



# Physics 124: Lecture 4

LCD Text Display  
Keypads and Time Slicing  
Interrupts

adapted from T. Murphy's lectures

# 2×16 LCD

- Typically 5×8 dots per character
- Note 16 pins: indicator of common interface



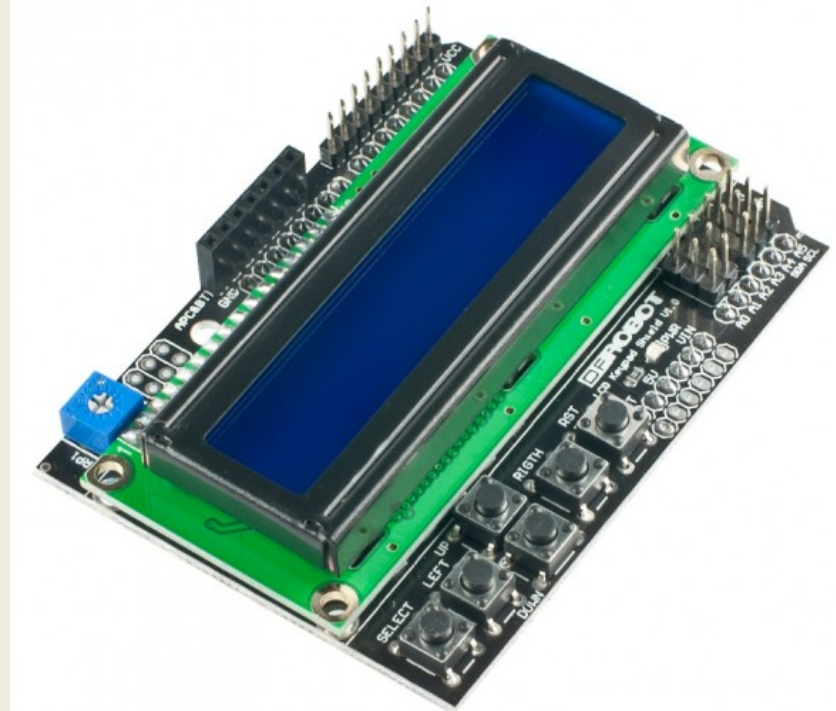
# Typical LCD Unit pinout

pin	function	Arduino pin (shield)
1	ground	GND
2	+5 V	+5 V
3	VEE (contrast via potentiometer between 0 and 5 V)	pot on shield
4	Register Select (LOW = command; HIGH = data/characters)	8
5	RW (LOW = Write; HIGH = Read)	GND
6	E (Enable strobe: toggle to load data and command)	9
7-14	data bus	4,5,6,7 → D4,D5,D6,D7
15	backlight +V	
16	backlight ground	

Note that most features are accessible using only the 4 MSB data pins

# Arduino LCD Shield

- Handy package, includes buttons, contrast pot, some pins/headers for other connections
  - consumes Arduino pins 4, 5, 6, 7, 8, 9
  - leaves 0, 1 for Serial, 2, 3, 10, 11, 12, 13
    - fails to make pin 10 available on header, though

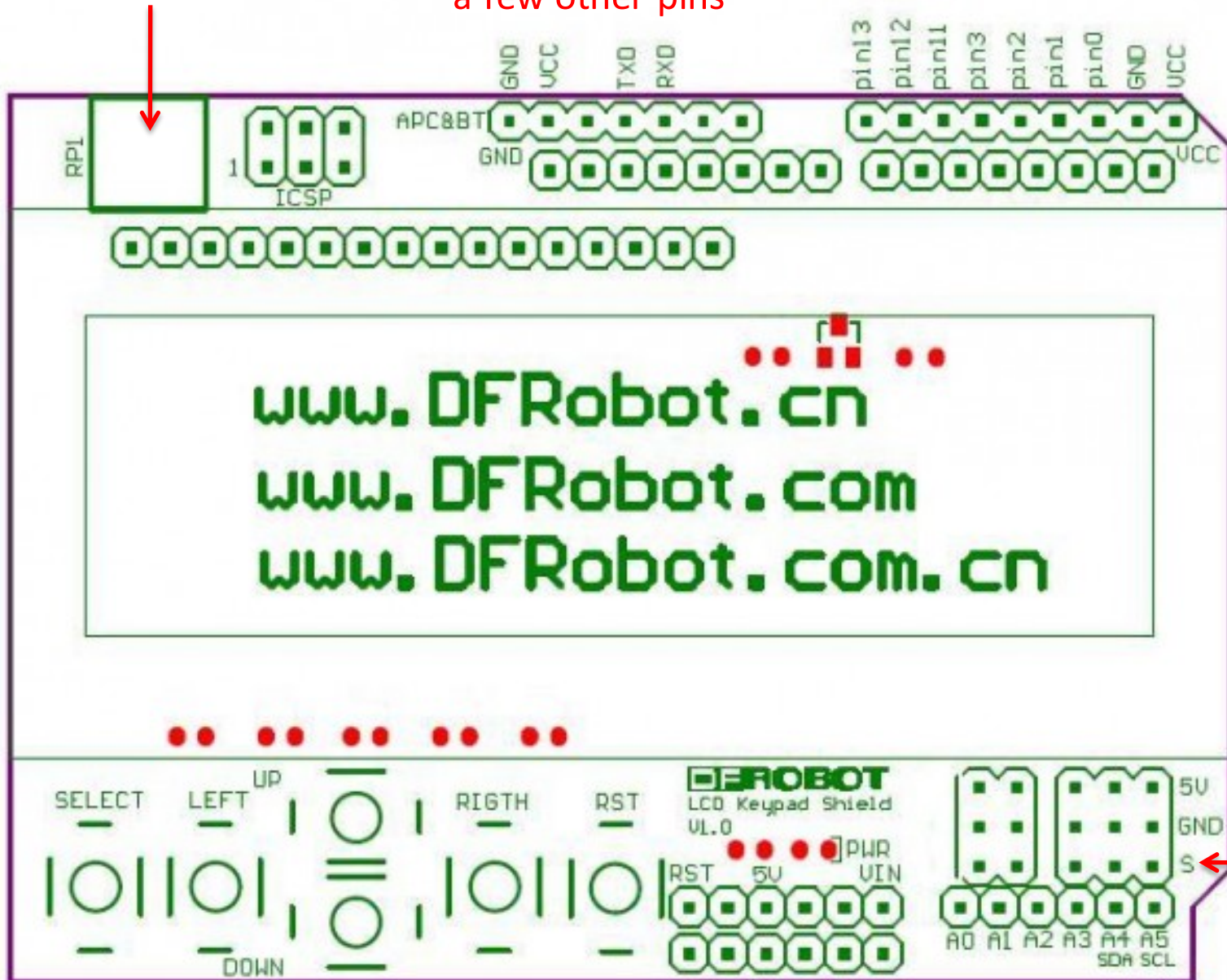




contrast adjust

a few other pins

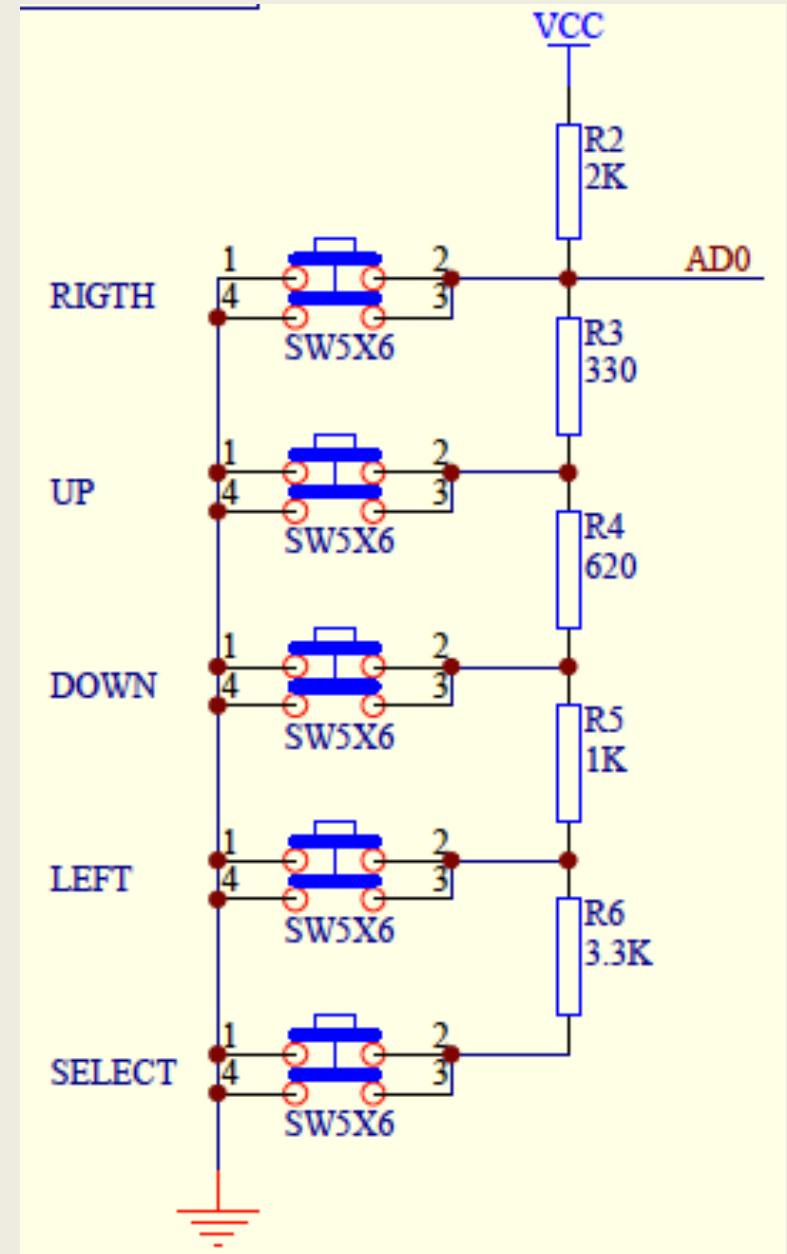
Arduino pin breakout



buttons utilize A0 analog input

# Buttons

- The buttons use a voltage divider tree to present an analog voltage to A0
  - note “RIGTH” typo made it onto printed circuit board!
- Tom measured the following:
  - none: 4.95 V
  - SELECT: 3.59 V
  - LEFT: 2.44 V
  - DOWN: 1.60 V
  - UP: 0.70 V
  - RIGHT: 0.0 V
- Easily distinguishable



# LCD Datasheet

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	1	P	`	P	B	α		°	À	0	À	À	À
xxxx0001	(2)		!	1	A	Q	a	9	A	J	i	±	À	Ñ	À	Ñ	Ñ
xxxx0010	(3)	“	”	2	B	R	b	r	Ж	Г	Φ	2	À	Ò	À	Ò	Ò
xxxx0011	(4)	”	#	3	C	S	c	s	3	π	£	3	À	Ó	À	Ó	Ó
xxxx0100	(5)		\$	4	D	T	d	t	H	Σ	⌘	⌘	À	Ô	À	Ô	Ô
xxxx0101	(6)		%	5	E	U	e	u	Й	σ	¥	μ	À	Ö	À	Ö	Ö

- For behind-the-scenes control of the LCD display, see the datasheet
  - [http://physics124.barreiro.ucsd.edu/wp-content/uploads/sites/41/2017/01/LCD\\_HD44780.pdf](http://physics124.barreiro.ucsd.edu/wp-content/uploads/sites/41/2017/01/LCD_HD44780.pdf)
- Above is just one snippet of the sort of things within

# And one other snippet from LCD datasheet

Character Codes (DDRAM data)								CGRAM Address								Character Patterns (CGRAM data)								
7	6	5	4	3	2	1	0	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
High				Low				High				Low				High				Low				
0 0 0 0 * 0 0 0								0 0 0				0	0	0	↑ ↓	*	*	*	1	1	1	1	0	Character pattern (1)
												0	0	1		1	0	0	0	1				
												0	1	0		1	0	0	0	1				
												0	1	1		1	1	1	1	0				
												1	0	0		1	0	0	0					
												1	0	1		1	0	1	0					
												1	1	0		0	0	1						
												1	1	1		0	0	0						
0 0 0 0 * 0 0 1								0 0 1				0	0	0	*	*	*	1	0	0	0	1	Character pattern (2)	
												0	0	1	0	1	0	0						
												0	1	0	1	1	1	1						
												0	1	1	0	0	0							
												1	0	0	1	1	1							
												1	0	1	0	0	0							
												1	1	0	1	0	0							
												1	1	1	0	0	0							
												0	0	0	*	*	*	0	0	0	0	0	Cursor position	
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	*	*	*	0	0	0	0	0	Cursor position	
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						
												0	0	0	0	0	0	0						

- Datasheets: they build character (at least characters)



# The LiquidCrystal Library

- This is one place few are itching for low-level control
  - or wait—where's the fun/challenge in *that* attitude?
- Library makes simple

```
#include <LiquidCrystal.h>

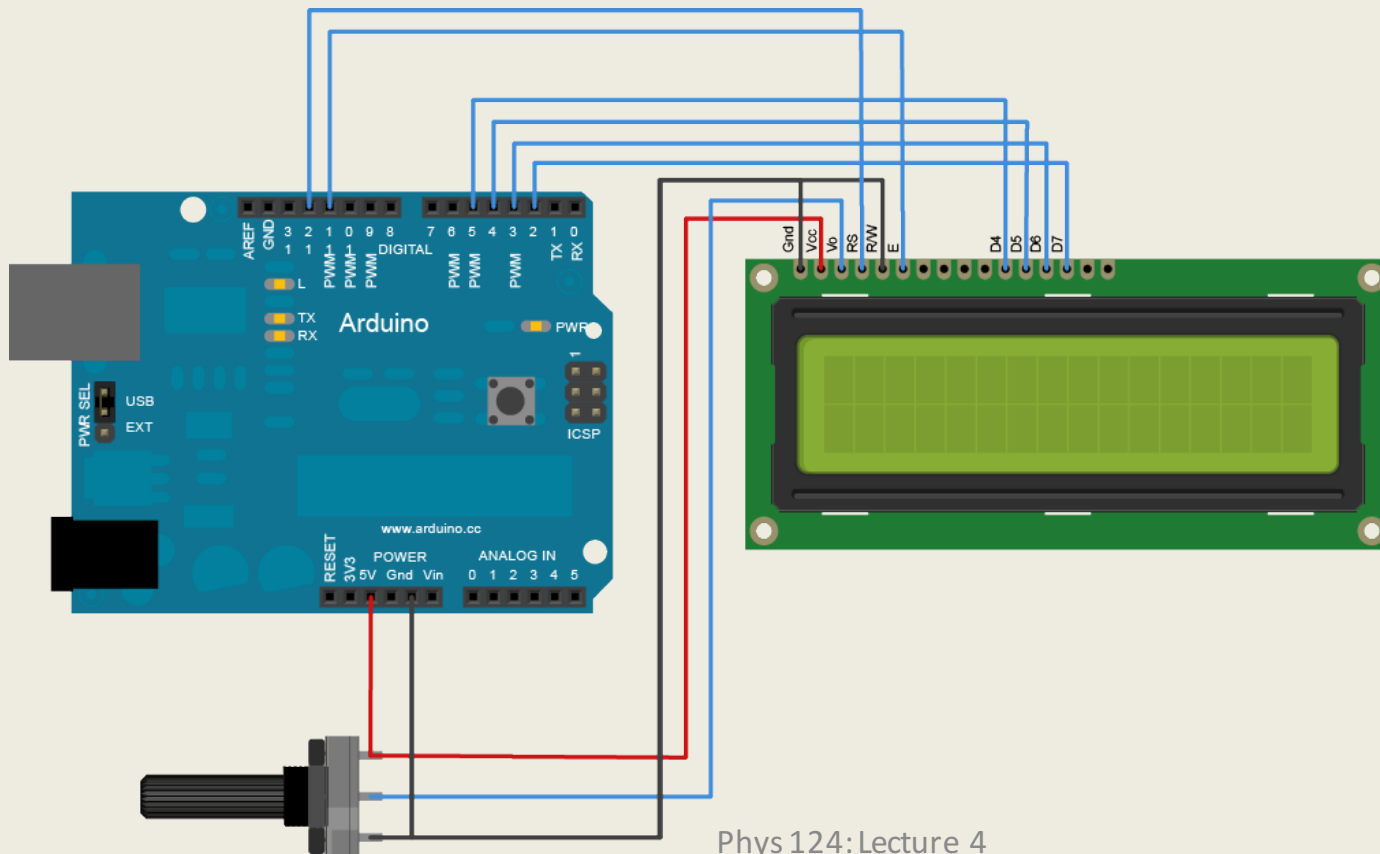
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // matches shield config

void setup() {
    lcd.begin(16, 2);           // # columns & rows
    lcd.print("Phys 124 Rules!");
}

void loop() {
    lcd.setCursor(0, 1);       // first col, second row (0 base)
    // print the number of seconds since reset:
    lcd.print(millis()/1000);
}
```

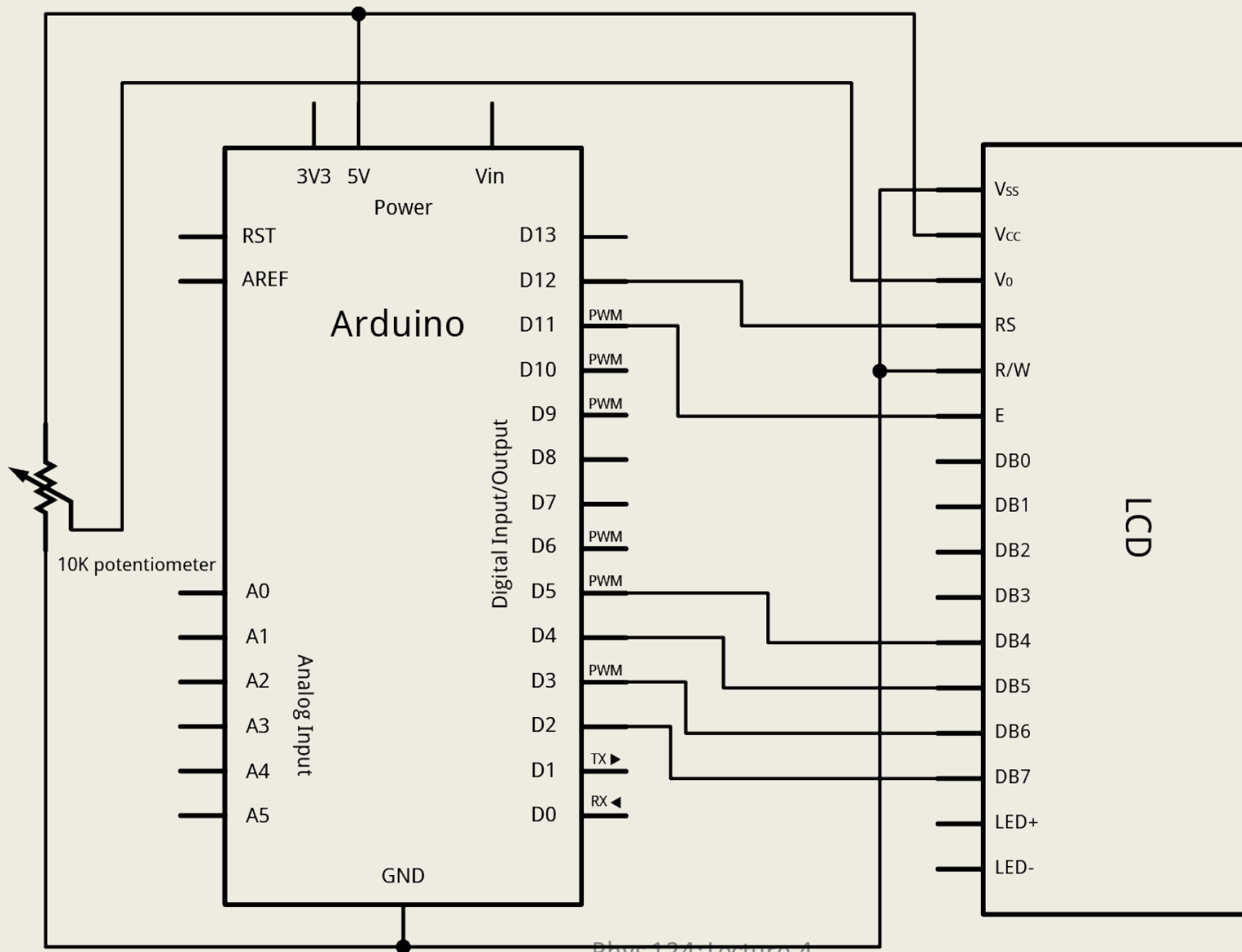
# The setup call

- Arguments in LiquidCrystal type are:
  - pins corresponding to: Register Select, Enable, D4, D5, D6, D7
  - don't need shield at all; just those 6 pins and power/gnd
  - here's one without shield: must hook R/W to gnd; rig pot



# Same thing in schematic form

- Note this pinout is different than shield's mapping



# Explore the library

- Can do a lot with a few functions, but more available
  - [LiquidCrystal\(\)](#) must use
  - [begin\(\)](#) must use
  - [clear\(\)](#)
  - [home\(\)](#)
  - [setCursor\(\)](#) almost certainly use
  - [write\(\)](#)
  - [print\(\)](#) almost certainly use
  - [cursor\(\)](#)
  - [noCursor\(\)](#)
  - [blink\(\)](#)
  - [noBlink\(\)](#)
  - [display\(\)](#)
  - [noDisplay\(\)](#)
  - [scrollDisplayLeft\(\)](#)
  - [scrollDisplayRight\(\)](#)
  - [autoscroll\(\)](#)
  - [noAutoscroll\(\)](#)
  - [leftToRight\(\)](#)
  - [rightToLeft\(\)](#)
  - [createChar\(\)](#)

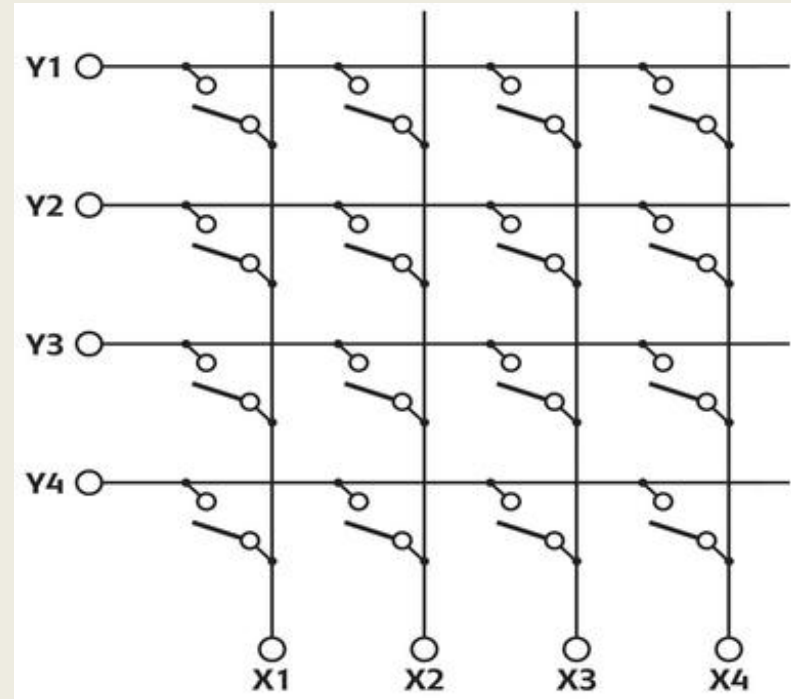
# LCD References

- Good general intro to LCD control
  - [http://spikenzielabs.com/SpikenzieLabs/LCD\\_How\\_To.html](http://spikenzielabs.com/SpikenzieLabs/LCD_How_To.html)
- Arduino page
  - <http://arduino.cc/en/Tutorial/LiquidCrystal>
- See links on course site:
  - <https://physics124.barreiro.ucsd.edu/doc-links/>
    - [LCD shield schematic](#)
    - [LCD datasheet](#)
- The goal is to get you started with one kind of displays and explore others on your own and our help.



# Keypads

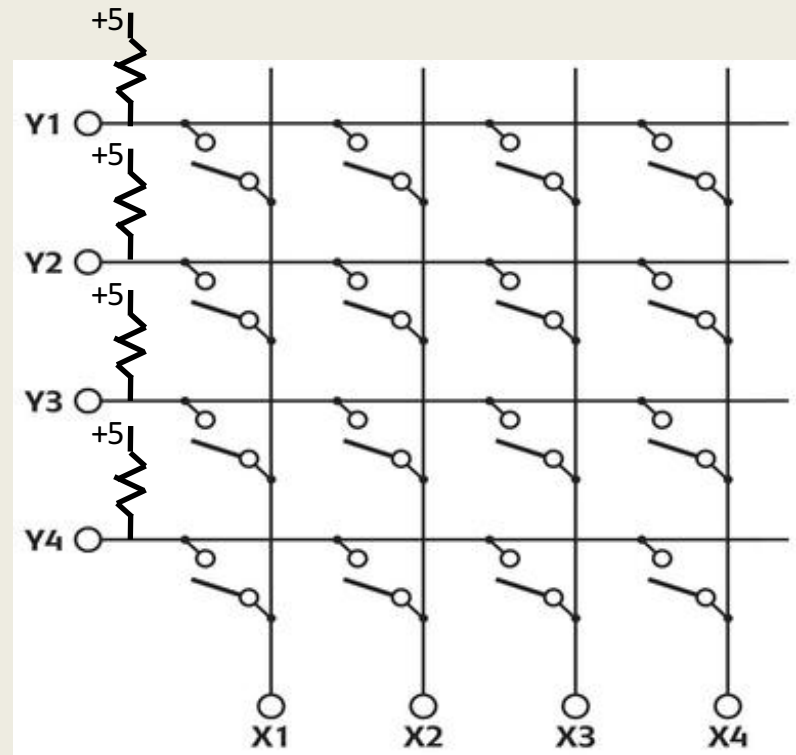
- Most keypads are matrix form: row contact and column contact
  - pressing button connects one row to one column



note crossings do not connect:  
dots indicate connection

# Reading the keypad

- Imagine we hooked the rows (Y) to four digital inputs with pull-up resistors
  - and hooked the columns (X) up to digital outputs
- Now cycle through X, putting each to zero (LOW) in turn
  - otherwise enforce high state
- Read each row value and see if any inputs are pulled low
  - means switch closed, button pressed
- Called time-slicing



# Those Pesky Pullups

- Arduino has a `pinMode` option to engage internal pullup resistors

- `pinMode(pin, INPUT_PULLUP);`
  - does just what we want

- Let's start by defining our pins (example values)
  - and our key characters

```
#define ROW1 12          // or whatever pin is hooked to row1
etc.
#define COL1 8
etc.
#define ROWS 4
#define COLS 4
char keys[ROWS][COLS] = {           // handy map of keys
    {'1','2','3','A'},              // black 4x4 keypad
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};
int pressed, last, row, col, ch; // variables used later
```

# Now set up pins in setup( )

```
pinMode(ROW1, INPUT_PULLUP);
```

etc.

```
pinMode(COL1, OUTPUT);
```

etc.

```
digitalWrite(COL1, HIGH); // def. state is high; start high
```

```
Serial.begin(9600); // We will use Serial Monitor
```

- Now in loop()

```
pressed = 0;
```

```
// value for no press
```

```
// row/col encoded in 8 bits
```

```
// assert col 1 low
```

```
digitalWrite(COL1, LOW);
```

```
if (digitalRead(ROW1) == LOW)
```

```
    pressed = 0x11;
```

```
// upper digit is row
```

```
if (digitalRead(ROW2) == LOW)
```

```
    pressed = 0x21;
```

```
// lower digit is col
```

etc.

```
digitalWrite(COL1, HIGH);
```

```
// reset col1 to high
```

etc. for all 4 columns; the scheme for `pressed` is just one way, my first impulse

# Piecing together at end of loop

```
if (pressed != 0 && pressed != last)
{
    row = pressed >> 4;           // row/col encoded in 8 bits
    col = pressed & 0x0f;         // drop 4 LSB, look at upper 4
    ch = keys[row-1][col-1];      // kill upper 4 bits; keep 4 LSB
    if (ch != '#')               // get character from map
        Serial.print(ch);        // treat # as newline
    else
        Serial.println("");      // just want return
}
last = pressed;                  // preserve knowledge
delay(40);                       // debounce delay
```

- print only if new press, new line if '**#**' pressed
  - note **>>** bit shift row look at high nibble;
  - and mask lower 4 bits for isolating lower nibble
  - thus decode into row and column (at least this is *one* way)



# Cleaning up code

- Repeating the sweep four times during the loop is a bit clumsy, from a coding point of view
  - begs to be function()-ized

```
int readCol(int column)
{
    int row_press = 0;
    digitalWrite(column, LOW);
    if (digitalRead(ROW1) == LOW)
        row_press = 1;
    if (digitalRead(ROW2) == LOW)
        row_press = 2;
    etc. // repeat for each row
    digitalWrite(column, HIGH);

    return row_press;
}
```

# Now a function to sweep columns

```
int sweepCols()
{
    int row_press;    // keep track of row
    int pressed = 0;  // returned key coordinates

    row_press = readCol(COL1);
    if (row_press > 0)
        pressed = (row_press << 4) + 1;
    etc.
    row_press = readCol(COL4);
    if (row_press > 0)
        pressed = (row_press << 4) + 4;

    return pressed;
}
```

now in main loop, just: `pressed = sweepCols();` and otherwise same

# And, there's a Library

- Of course there is...

On Arduino IDE (1.6 or above, in the labs we have >1.8):

**Sketch->Include Library->Manage Libraries...** Then search for *Keypad* & install.

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {{ '1', '2', '3'}, { '4', '5', '6'},
                          { '7', '8', '9'}, { '#', '0', '*'}};

byte rowPins[ROWS] = {5, 4, 3, 2}; //conn. to the row pins of the keypad
byte colPins[COLS] = {8, 7, 6}; //conn. to the col pins of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup(){
  Serial.begin(9600);}

void loop(){
  char key = keypad.getKey();
  if (key != NO_KEY)
    Serial.println(key);
}
```

# Some Notes on the Keypad Library

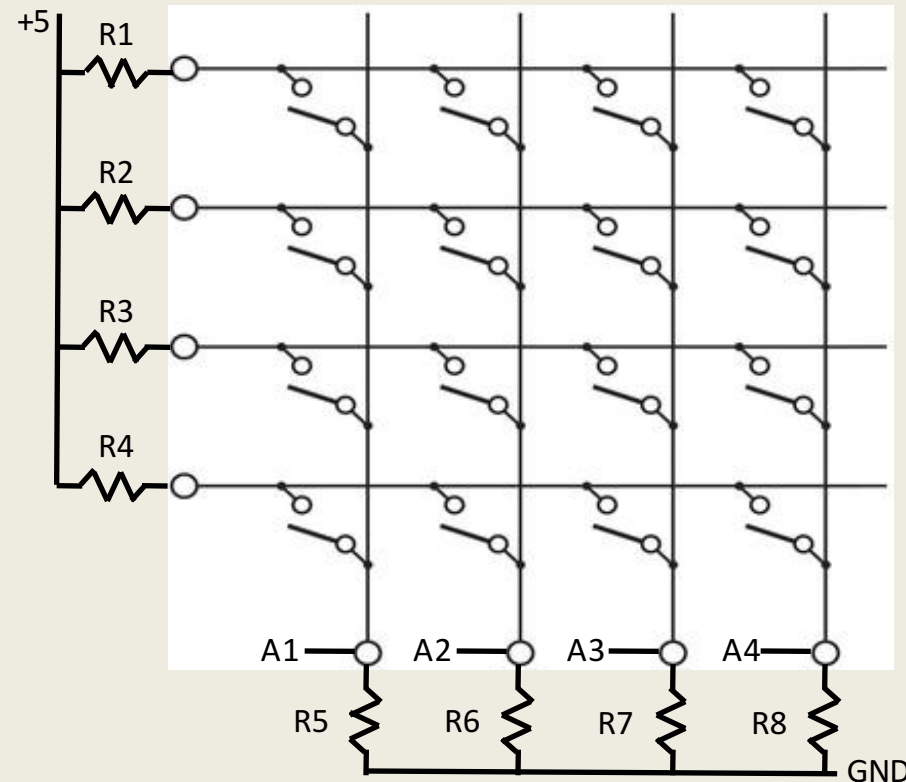
- Note that the key map is taken seriously by Keypad.h
  - if any character appears twice, it messes up
  - therefore more than a printing convenience; a core functional element of the operation
- Functions
  - `void begin(makeKeymap(userKeymap))`
  - `char waitForKey()`
  - `char getKey()`
  - `KeyState getState()`
  - `boolean keyStateChanged()`
  - `setHoldTime(unsigned int time)`
  - `setDebounceTime(unsigned int time)`
  - `addEventListener(keypadEvent)`
- Consult link on previous slide for descriptions

# Combining LCD and Keypad?

- The LCD uses **six** digital pins
- A 4x4 keypad needs **8** pins
- Uno has **14**, but pins 0 and 1 are used by Serial
  - could forgo serial communications, and max out pins
- Need a better way, *less greedy*
- Take a page from LCD shield buttons: use analog input
- Many schemes are possible
  - generally: **+5 V** on rows/cols, **GND** on other, resistors between
  - could have all 16 buttons map to a **single** analog input
    - interesting problem in designing appropriate network (done last year by one team)
  - or make it easier and map to four analog inputs



# Four-Input Scheme



- R1 thru R4 could be 10 k $\Omega$ , 4.7 k $\Omega$ , 2.2 k $\Omega$ , 1 k $\Omega$
- R5 thru R8 could be all 3.3 k $\Omega$ , or in that ballpark
  - voltages will be 0 (nothing pressed), 1.25 V (top row), 2.06V; 3 V; and 3.8 V for resp. rows — lots of separation
- Poll each A# input to ascertain keypress

# Interrupts

- Sometimes we can't afford to miss a critical event, while the main loop is busy, or in a delay, etc.
- Interrupts demand immediate attention
- Uno has two interrupts
  - int.0 on pin 2; int.1 on pin 3
  - Mega has 6 available interrupts
- You can exempt some of loop from interruption
  - may be rare that you need to do this, but...

```
void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

# Easily implemented

- Just have to attach an interrupt to a service routine
  - `attachInterrupt(int#, function, trigger_type);`
  - the interrupt number is 0 or 1 on Uno (pins 2 or 3)\*\*
  - the function, or service routine, is some function you've created to service the interrupt: name it whatever makes sense
  - `trigger_type` can be
    - RISING: detects edge from logic low to logic high
    - FALLING: detects falling edge
    - CHANGE: any change between high/low (watch out for bounce!)
    - LOW: a low state will trigger an interrupt
  - note that `delay()` will not work within the service routine
    - need `delayMicroseconds()`, only good up to 16383  $\mu\text{s}$
    - but not often interested in delay in interrupt routine

# Simple example

- Turn on/off LED via interrupt; note volatile variable

```
int pin = 13;
volatile int state = LOW;

void setup()
{
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE); // interrupt 0 is pin 2
}

void loop()
{
    digitalWrite(pin, state);
    // careful with long delays here!
}

void blink()
{
    state = !state;
}
```

# Interrupt Notes

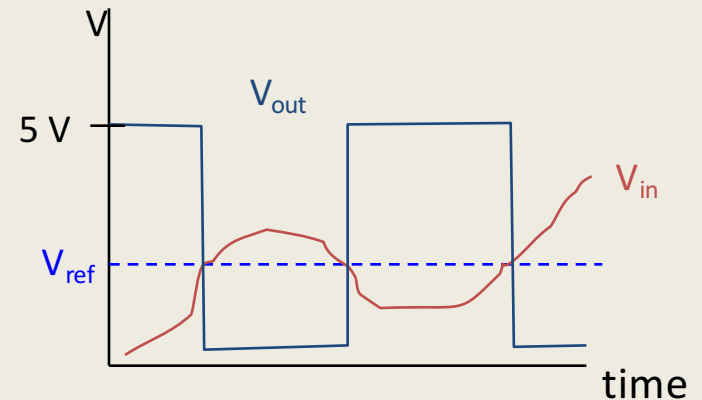
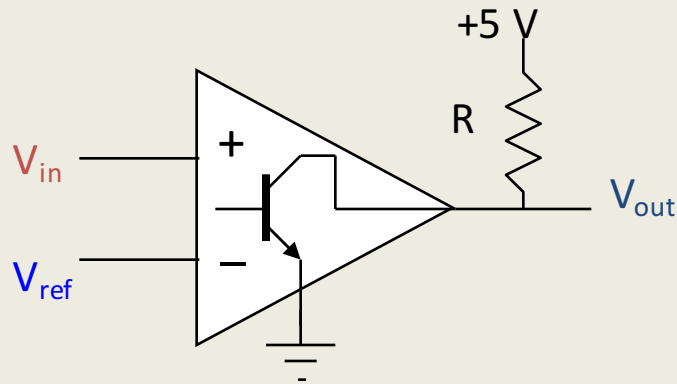
- *Inside the attached function, `delay()` won't work and the value returned by `millis()` will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function.*
- See the page for `attachInterrupt()`:
  - <http://arduino.cc/en/Reference/AttachInterrupt>



# Interrupts from analog?

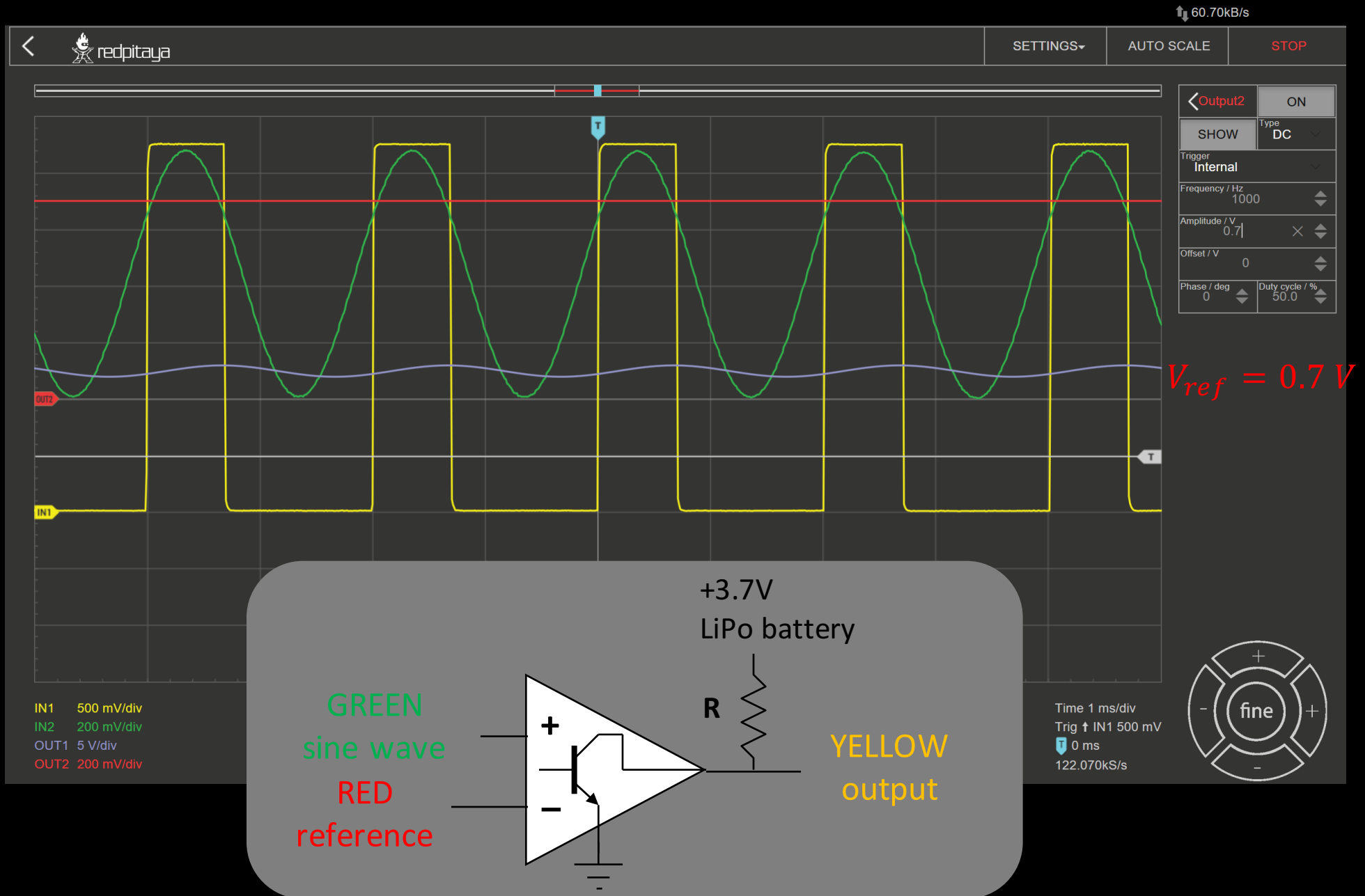
- What if we need to make a digital interrupt out of an analog signal like the analog-scheme keypad?
- Can use a *comparator* to sense if we're above or below some threshold voltage
  - output is *digital* state
  - could also use a high-pass (differentiator) to sense any significant *change* in the analog level, fed into a comparator

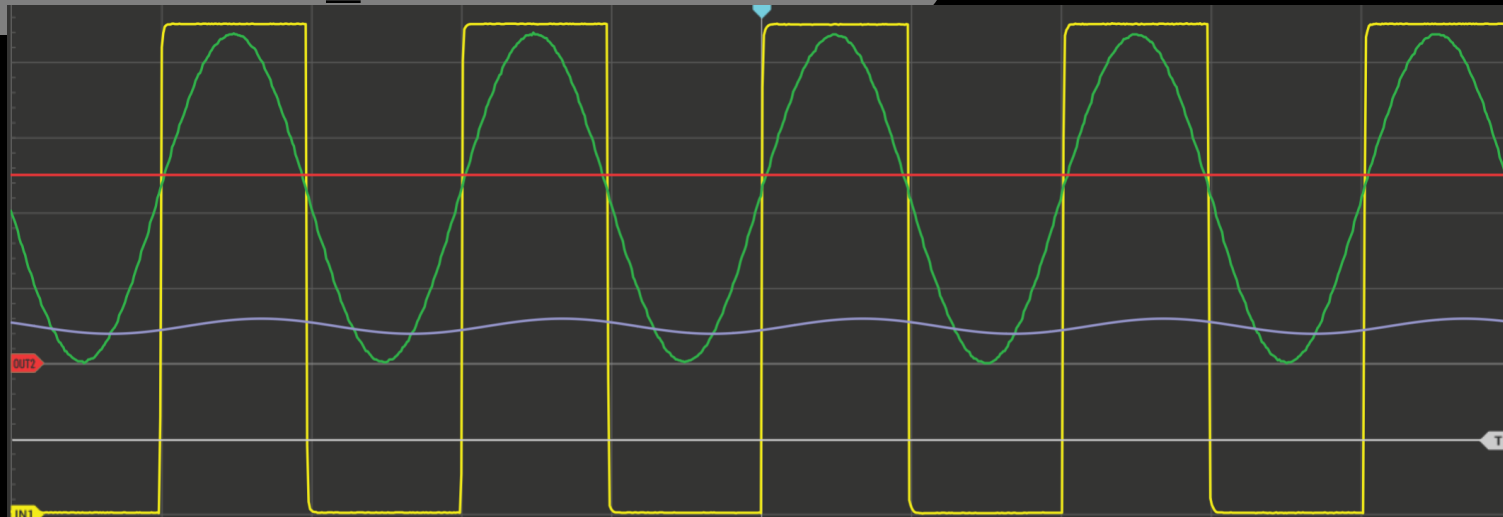
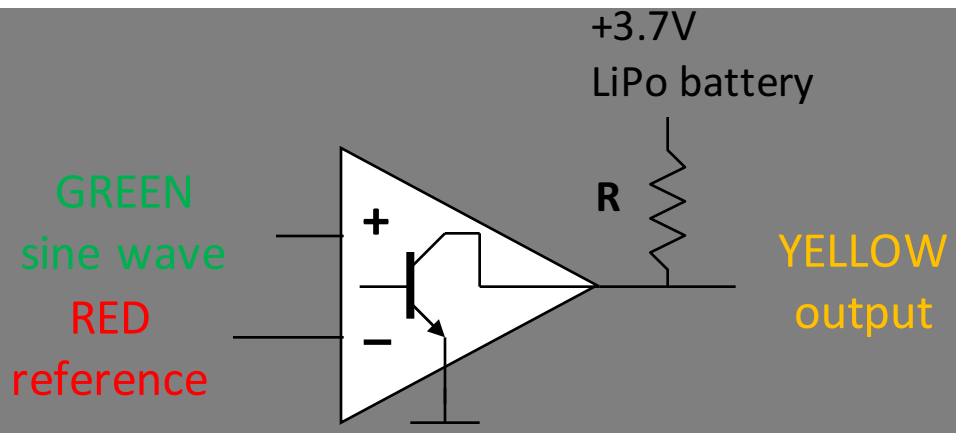
# Comparator Basics



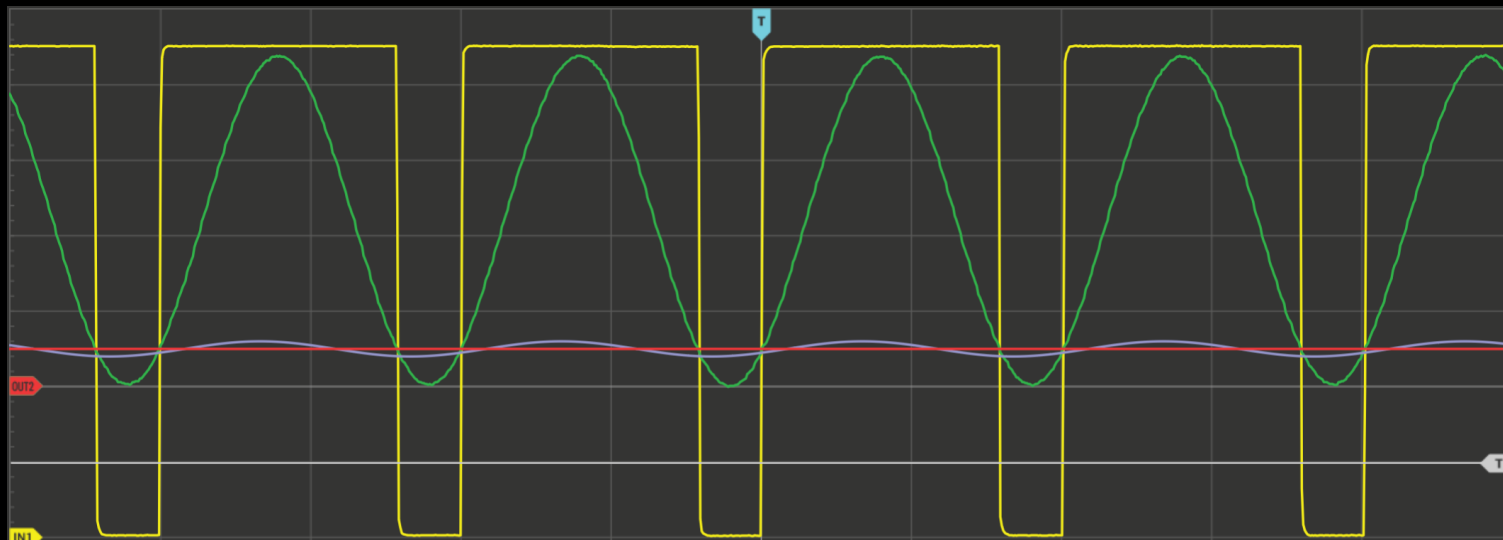
- Scheme is: when + input larger than - input, transistor driven to ON
  - then current flows through transistor and output is pulled low
- When  $V_{in} < V_{ref}$ ,  $V_{out}$  is pulled high (through the pull-up resistor—usually 1 k $\Omega$  or more)
  - this arrangement is called “open collector” output: the output is basically the collector of an npn transistor: in saturation it will be pulled toward the emitter (ground), but if the transistor is not driven (no base current), the collector will float up to the pull-up voltage
- The output is a “digital” version of the signal
  - with **settable** low and high values (here ground and 5V)

# Comparator Demo with RedPitaya





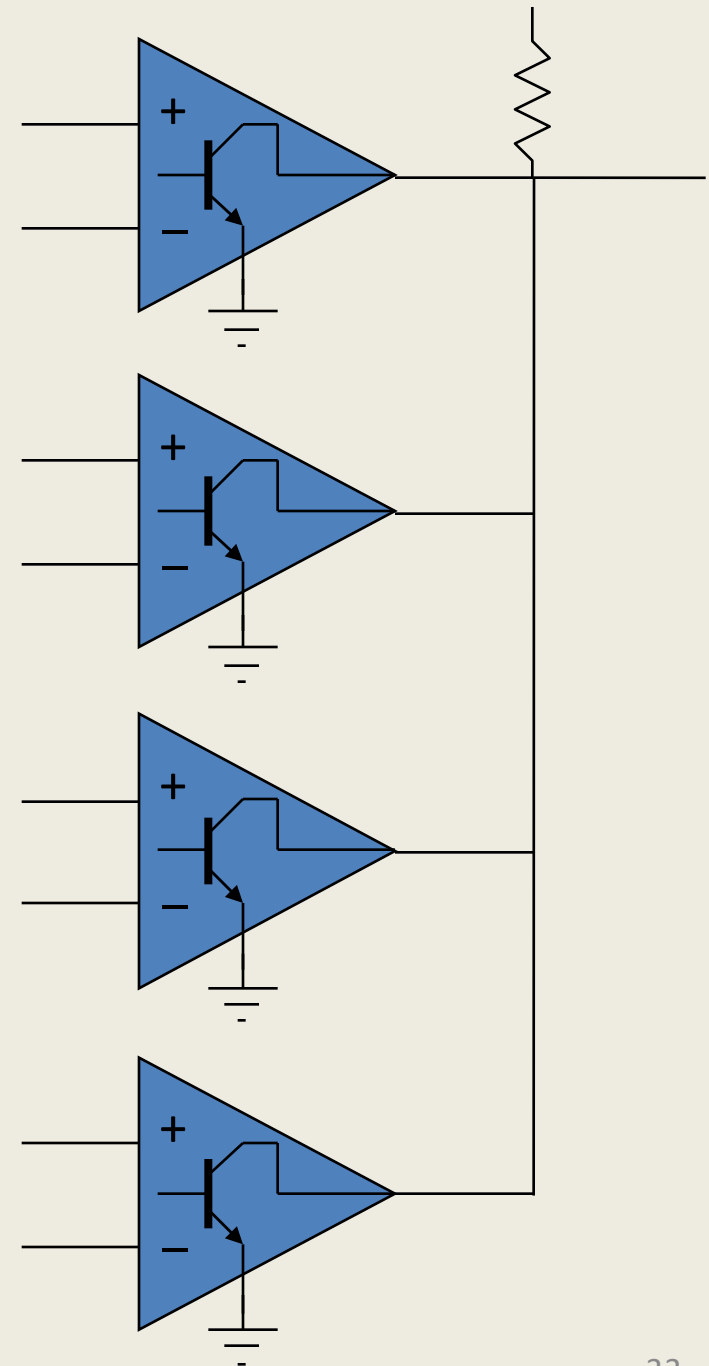
$$V_{ref} = 0.5 V$$



$$V_{ref} = 0.1 V$$

# Can Gang Open-Collector Comparators into Chain

- Put same (or different) threshold values on – inputs and four different analog signals on +
    - tie all four open collectors together with common pull-up
    - if any comparator activates, the associated transistor will pull the combined output low, and the other (off) transistors won't care
  - The “311” comparator is standard: LM311 ors LM339
    - LM311 *is obsolete!*
- Be careful how you use this! Think ;)



# Announcements

- Grades will be posted on TED
- All labs 1 turned in.
- In Week 3 lab, we will:
  - make an LCD analog voltage meter
  - read a 4x4 keypad using the time-slice method and 8 pins
  - combine the keypad, LCD, and interrupts into a party