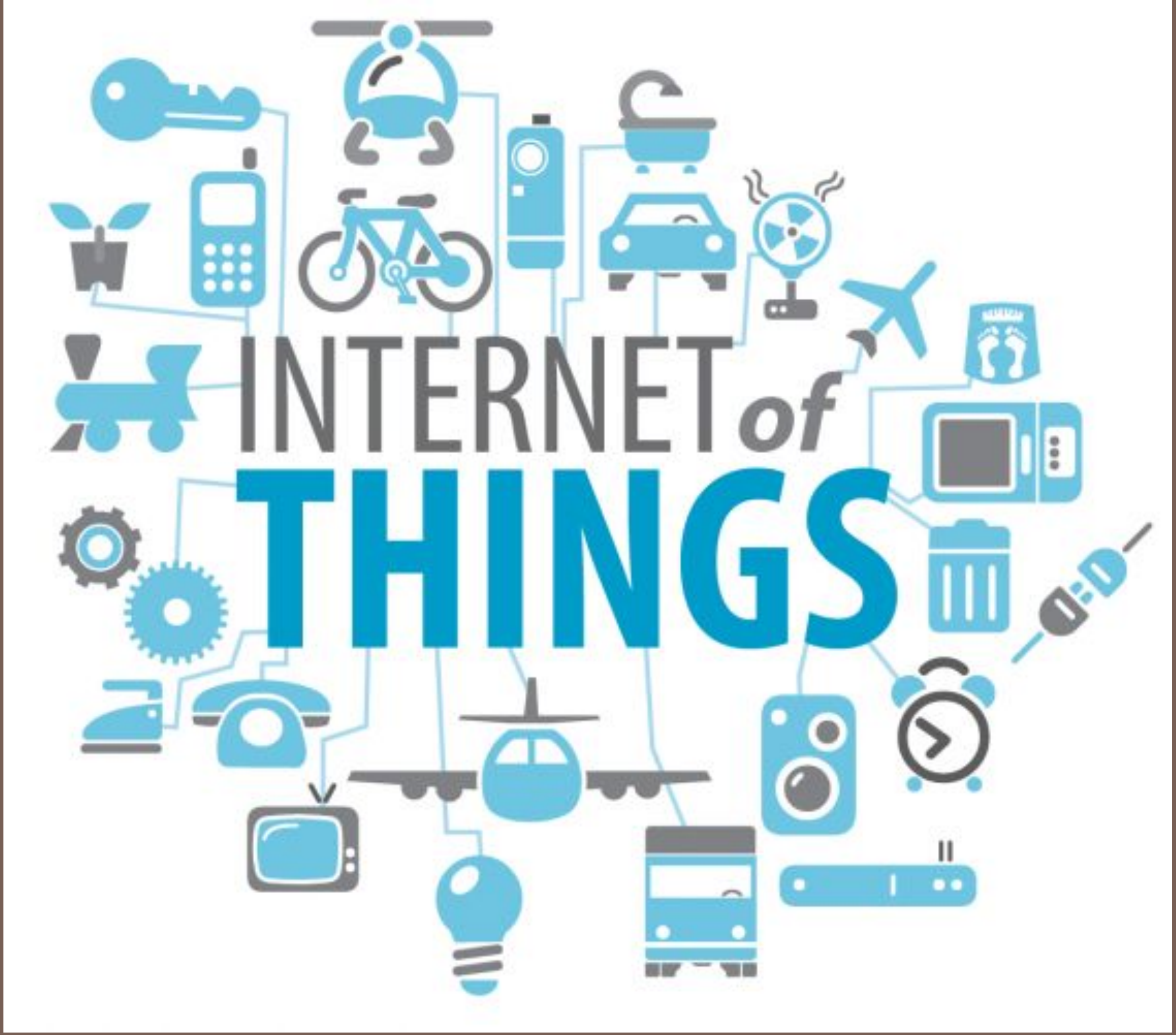


IMPLEMENTING IOE

Week 9

Assist. Prof. Rassim Suliyev - SDU 2017

Remotely Controlling Devices

- interact with almost any device that uses some form of remote control
  - TVs, audio equipment, cameras, garage doors, appliances, and toys
- remote controls work by sending digital data from a transmitter to a receiver
  - infrared light (IR) or wireless radio technology
- different protocols (signal patterns) used
  - translate key presses into a digital signal

IR remote

- works by turning an LED on and off in patterns
  - produce unique codes
  - codes are typically 12 to 32 bits
- Each key on the remote is associated with a specific code that is transmitted when the key is pressed
- If the key is held down, the remote usually sends the same code repeatedly
  - some remotes (e.g.,NEC) send a special repeat code when a key is held down

IR remote

- low-cost IR receiver module
  - detect the signal and provide a digital output that the Arduino can read
  - digital output is then decoded by a library called IRremote
  - same library is used when Arduino sends commands to act like a remote control
- to install the library, place it in the folder named libraries in your Arduino sketch folder

Wireless radio technology

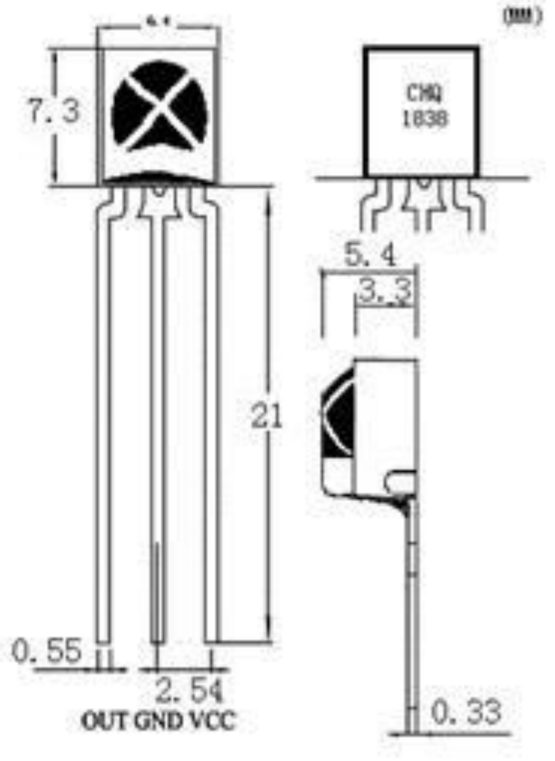
- more difficult to emulate than IR controls
- button contacts on these controls can be activated by Arduino
  - simulate button presses by closing the button contacts circuit inside the remote control
- need to take apart the remote control and connect wires from the contacts to Arduino
- isolation prevents voltages from Arduino from harming the remote control, and vice versa

Optocouplers

- Components called optocouplers are used to provide electrical separation between Arduino and the remote control
- enable you to safely control another circuit
- provide a way to keep things electrically separated
- contain an LED, which can be controlled by an Arduino digital pin
- light from the LED in the optocoupler shines onto a light-sensitive transistor
- turning on the LED causes the transistor to conduct

Responding to an IR Control

- Arduino responds to IR remote signals using a device called an IR receiver module
- The IR receiver converts the IR signal to digital pulses
- IR remote library decodes these pulses and provides a numeric value for each key



Responding to an IR Control

```
#include <IRremote.h> //adds the library code to the sketch
const int irReceiverPin = 2; //pin the receiver is connected to
const int ledPin = 13;
IRrecv irrecv(irReceiverPin); //create an IRrecv object
decode_results decodedSignal; //stores results from IR detector
boolean lightState = false; //keep track of whether the LED is on
unsigned long last = millis(); //remember when we last received an IR
message
void setup(){
  pinMode(ledPin, OUTPUT);
  irrecv.enableIRIn(); // Start the receiver object
}
void loop(){
  if (irrecv.decode(&decodedSignal) == true){ //true if message received
    if (millis() - last > 250) { //has it been 1/4 sec since last message?
      lightState = !lightState; //Yes: toggle the LED
      digitalWrite(ledPin, lightState);
    }
    last = millis();
    irrecv.resume(); // watch out for another message
  }
}
```

IRremote library

- #include <IRremote.h> - makes the library code available to your sketch
- IRrecv irrecv(irReceiverPin); - creates an IRrecv object to receive signals from an IR receiver module connected to irReceiverPin
- use the irrecv object to access the signal from the IR receiver
- decoded responses provided by the library are stored in a variable named decode_results

IRremote library

- irrecv.enableIRIn(); - start listening the IR receiver
- irrecv.decode(&decodedSignal); - check the results of receiver
  - returns true if there is data, which will be placed in the decodedSignal
- irrecv.resume(); - library needs to be told to continue monitoring for signals
- code toggles the LED if more than ¼ s since the last received message (otherwise, the LED will toggle quickly by remotes that send codes more than once)

Decoding IR Control Signals

```
#include <IRremote.h>
const int RECV_PIN = 2;
const int NUM_OF_BUTS = 21;
long BUTTON_CODES[NUM_OF_BUTS] = {
    0xE0E08877, //button 0
    0xE0E020DF, //button 1
    0xE0E0A05F, //button 2
    0xE0E0609F, //button 3
    0xE0E010EF, //button 4
    0xE0E0906F, //button 5
    0xE0E050AF, //button 6
    0xE0E030CF, //button 7
    0xE0E0B04F, //button 8
    0xE0E0708F, //button 9
    0xE0E08679, //button 100+/DOWN
    0xE0E006F9, //button 200+/UP
    0xE0E0D02F, //button VOL-
    0xE0E0E01F, //button VOL+
    0xE0E016E9, //button EQ/MENU
    0xE0E0A659, //button PREV/LEFT
    0xE0E046B9, //button NEXT/RIGHT
    0xE0E0F00F, //button PAUSE/MUTE
    0xE0E008F7, //button CH-
    0xE0E048B7, //button CH+
    0xE0E040BF  //button CH/POWER
  };
String BUTTON_NAMES[NUM_OF_BUTS] = {
    "0","1","2","3","4","5","6","7","8","9",
    "100+/DOWN","200+/UP","VOL-","VOL+",
    "EQ/MENU","PREV/LEFT","NEXT/RIGTH",
    "PAUSE/MUTE","CH-","CH+","CH/POWER"
  };
```

```
IRrecv irrecv(RECV_PIN);
decode_results results;
void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}
void loop() {
  if (irrecv.decode(&results)) {
    //Serial.println(results.value, HEX);
    int button_id = findButton(results.value);
    if(button_id <0){
      Serial.println("Unknown button is pressed!");
    } else{
      Serial.print("Button ");
      Serial.print(BUTTON_NAMES[button_id]);
      Serial.println(" is pressed!");
    }
    irrecv.resume(); // Receive the next value
  }
}
int findButton(long val){
  for(int i = 0; i < NUM_OF_BUTS; i++){
    if(val == BUTTON_CODES[i]) return i;
  }
  return -1;
}
```

Imitating Remote Control Signals

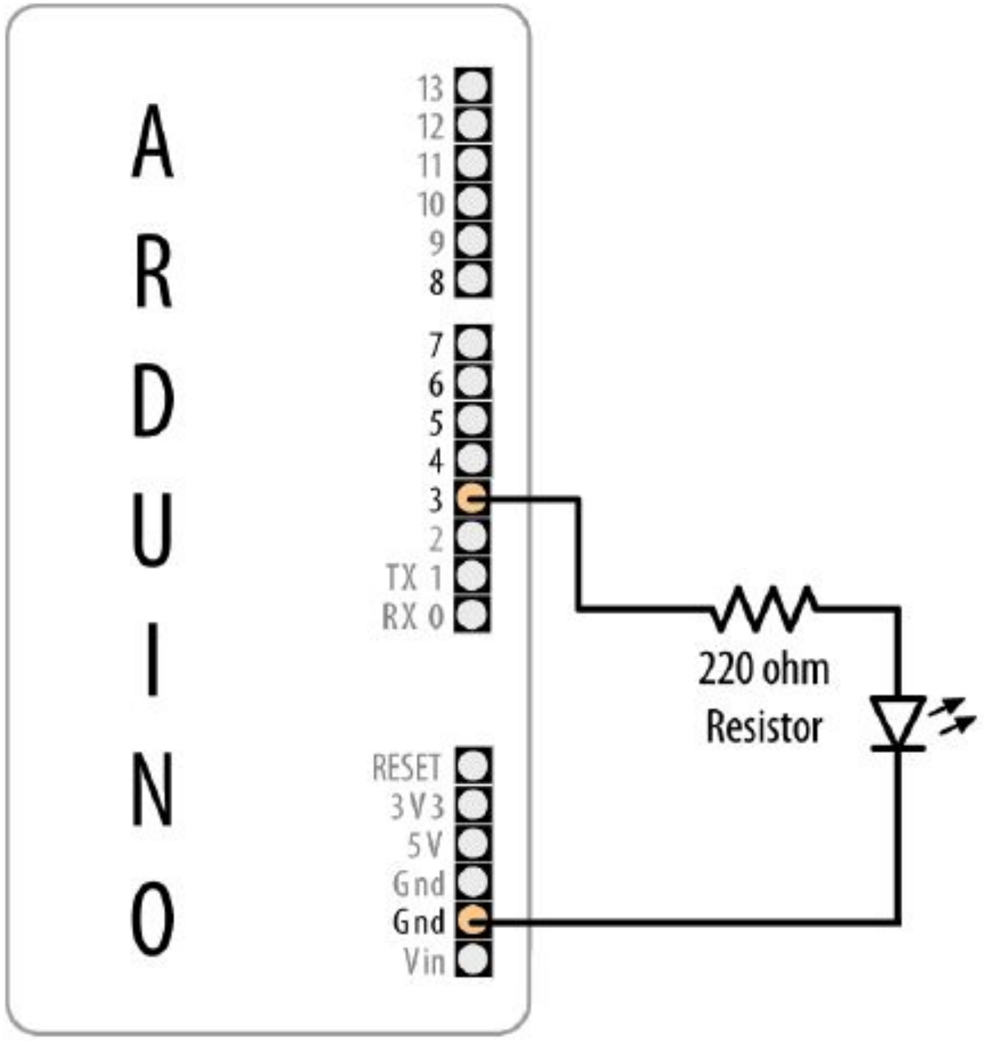
- want to use Arduino to control a TV or etc
- Arduino controls the device by flashing an IR LED
  - duplicate the signal that would be sent from your remote control
- IR library handles the translation from numeric code to IR LED flashes
- need to create an object for sending IR messages
- IRsend object - control the IR LED on pin 3
  - hardcoded within the library, can not change it

Imitating Remote Control Signals

- irsend.sendSony(code, nob); - send code in sony format which has nob bits
- irSend object has different functions for various popular infrared code formats
  - check the library documentation for other formats
- Each character in hex represents a 4-bit value
- The codes here use eight characters, so they are 32 bits long

Imitating Remote Control Signals

- LED is connected with a current-limiting resistor
- to increase the sending range, use multiple LEDs or select one with greater output
- you won’t see anything when the codes are sent because the light from the IR LED isn’t visible to the naked eye.
- verify that an infrared LED is working with a digital camera
- you should be able to see it flashing in the camera’s LCD viewfinder.



Imitating Remote Control Signals

```
#include <IRremote.h>
const int NUM_OF_BUTS = 21;
long BUTTON_CODES[NUM_OF_BUTS] = {
    0xE0E08877, //button 0
    0xE0E020DF, //button 1
    0xE0E0A05F, //button 2
    0xE0E0609F, //button 3
    0xE0E010EF, //button 4
    0xE0E0906F, //button 5
    0xE0E050AF, //button 6
    0xE0E030CF, //button 7
    0xE0E0B04F, //button 8
    0xE0E0708F, //button 9
    0xE0E08679, //button 100+/DOWN
    0xE0E006F9, //button 200+/UP
    0xE0E0D02F, //button VOL-
    0xE0E0E01F, //button VOL+
    0xE0E016E9, //button EQ/MENU
    0xE0E0A659, //button PREV/LEFT
    0xE0E046B9, //button NEXT/RIGHT
    0xE0E0F00F, //button PAUSE/MUTE
    0xE0E008F7, //button CH-
    0xE0E048B7, //button CH+
    0xE0E040BF  //button CH/POWER
  };
```

```
IRsend irsend;
void setup(){
}
void loop() {
  for (int i = 0; i < NUM_OF_BUTS; i++) {
    irsend.sendSony(BUTTON_CODES[i], 32);
    delay(40);
    //half second delay between each signal burst
  }
}
```

Liquid crystal displays (LCD)

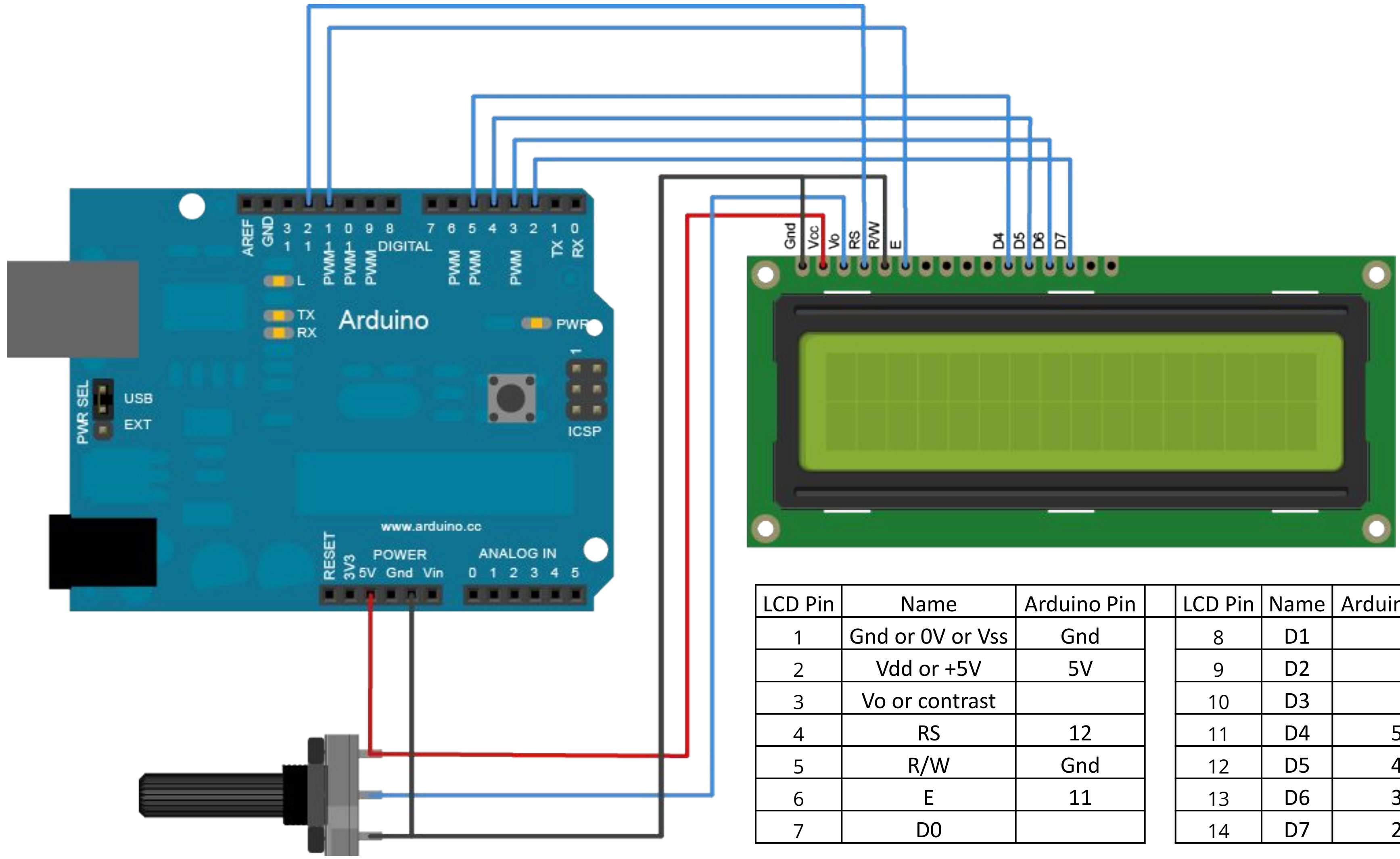
- offer a convenient and inexpensive way to provide a user interface for a project
- is the text panel that displays two or four lines of text, with 16 or 20 characters per line
- library for driving LCD is provided with Arduino
- can do more than display simple text
- can be scrolled or highlighted
- can display special symbols and non-English characters

LCD vs. Graphical Display

- create your own symbols and block graphics with a text LCD
- small resolution (5x9, 7x12, 8x16)

- graphical LCD has fine graphical details
- can display up to eight lines of 20 text characters in addition to graphics

Connecting and Using a Text LCD Display



LCD Pin	Name	Arduino Pin
1	Gnd or 0V or Vss	Gnd
2	Vdd or +5V	5V
3	Vo or contrast	
4	RS	12
5	R/W	Gnd
6	E	11
7	D0	

LCD Pin	Name	Arduino Pin
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2

Connecting and Using a Text LCD Display

```
#include <LiquidCrystal.h> // include the library code
//constants for the number of rows and columns in the LCD
const int numRows = 2;
const int numCols = 16;
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //RS, E, D4, D5, D6, D7
void setup(){
  lcd.begin(numCols, numRows);
  lcd.print("hello, world!"); // Print a message to the LCD.
}
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);
}
```

If you don’t see any text and you have double-checked that all wires are connected correctly, you may need to adjust the contrast pot. With the pot shaft rotated to one side (usually the side connected to Gnd), you will have maximum contrast and should see blocks appear in all the character positions. With the pot rotated to the other extreme, you probably won’t see anything at all. The correct setting will depend on many factors, including viewing angle and temperature—turn the pot until you get the best looking display

LCD library functions

- LiquidCrystal(rs, enable, d4, d5, d6, d7) - Creates a variable of type LiquidCrystal and initializes it with indicated pins
- lcd.begin(cols, rows) - Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display
- lcd.clear() - Clears the LCD screen and positions the cursor in the upper-left corner
- lcd.home() - Positions the cursor in the upper-left of the LCD

LCD library functions

- lcd.setCursor(col, row) - set the location (column and row) at which subsequent text written to the LCD will be displayed
- lcd.write(data) - Write a custom character to the LCD
- lcd.print(data) - Prints text to the LCD.
- lcd.cursor() / lcd.noCursor() - Set cursor visible/invisible
- lcd.blink() / lcd.noBlink() - Turns blinking of the cursor on/off
- lcd.display() / lcd.noDisplay() - Turns on/off the LCD display
- lcd.scrollDisplayLeft() / lcd.scrollDisplayRight() - Scrolls the contents of the display one space to the left/right
- lcd.createChar(num, data) - Create a custom character specified by data. Up to eight characters of 5x8 pixels are supported (num - 0 to 7)

Formatting Text

```
#include <LiquidCrystal.h> // include the library code:
//constants for the number of rows and columns in the LCD
const int numRows = 2;
const int numCols = 16;
int count;
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup(){
 lcd.begin(numCols, numRows);
 lcd.print("Starting in "); // this string is 12 characters long
 for(int i=9; i > 0; i--){ // count down from 9
   // the top line is row 0
   lcd.setCursor(12,0); // move the cursor to the end of the string
   lcd.print(i);
   delay(1000);
 }
}
void loop(){
 int columnWidth = 4; //spacing for the columns
 int displayColumns = 3; //how many columns of numbers
 lcd.clear();
 for(int col=0; col < displayColumns; col++){
   lcd.setCursor(col * columnWidth, 0);
   count = count + 1;
   lcd.print(count);
 }
 delay(1000);
}
```

Turning the Cursor and Display On or Off

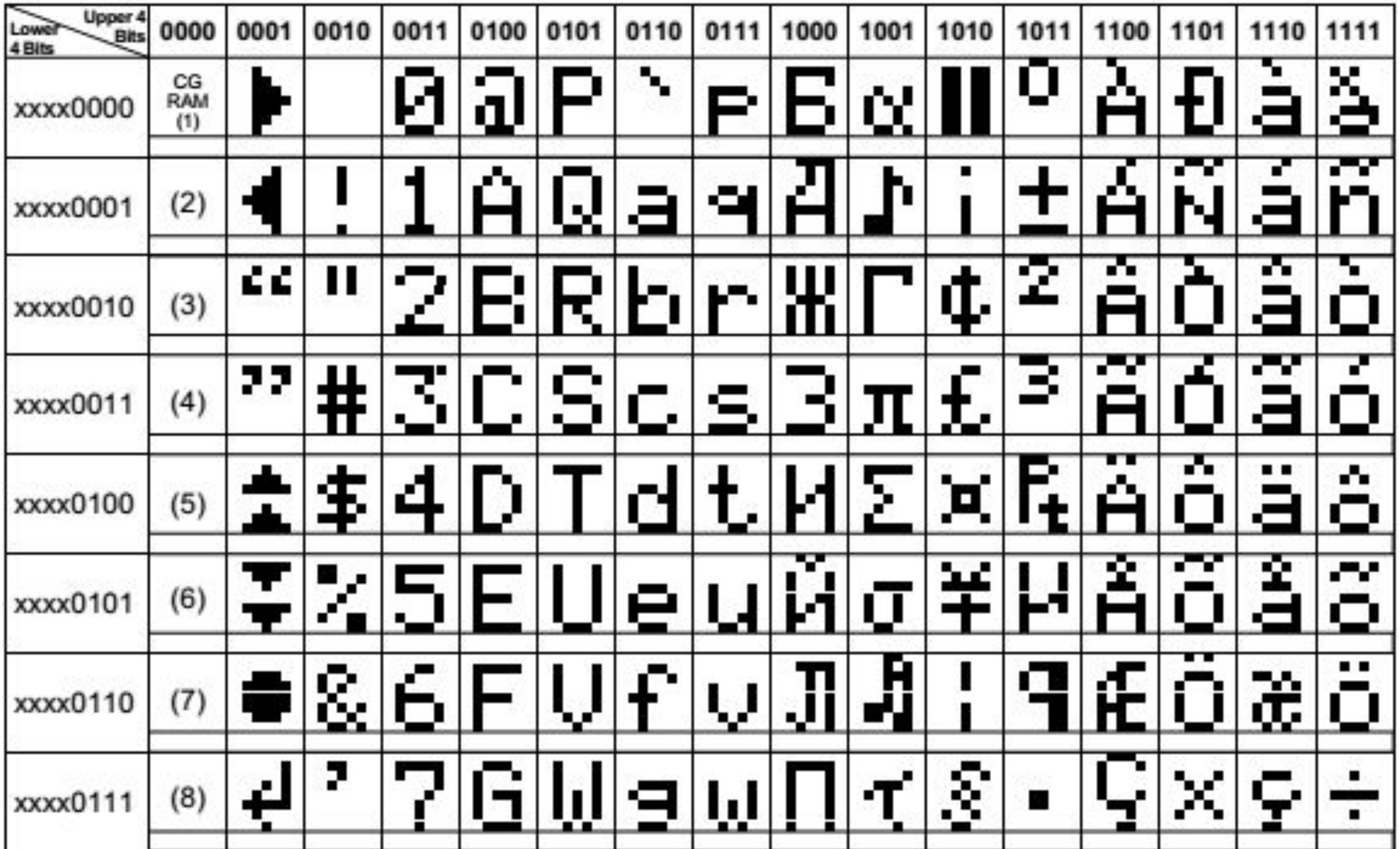
```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup(){
 lcd.begin(16, 2);
 lcd.print("hello, world!");
}
void displayBlink(int blinks, int duration){
 while(blinks--){
  lcd.noDisplay();
  delay(duration);
  lcd.display();
  delay(duration);
 }
}
```

```
void loop(){
 lcd.setCursor(0, 1);
 lcd.print("cursor blink");
 lcd.blink();
 lcd.setCursor(0, 1);
 lcd.print("noBlink");
 delay(2000);
 lcd.noBlink();
 delay(2000);
 lcd.clear();
 lcd.print("Display off ...");
 delay(1000);
 lcd.noDisplay();
 delay(2000);
 lcd.display();
 // turn the display back on
 lcd.setCursor(0, 0);
 lcd.print(" display flash !");
 displayBlink(2, 250);
 // blink twice
 displayBlink(2, 500);
 // and again for twice as long
 lcd.clear();
}
```

Displaying Special Symbols

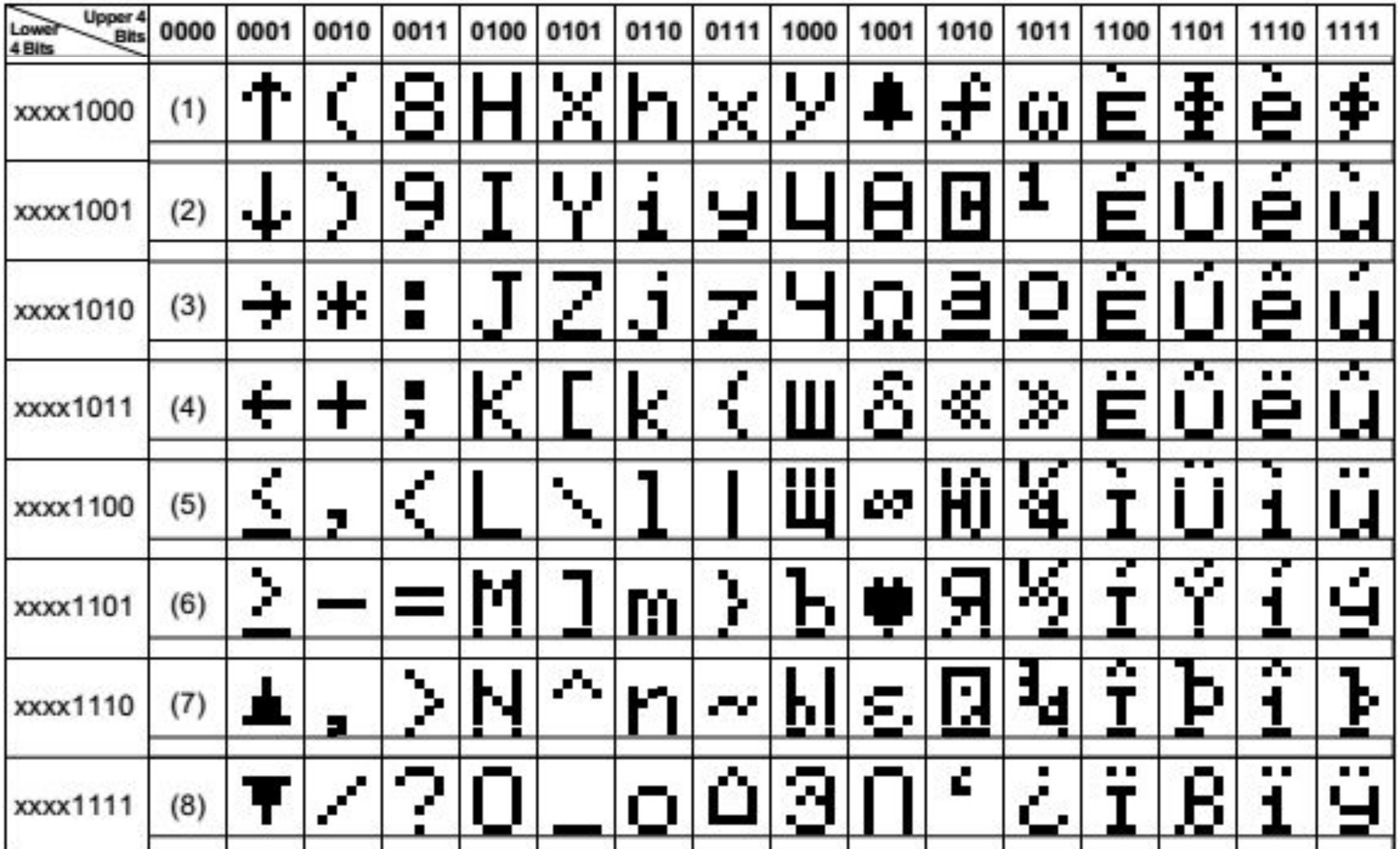
```
#include <LiquidCrystal.h>
const int numRows = 2;
const int numCols = 16;
const byte degreeSymbol = B11011111;
const byte piSymbol = B11110111;
const byte centsSymbol = B11101100;
const byte sqrtSymbol = B11101000;
const byte omegaSymbol = B11110100; // the symbol used for ohms
byte charCode = 32; // the first printable ascii character
int col; int row;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup(){
 lcd.begin(numRows, numCols);
}
void loop(){
 showSymbol(degreeSymbol, "degrees");
 showSymbol (piSymbol, "pi");
 showSymbol(centsSymbol, "cents");
 showSymbol(sqrtSymbol, "sqrt");
 showSymbol(omegaSymbol, "ohms");
}
void showSymbol( byte symbol, char * description){
 lcd.clear();
 lcd.write(symbol);
 lcd.print(' '); // add a space before the description
 lcd.print(description);
 delay(3000);
}
```

Character codes



Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)	▶		0	@	P	\`	p	Б	α	‖	°	À	Đ	à	ð
xxxx0001	(2)	◀	!	1	A	Q	a	q	Д	♪	¡	±	Á	Ñ	á	ñ
xxxx0010	(3)	“	"	2	B	R	b	r	Ж	Г	¢	²	Â	Ò	â	ò
xxxx0011	(4)	”	#	3	C	S	c	s	З	π	£	³	Ã	Ó	ã	ó
xxxx0100	(5)	⏫	$	4	D	T	d	t	И	Σ	¤	₧	Ä	Ô	ä	ô
xxxx0101	(6)	⏬	%	5	E	U	e	u	Й	σ	¥	µ	Å	Õ	å	õ
xxxx0110	(7)	●	&	6	F	V	f	v	Л	♫	¦	¶	Æ	Ö	æ	ö
xxxx0111	(8)	↵	’	7	G	W	g	w	П	τ	§	·	Ç	×	ç	÷

Character codes



Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx1000	(1)	↑	(	8	H	X	h	x	Y	🔔	ƒ	ω	È	Φ	è	φ
xxxx1001	(2)	↓	)	9	I	Y	i	y	Ц	Б	©	¹	É	Ù	é	ù
xxxx1010	(3)	→	*	:	J	Z	j	z	Ч	Ω	ª	º	Ê	Ú	ê	ú
xxxx1011	(4)	←	+	;	K	[	k	{	Ш	δ	«	»	Ë	Û	ë	û
xxxx1100	(5)	≤	,	<	L	\	l	|	Щ	∞	Ю	¼	Ì	Ü	ì	ü
xxxx1101	(6)	≥	−	=	M	]	m	}	Ъ	♥	Я	½	Í	Ý	í	ý
xxxx1110	(7)	▲	,	>	N	^	n	~	Ы	Ξ	®	¾	Î	Þ	î	þ
xxxx1111	(8)	▼	/	?	O	_	o	⌂	Э	Π	ʿ	¿	Ï	ß	ï	ÿ

Creating Custom Characters

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
byte happy[8] =
{
 B00000,
 B10001,
 B00000,
 B00000,
 B10001,
 B01110,
 B00000,
 B00000
};
byte saddy[8] =
{
 B00000,
 B10001,
 B00000,
 B00000,
 B01110,
 B10001,
 B00000,
 B00000
};
```

```
void setup() {
 lcd.createChar(0, happy);
 lcd.createChar(1, saddy);
 lcd.begin(16, 2);
}
void loop() {
 for (int i=0; i<2; i++){
   lcd.setCursor(0,0);
   lcd.write(i);
   delay(500);
 }
}
```