, and outputs that act as voltage sources.
- The voltage output depends on the input voltage.
- The input/output relationship characterizes the gate.
- Logic voltages are either close to 5V or close to 0V.
- A High voltage is referred to as a logic 1, where a low voltage is a logic 0.
- The inputs and outputs to logic gates could be 1's or 0's.

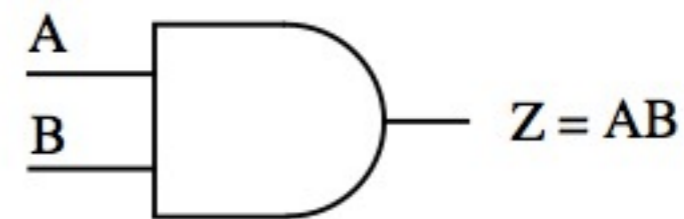


AND Gate

- Two basic kinds of gates are the AND gate and the OR gate.
- A simple AND gate has two inputs, A and B, and one output, Z.
- All inputs to an AND gate must be 1 for the output to be 1.
- If both A and B are 1 then the output is 1. Otherwise, the output is 0.

Input: A	Input: B	Output: AB
0	0	0
0	1	0
1	0	0
1	1	1

Truth table



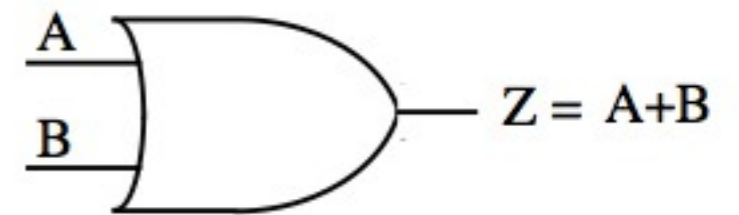
Circuit Schematic



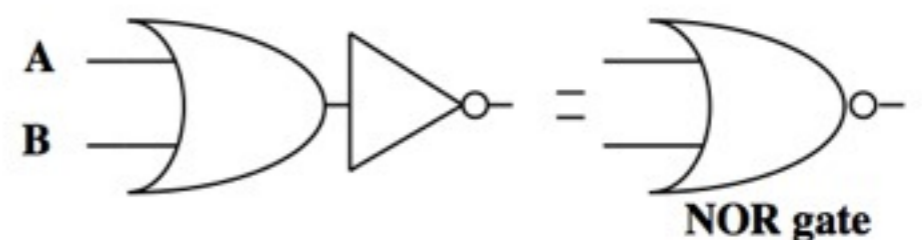
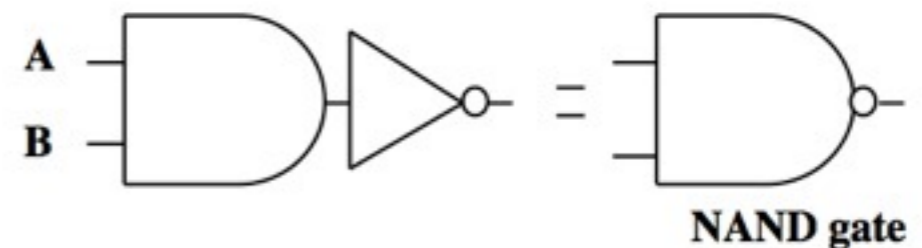
OR, NOR and NAND Gates

- A basic OR gate has two inputs, A and B, and an output.
- The output is 1 if either A or B, or both, are 1, and 0 only when both A and B are 0.

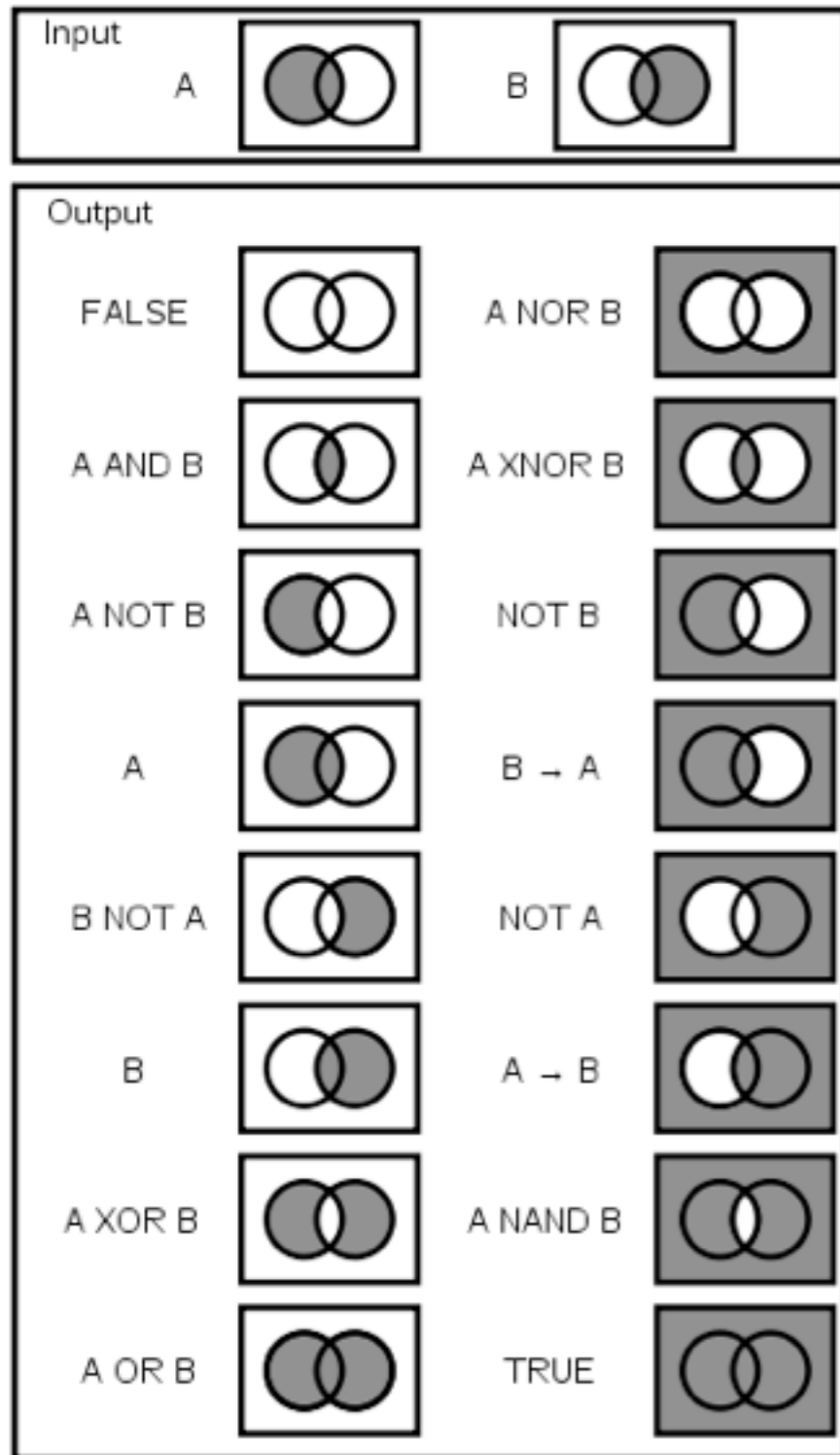
Input: A	Input: B	Output: A+B
0	0	0
0	1	1
1	0	1
1	1	1



- NAND and NOR gates are constructed by adding an inverter after AND and OR operators



NAND Gate



NAND gates are perhaps the most widely used logic gates.

Why?

1) Its a universal gate. You can create any other logic function using a NAND gate. Can you come up with a way to re-create the logic functionality of a NOT gate with just NANDs? What about XORs and ORs?

2) Its implementation contains very few transistors. Your flash drivers are made out of NAND gates.



When to use logic gates?

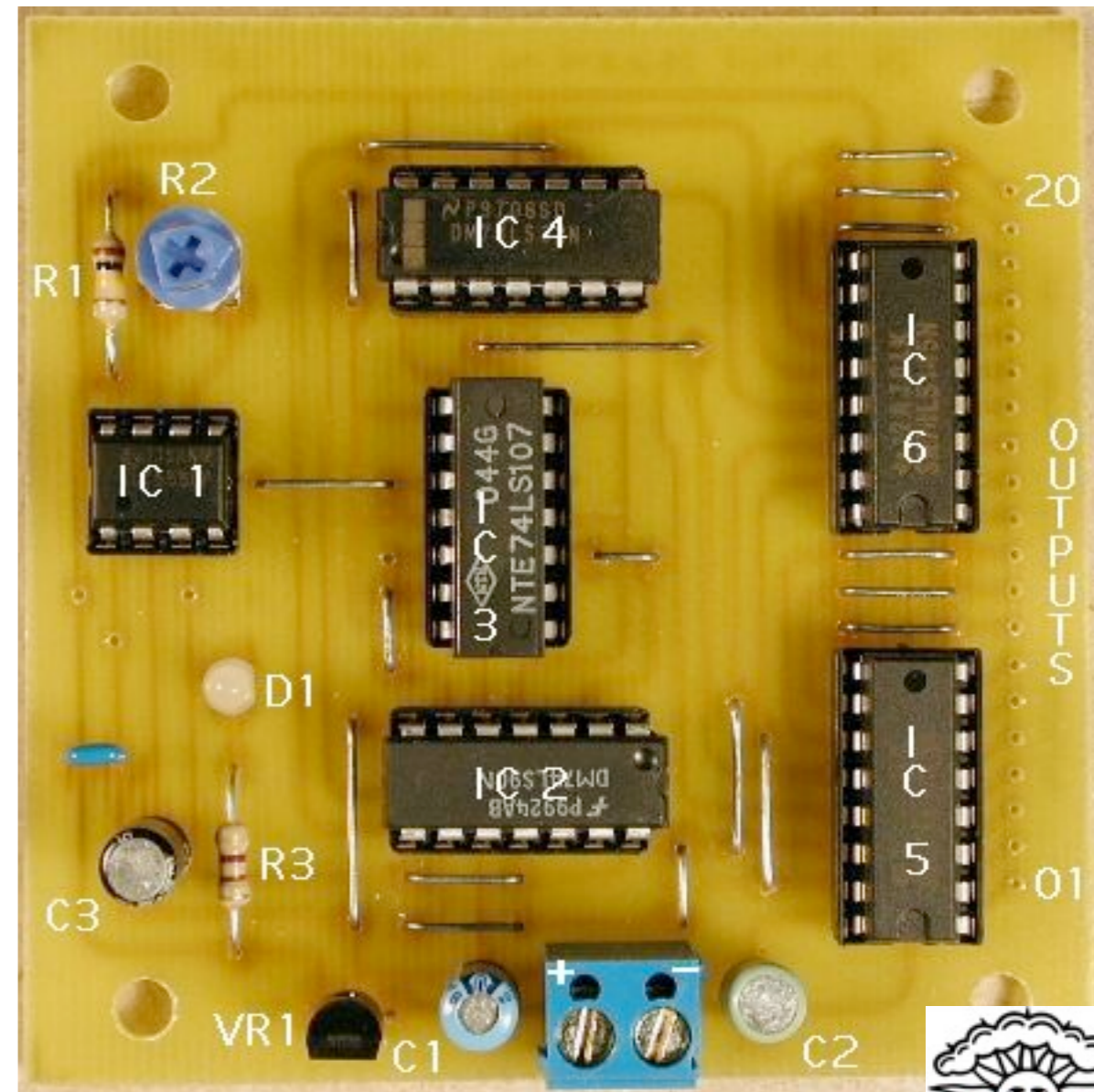
The only purpose of logic gates is to generate logic outcomes based on logic inputs.

Two examples:

- Control traffic lights at a busy intersection
- Implement interactive art installations : when a visitor touches a piece and while another visitor walks behind him, turn off lights.

You could use your arduino to perform these logic operations, but for certain projects you may run out of available pins.

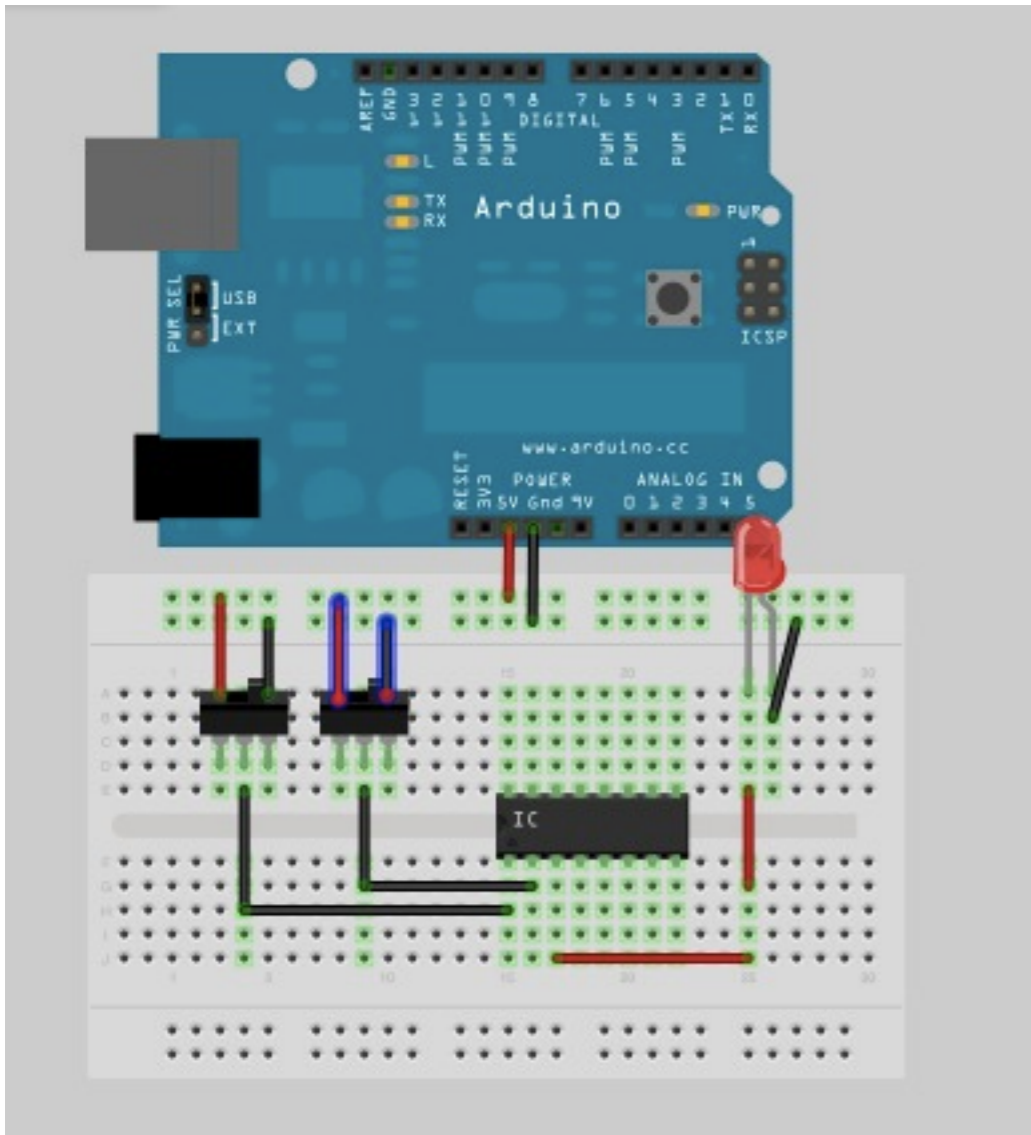
For very simple physical world / computer interactions, implementing your design in logic gates may be much cheaper and more reliable than using a micro-controller.



Playing with logic gates

This is going to be our experimental setting:

- The two switches are connected to VDD and GND.
- The middle pin is connected to an IC.
- The exact location of the IC pins are determined by looking at the specifications sheet.
- Connect the output of the logic gate to a LED.
- Remember... you could replace the switches with sensors!!!

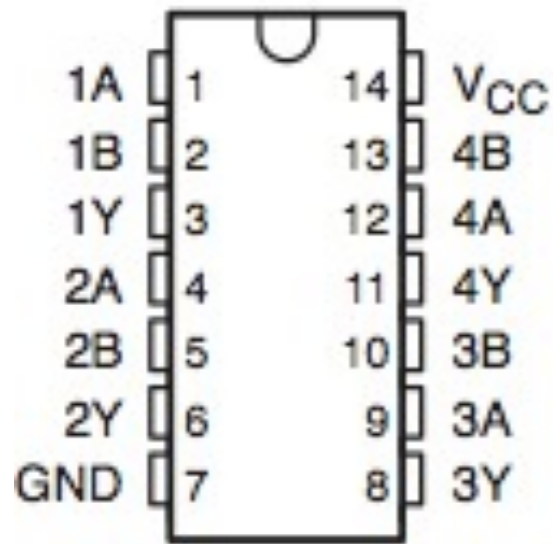


IC stands for *Integrated Circuit*.



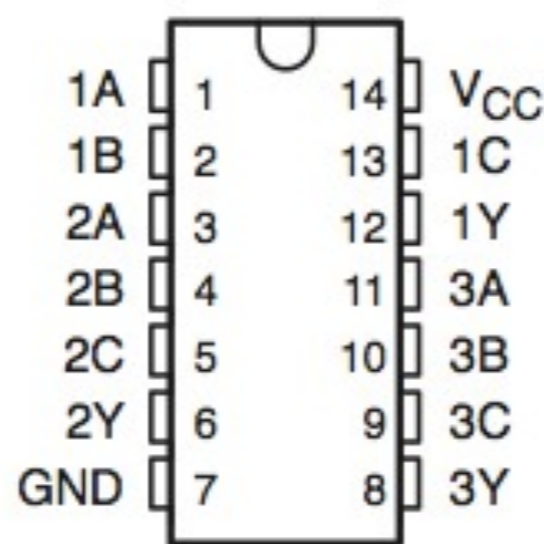
Exercise

SN74S00N
2-Input NAND

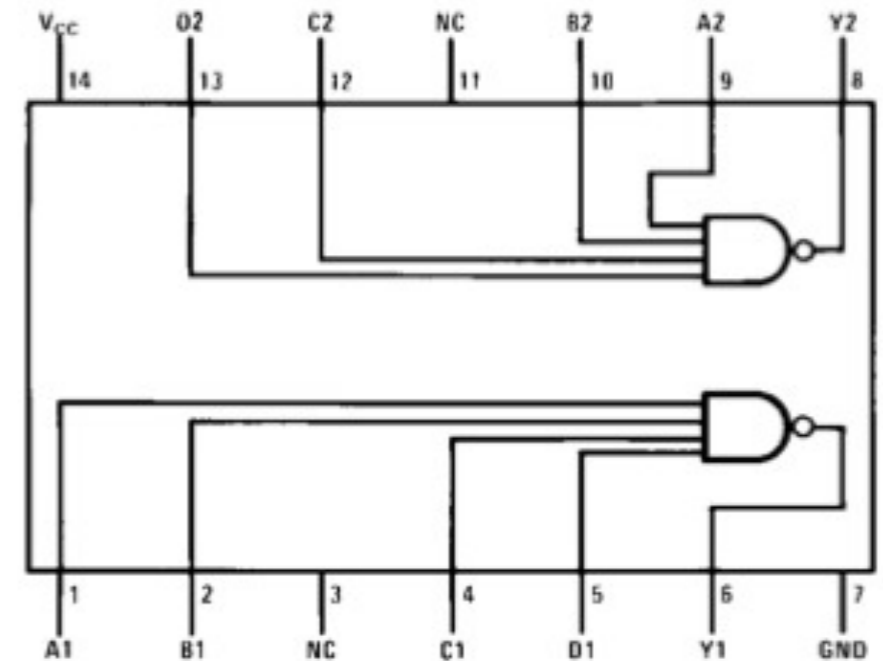


$$Y = \text{NAND}(A, B)$$

SN74ACT10N
3-Input NAND



DM74S20
4-Input NAND



Using one of these ICs, implement the following circuit:

- Leave LED ON... unless
- Switch A is ON and switch B is OFF

With a 2-Input NAND:
 $Y = \text{NAND}(A, \text{NAND}(A, B))$

With a 3-Input NAND:
 $Y = \text{NAND}(A, \text{NAND}(A, B, I), I)$



2 Input NAND Logic Table

A	B	NAND(A,B)	NAND(NAND(A,B),B)
0	0	1	1
0	1	1	1
1	0	1	0
1	1	0	1



LCD Displays



LCD Displays

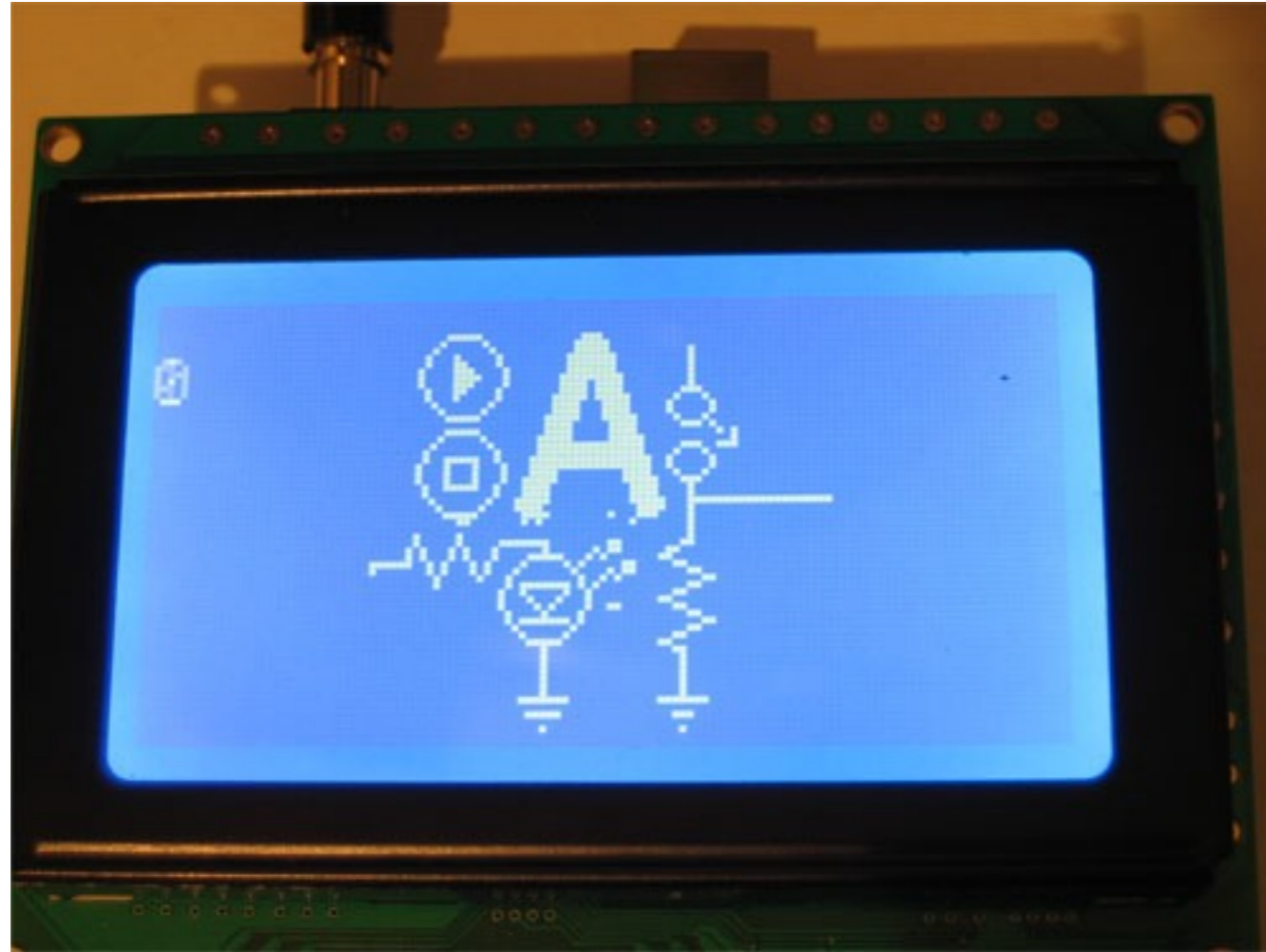
- There are hundreds of different kinds of LCDs, the ones we'll be covering are character LCDs.
- Character LCDs are ideal for displaying text. They can also be configured to display small icons but the icons must be only 5x7 pixels or so (very small!)
- Here is an example of a character LCD, 16 characters by 2 lines



If you look closely you can see the little rectangles where the characters are displayed. Each rectangle is a grid of pixels.



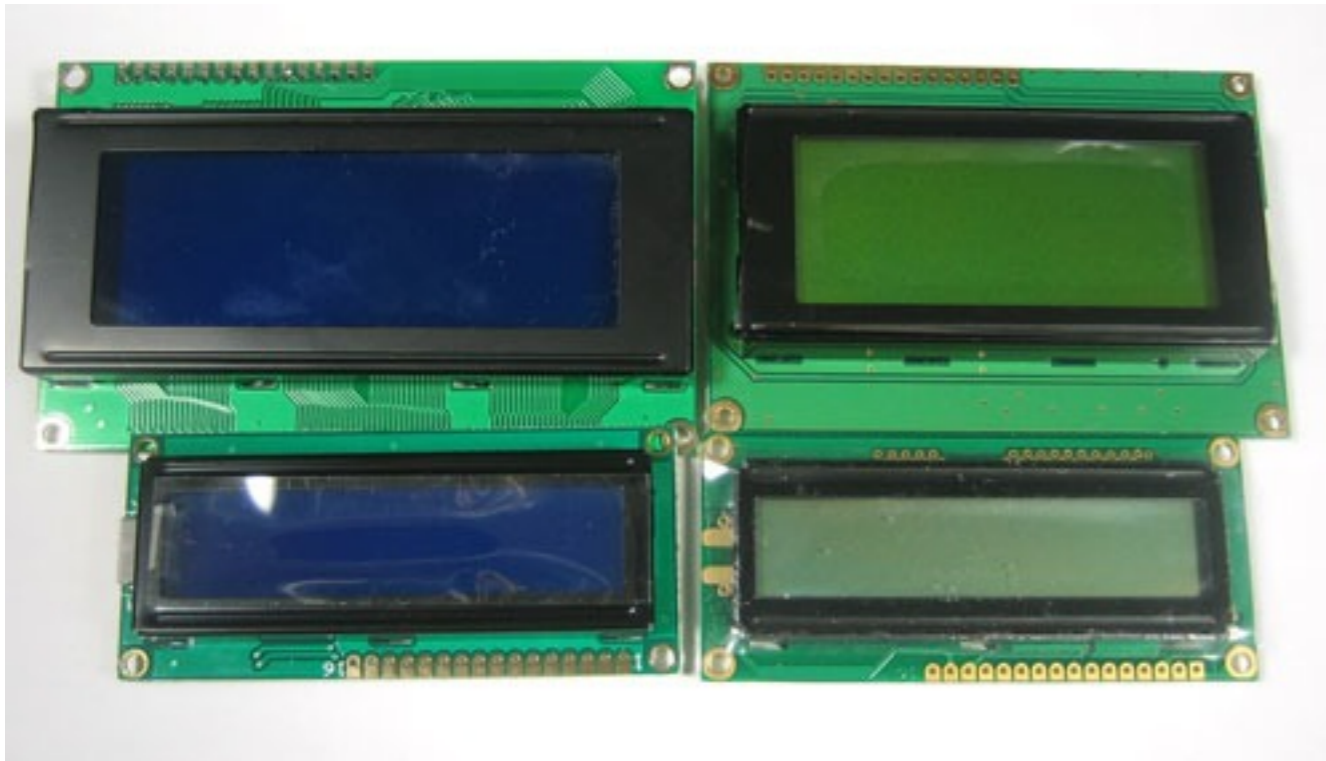
Graphical LCD Displays



- The graphical LCD has one big grid of pixels (in this case 128x64 of them).
- It can display text but its best at displaying images.
- Graphical LCDs tend to be larger, more expensive, difficult to use and need many more pins because of the complexity added.



LCD Displays



- LCD displays come in many sizes, shapes and colors.
- There is an LCD standard (HD44780) and also an arduino library that supports that standard (Liquid Crystal Library).
- Easy to spot due to their 16-pin interface.
- This means these displays are 'swappable' - if you build your project with one you can unplug it and use another size.

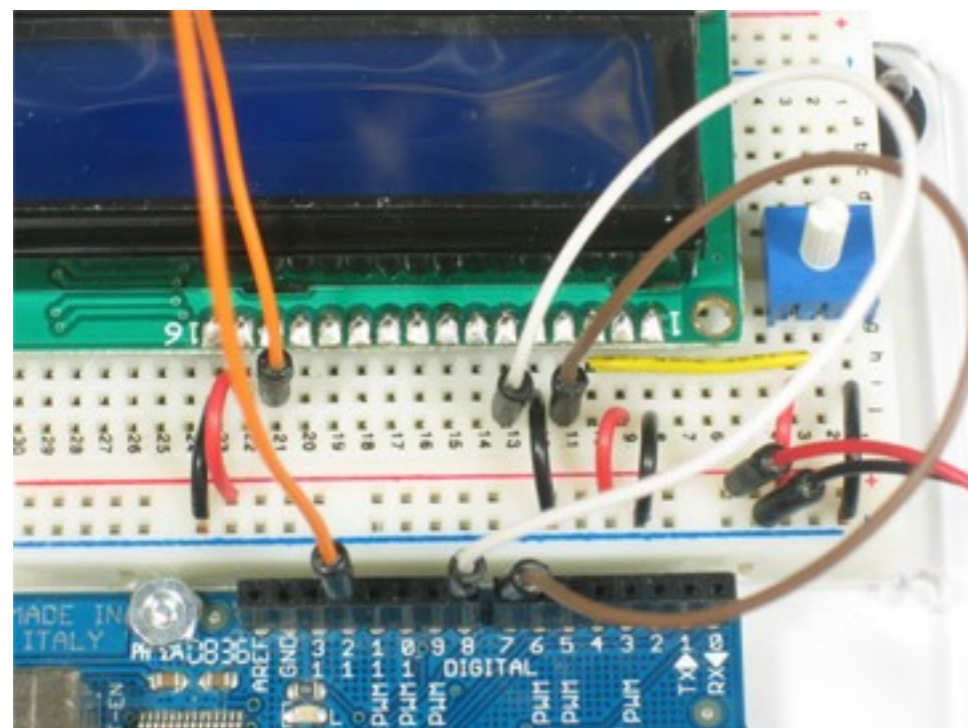
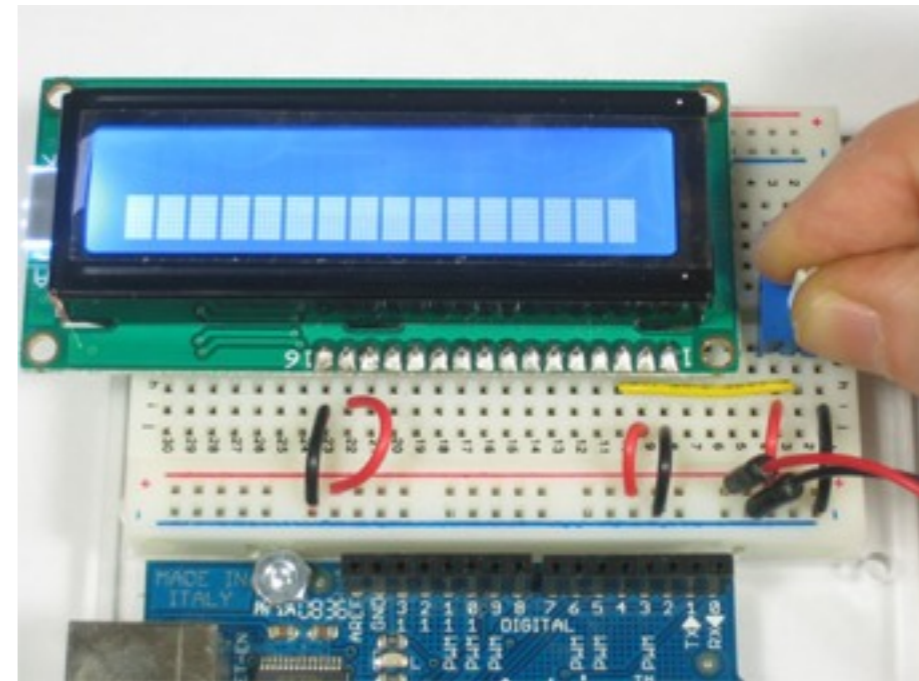
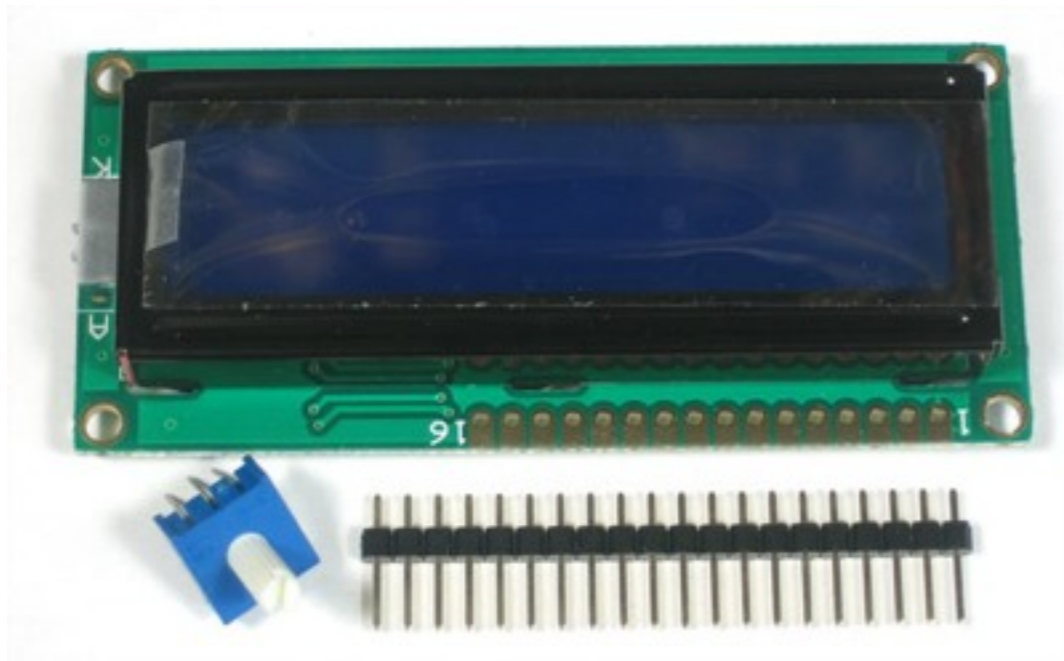


- Your code may have to adjust to the larger size but at least the wiring is the same!



Preparing LCD Displays

Check this excellent site for step by step instructions:
<http://www.ladyada.net/learn/lcd/charlcd.html>



Liquid Crystal Library

- Makes it easy to program messages on LCD displays
- Check it out at <http://arduino.cc/en/Reference/LiquidCrystal>



```
#include <LiquidCrystal.h>

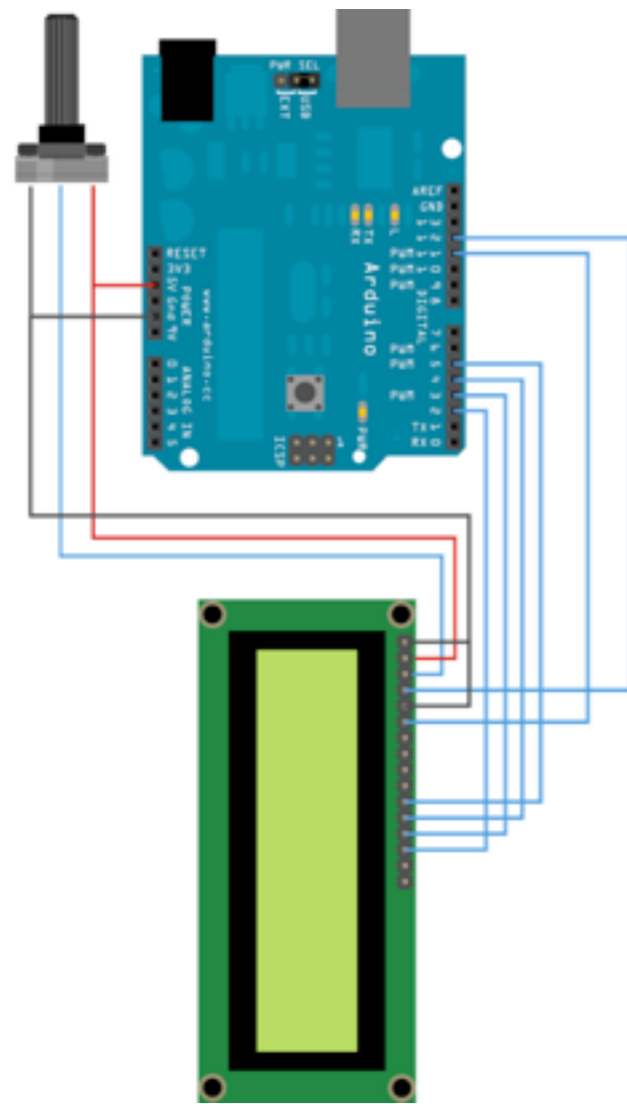
//interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {
  // set up the LCD's number of columns/rows
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since
  counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis()/1000);
}
```



Blinking Cursor



```
// include the library code:
#include <LiquidCrystal.h>

//interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {
  // set up the LCD's number of columns/rows
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // Turn off the blinking cursor:
  lcd.noBlink();
  delay(3000);
  // Turn on the blinking cursor:
  lcd.blink();
  delay(3000);
}
```



Liquid Crystal Functions

- `clear()` : Clears entire LCD screen
- `home()` : Positions the cursor in the upper-left of the LCD.
- `setCursor()` : Set the location at which subsequent text will be displayed.
- `write()` : Write a single character to the LCD.
- `print()` : Prints a string.
- `cursor()` : Display an “_” at the position where the next character will be written.
- `noCursor()` : Hides the LCD cursor.
- `blink()` : Blinks the cursor.
- `noBlink()` : Turns off the blinking LCD cursor.
- `display()` : Turns ON the LCD display, after it's been turned off with `noDisplay()`
- `noDisplay()` : Turns OFF the LCD display
- `scrollDisplayLeft()` : Scrolls the contents of the display one space to the left.
- `scrollDisplayRight()` : Scrolls the contents of the display one space to the right.
- `autoscroll()` : Turns on automatic scrolling of the LCD.
- `noAutoscroll()` : Turns OFF automatic scrolling of the LCD.
- `leftToRight()` : Set the direction for text written to the LCD to left-to-right
- `rightToLeft()` : Set the direction for text written to the LCD to right-to-left



Create Custom Characters

- With function `createChar()`, we can create a custom character (glyph) for use on the LCD.
- Up to eight characters of 5x8 pixels are supported (numbered 0 to 7).
- The appearance of each custom character is specified by an array of eight bytes, one for each row.
- The five least significant bits of each byte determine the pixels in that row. To display a custom character on the screen, write() its number.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

byte smiley[8] = {
    B00000,
    B10001,
    B00000,
    B00000,
    B10001,
    B01110,
    B00000,
};

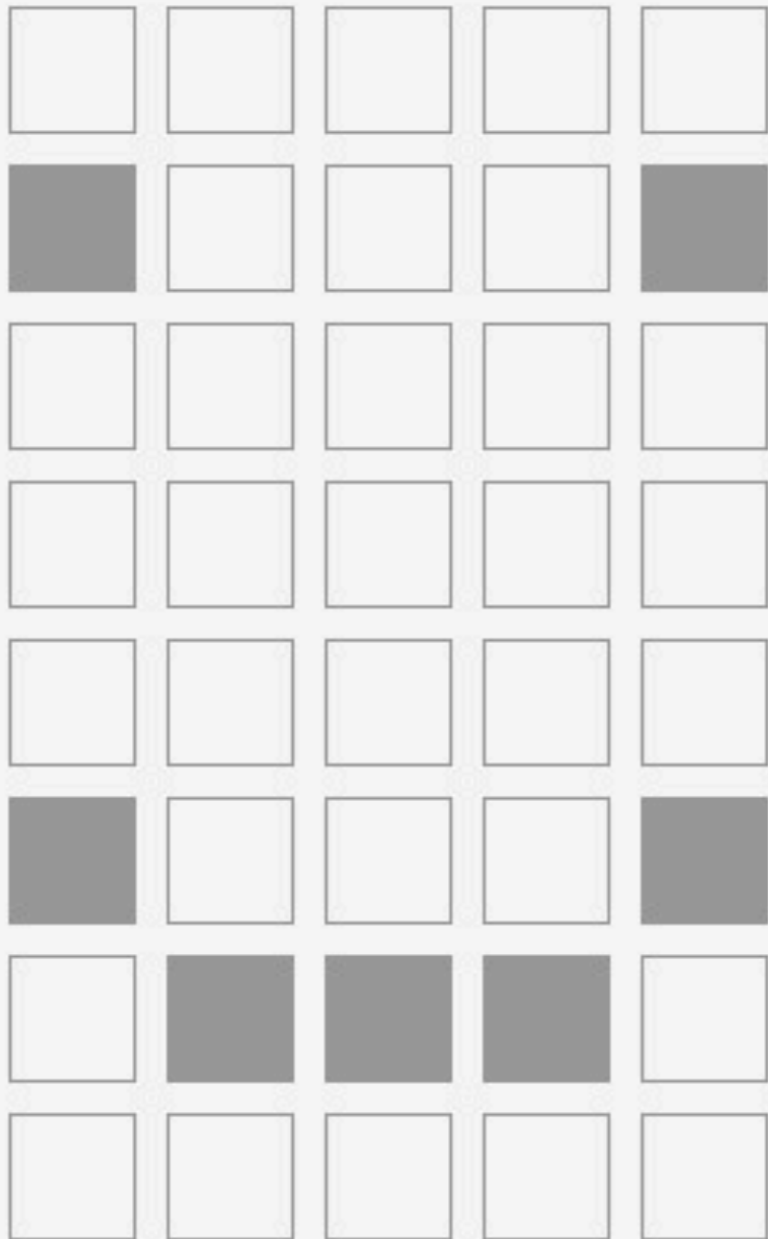
void setup() {
    lcd.createChar(0, smiley);
    lcd.begin(16, 2);
    lcd.write(0);
}

void loop() {}
```



Arduino LiquidCrystal

Character Creator



Sketch Code:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte newChar[8] = {
    B00000,
    B10001,
    B00000,
    B00000,
    B00000,
    B10001,
    B01110,
    B00000
};

void setup() {
    lcd.createChar(0, newChar);
    lcd.begin(16, 2);
    lcd.write(0);
}

void loop() {}
```

clear/paint all

sample 1

sample 2

<http://icontexto.com/charactercreator/>



Exercise: Display text on LCD displays

(A) Using the serial interface (see example below), write a HAPPY smiley face to the LCD screen when a 0 is pressed and a sad smiley face when the character I is pressed.

```
int readKey = 0;
void setup()
{
  //initiate serial communication
  Serial.begin(9600);
}

void loop()
{
  while (Serial.available()>0)
  {
    readKey = Serial.read();
  }
  //print contents of readVar
  Serial.println(readKey);
  //wait half a second
  delay(500);
}
```



Exercise Code

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

byte smiley[8] = { B00000,B10001,B00000, B00000,B10001,B01110,B00000};
byte sad[8] = {B00000,B10001,B00000,B00000,B00000,B01110,B10001};
int readKey=-1;

void setup()
{
  Serial.begin(9600);
  lcd.begin(16, 2); lcd.createChar(0, smiley);lcd.createChar(1, sad);
}
void loop()
{
  while (Serial.available()>0) {readKey = Serial.read();}
  if (readKey=='1')
  {
    Serial.println("1 was pressed\n");
    lcd.setCursor(0, 0); lcd.write(0);
  }
  if (readKey=='0')
  {
    Serial.println("0 was pressed\n");
    lcd.setCursor(0, 0); lcd.write(1);
  }
  readKey=-1;
}
```



Giving Credit

- Some of these slides were “borrowed” from the AS220 Arduino class slides.

