University of British Columbia
Electrical and Computer Engineering
ELEC291/292

Lab 4: SPI, RS232, Temperature, Heart Rate, Strip Charts

Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

January 18, 2022

Objectives

- Use SPI devices with the AT89LP51RC2 microcontroller.
- Use the serial port to connect the AT89LP51RC2 to a computer and interchange information.
- ELEC291: Measure temperature with the LM335. Display in strip chart.
- ELEC292: Measure heart rate. Display in strip chart.

For this lab you can work with a partner.

The Serial Peripheral Interface (SPI)

- Used to add multiple external I/O devices or functions to the microcontroller.
- SPI devices are **extremely** simple to connect to most modern microcontrollers/microcomputers.
- SPI devices are usually smaller and cheaper than their parallel bus equivalents.
- Many SPI devices available. Go to Digikey/Mouser and search for 'SPI'.
- SPI article in Wikipedia is excellent!

SPI Synchronous Data Communication Data Format

- Data length can be any size but optimized for 8 bits on most 8-bit microcontrollers.
- Clock is transmitted in a separate wire. Both the polarity of the clock and its phase are selectable.
- Data and clock are transmitted at the same time. Data is also transmitted and received at the same time!
- Most microcontrollers, including the 8051, can be setup as master or slaves to implement master/slave networks.

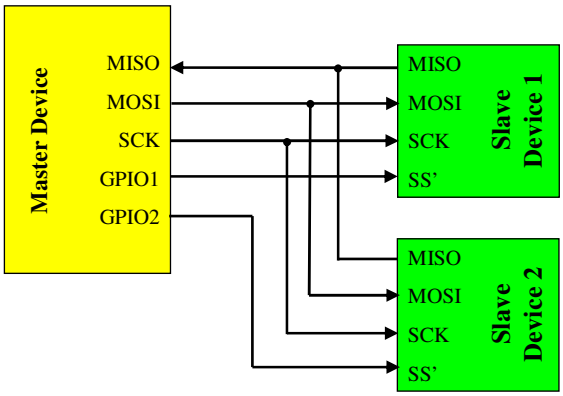
Standard SPI Signals

- Master in / Slave out (MISO)
- Master out / Slave in (MOSI)
- Serial Clock (SCK)
- Slave select (SS')

SPI Wiring modes

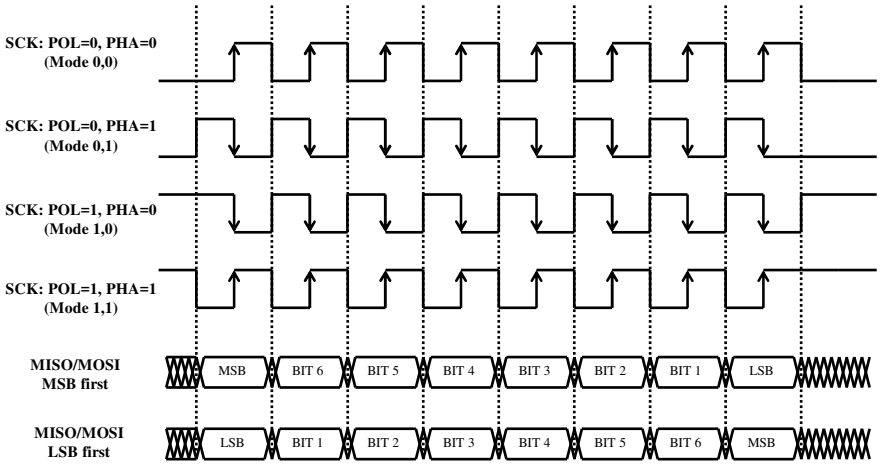
- Master: The device initiates and controls the flow of data. Can be configured as:
  - Multi-master.
  - Three wire single master.
  - Four (or more) wire master.
  - Four wire master with daisy-chained slaves.
- Slave: The device needs to be selected, and the clock is provided to it.

SPI Wiring

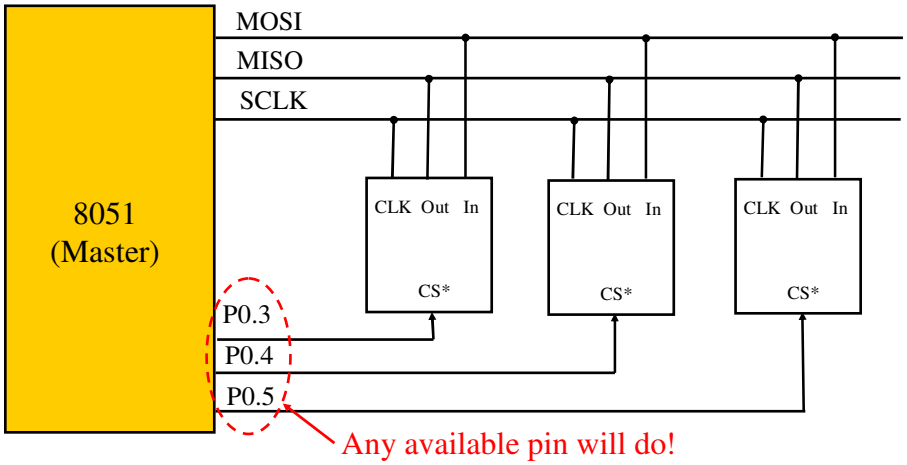


4-Wire Single Master and 4-Wire Multiple Slave Mode Connection Diagram. This is the most common configuration.

SPI Timing



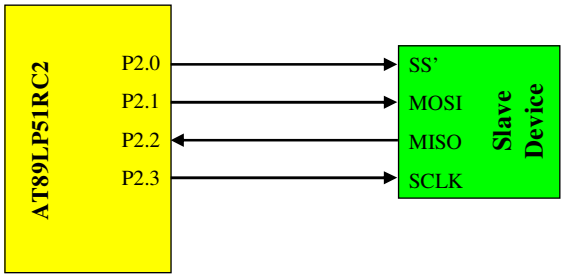
Connecting Devices to the SPI port



Bit Bang SPI

- Many modern microcontrollers have one or more built-in SPI controllers. This includes almost all Freescale (formerly Motorola), PICs from Microchip, 8051s derivatives from Silabs, Atmel, Dallas/Maxim, and NXP (formerly Philips), and ARMs from Atmel, NXP, and many others.
- Occasionally you may need to connect a SPI device to a microcontroller without an SPI controller. In this case we can **Bit Bang** SPI signals using GPIO pins!
- The AT89LP51RC2 has a built-in SPI controller but I used the SPI pins to In-System-Program (ISP) the µC, so… Bit Bang SPI it is!

Bit Bang SPI



GPIO pins "arbitrarily" chosen… (they are actually the closest pins to the MCP3008 as you'll see in the next slides)

Bit Bang SPI in Mode (0,0)

```
#define ADC_CE  P2_0
#define BB_MOSI P2_1
#define BB_MISO P2_2
#define BB_SCLK P2_3

unsigned char SPIWrite( unsigned char out_byte)
{
	// In the 8051 architecture both ACC and B are bit addressable!
	ACC=out_byte;

	BB_MOSI=ACC_7; BB_SCLK=1; B_7=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_6; BB_SCLK=1; B_6=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_5; BB_SCLK=1; B_5=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_4; BB_SCLK=1; B_4=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_3; BB_SCLK=1; B_3=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_2; BB_SCLK=1; B_2=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_1; BB_SCLK=1; B_1=BB_MISO; BB_SCLK=0;
	BB_MOSI=ACC_0; BB_SCLK=1; B_0=BB_MISO; BB_SCLK=0;

	return B;
}
```

Example: Using the MCP3008 10-bit, 8-channel ADC

Functional Block Diagram

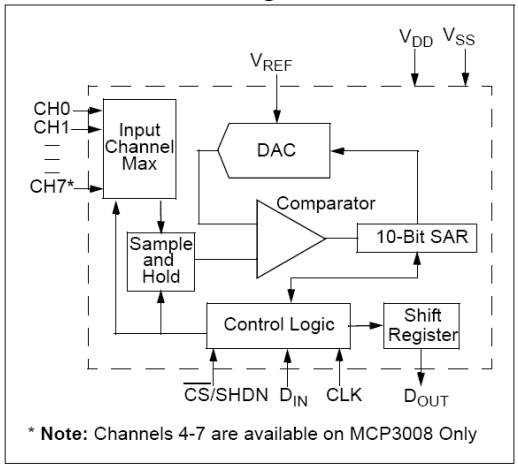
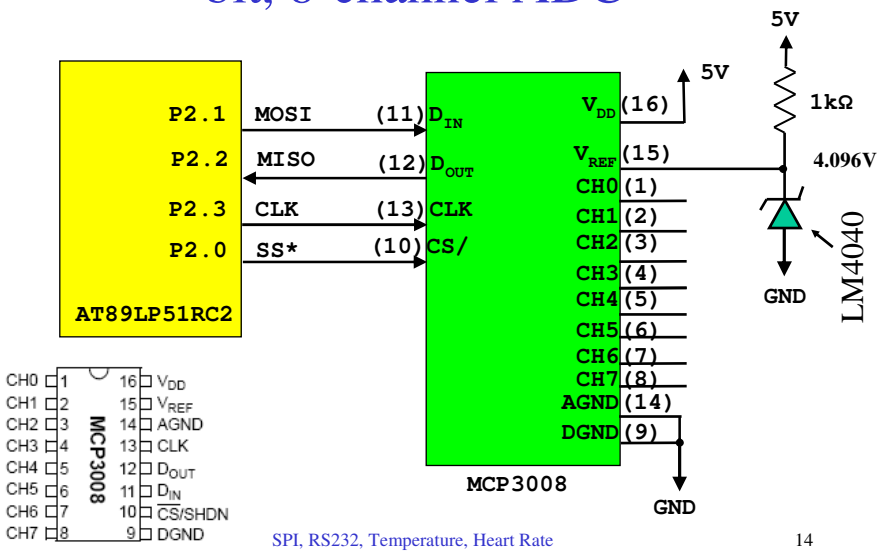


Figure taken From Microchip's MCP3008 datasheet

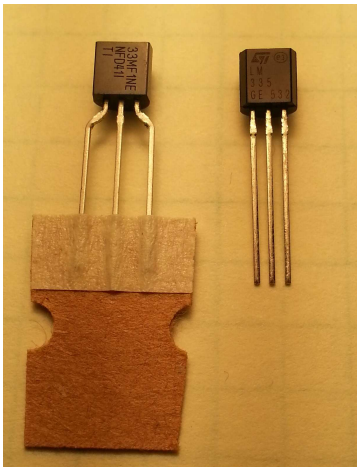
Example: Using the MCP3008 10-bit, 8-channel ADC



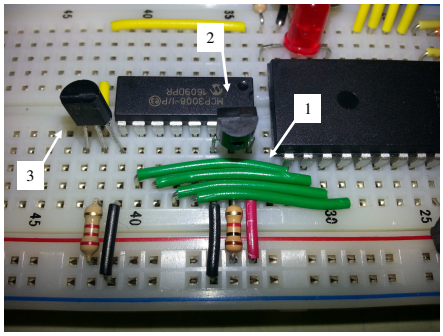
Voltage Reference LM4040DIZ-4.1

- Ideally we should use a good voltage reference for the "V_{REF}" pin of the ADC. In your kit you have the LM4040DIZ-4.1: 1%, 150ppm/°C, 4.096V voltage reference. There are two ways of using the voltage reference:
  - We keep V_{CC} connected to V_{REF}, but connect the 4.096V voltage to one of the inputs of the ADC. With that input we can easily determine the true value of V_{CC}. The range of the other inputs is then nominally 0V to V_{CC}.
  - We connect the 4.096V directly to the reference input of the ADC. The range of the inputs is then 0V to 4.096V. This method seems to be the most sensible option for ELEC291/292.
- WARNING: Some old markings in the reference IC say "33MF1NE NFD41I TI" and has a piece of cardboard taped to its pins. Also, its package looks similar to that of BJTs.

LM4040 (reference) and LM335 (temperature sensor)



Wiring the Circuit (one possible way)



1. P2.0 (pin 20) to P2.3 (pin 23) connections to ADC.
2. LM4040DIZ-4.1
3. LM335 (un-adjusted) connected to channel 0.

Example: Using the MCP3008 10-bit, 8-channel ADC

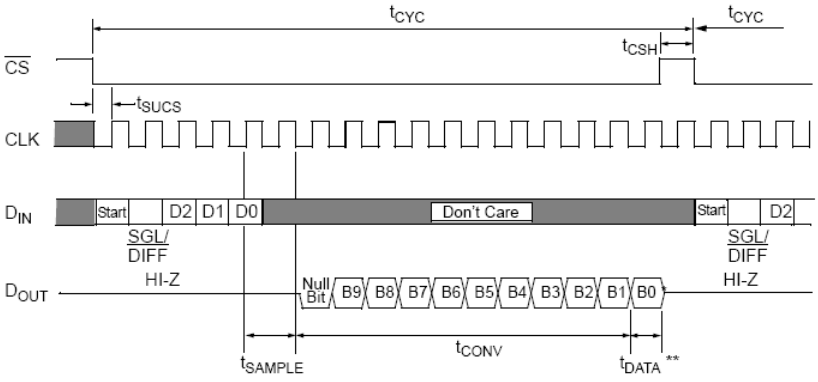


Figure taken from Microchip's MCP3008 datasheet

Example: Using the MCP3008 10-bit, 8-channel ADC

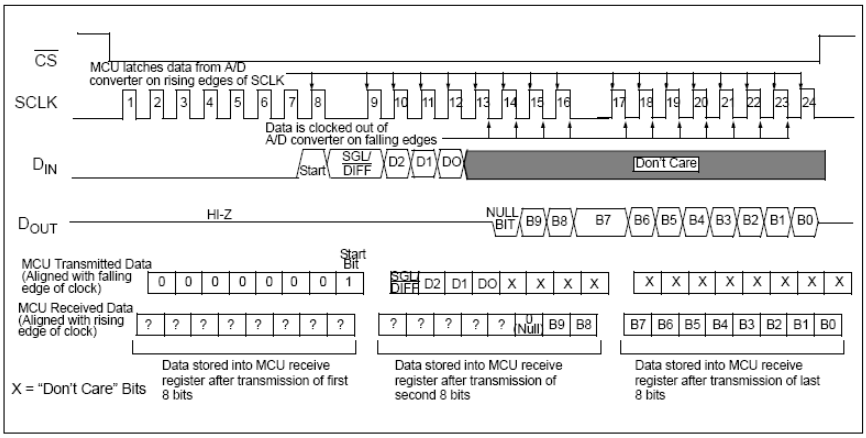


Figure taken from Microchip's MCP3008 datasheet

FIGURE 6-1: *SPI Communication with the MCP3004/3008 using 8-bit segments (Mode 0,0: SCLK idles low).*

Example: Using the MCP3008 10-bit, 8-channel ADC

```
/*Read 10 bits from the MCP3008 ADC converter*/
unsigned int volatile GetADC(unsigned char channel)
{
    unsigned int adc;
    unsigned char spid;

    ADC_CE=0; // Activate the MCP3008 ADC.

    SPIWrite(0x01); // Send the start bit.
    spid=SPIWrite((channel*0x10)|0x80); //Send single/diff* bit, D2, D1, and D0 bits.
    adc=((spid & 0x03)*0x100); //spid has the two most significant bits of the result.
    spid=SPIWrite(0x00); // It doesn't matter what we send now.
    adc+=spid; // spid contains the low part of the result.

    ADC_CE=1; // Deactivate the MCP3008 ADC.

    return adc;
}
```

The Serial Port

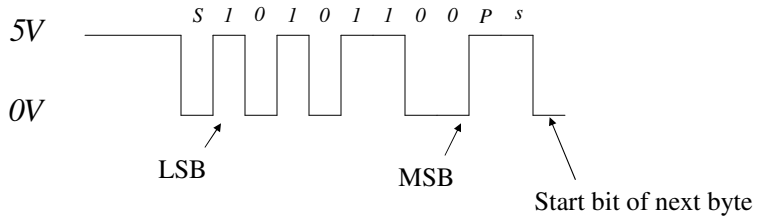
- The AT89LP51RC2 as well as most popular microcontrollers have one or more serial ports.
- The serial port uses the RS-232 communication standard. It was introduced in 1962!
- Perhaps the easiest way to communicate between a microcontroller and a computer!
- Unlike SPI, RS-232 is asynchronous: the clock is not shared between the processors.

Asynchronous Data Communication Data Format

- A start bit used to synchronize the data. '0' or space.
- 5 to 8 data bits. For the standard 8051 the number of data bits is usually 8.
- Optional parity bit. Set or reset so that the number of ones transmitted is either odd or even. For the standard 8051 the parity is set to 'none' by default.
- One, one and half, or two stop bits. Always '1' or mark. For the standard 8051 is set to one stop bit.

Asynchronous Data Communication Data Format

- For example, transmit “00110101” using 8 bits, odd parity, one stop bit:



Baud Rate

$$BR = \frac{1}{t_{bit}}$$ Unit is ‘baud’

- Standard baud rates are: 110, 300, 600, 1200, 4800, 9600, 14400, 19200, 38400, and so on…
- The 8051/8052 with the correct crystal (For example 22.1184 MHz) can generate all the standard baud rates up to 115200 baud!

"Magic Crystal" Frequencies

https://en.wikipedia.org/wiki/Crystal_oscillator_frequencies

Frequency (MHz)	comm	UART	A/V	RTC	Primary uses
0.032				✓	Real-time clocks, watches; allows binary division to 1 kHz signal (2^5×1 kHz).
0.032768				✓	Real-time clocks, quartz watches and clocks; allows binary division to 1 Hz signal (2^{15}×1 Hz); also low-speed low-power microcontrollers. Very common. Available as TCXO.[1]
					/180/150/110 kHz to both sides from the 10.7 MHz frequency)
11.0592		115200			UART clock (6×1.8432 MHz); allows integer division to common baud rates (96×115200 baud or 96×96×1,200 baud); common clock for Intel 8051 microprocessors[18]
					Used in CD-DA systems and CD-ROM drives; allows binary
22.1184		460800			UART clock allows integer division to common baud rates up to 460,800(x16x3) or 921600(x8x3). Twice the 11.0592 MHz frequency.

Serial port in the AT89LP51RC2

- "Hello, World!" example provided in Canvas.
- Download and install PuTTY

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

- During PuTTY installation, use a folder WITHOUT white spaces in the name.

Baud rate setup

- Using the baud rate generator (preferred in the AT89LP51RC2, as we can use the timers for something else):

$$BRL = 256 - \frac{f_{osc}}{16 \times baud}$$

$$BRL = 256 - \frac{22.1184MHz}{16 \times 115200} = 244$$

"Hello, World!" example

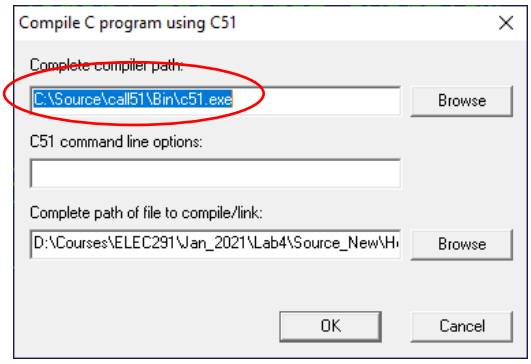
```
1  //  AT89LP51RC2 "Hello, World!" example.
2  //  ~C51~
3
4  #include <stdio.h>
5  #include <at89lp51rd2.h>
6
7  #define CLK    22118400L // SYSCLK frequency in Hz
8  #define BAUD     115200L // Baud rate of UART in bps
9  #if ((CLK/(16L*BAUD))>0x100L)
10 #error "Can not set baud rate because (CLK/(16*BAUD)) > 0x100 "
11 #endif
12 #define BRG_VAL (0x100-(CLK/(16L*BAUD)))
13
14 unsigned char _c51_external_startup(void)
15 {
16     AUXR=0B_0001_0001; // 1152 bytes of internal XDATA, P4.4 is I/O
17
18     // Configure serial port and baud rate
19     PCON|=0x80;
20     SCON = 0x52;
21     BDRCON=0;
22     BRL=BRG_VAL;
23     BDRCON=BRR|TBCK|RBCK|SPD;
24
25     return 0;
26 }
27
28 void main (void)
29 {
30     printf("Hello, World!\n");
31 }
32
```

Serial port in C for the 8051

- putchar() and getchar() already defined in the standard libraries.
- printf() uses putchar().
- scanf() uses getchar(). (There is a bug in scanf() that prevents reading to a 'unsigned char' or 'char' variable properly. I don't know how to fix it even though I wrote the function! Lines 369 and 384 of scan_format.c. The work around is to read to an integer variable.)
- Floating point arithmetic supported. Double not supported!
- Don't call scanf() or printf() from inside an interrupt service routine. EVER. Not even when you are using a fancy ARM or MIPS processor.

Compiling with C

C compiler



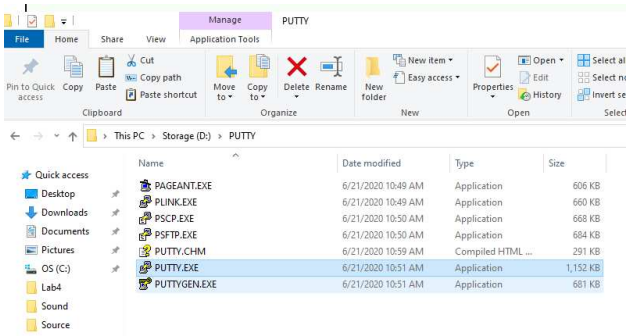
Load as Usual

PuTTY

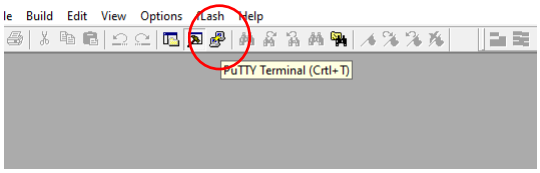
- Download from here:

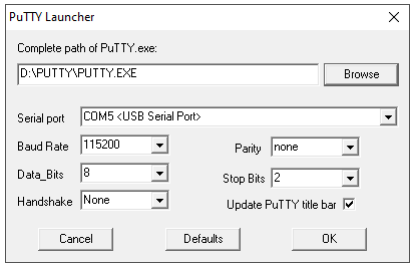
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

- Install in a folder **without spaces** in the name.



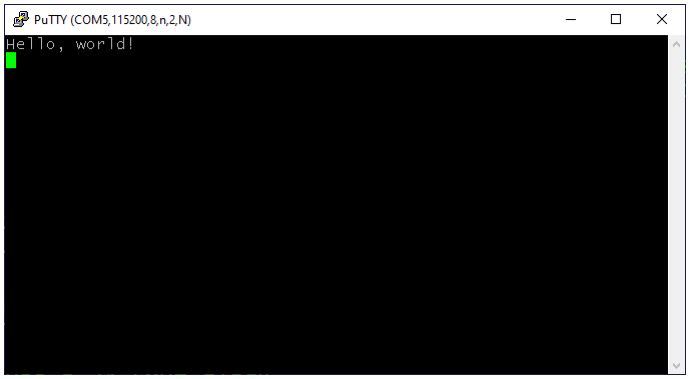
Calling PuTTY from CrossIDE





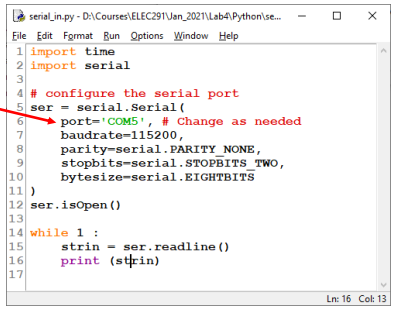
Running PuTTY

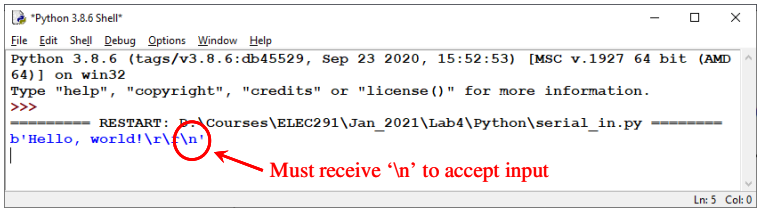
Press the RESET push button on the AT89LP51RC2:



Using the Serial Port with Python

Change to Match your computer





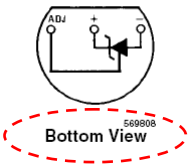
Using the Serial Port with Python

```
import time
import serial

# configure the serial port
ser = serial.Serial(
    port='COM5', # Change as needed
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)
ser.isOpen()

while 1 :
    strin = ser.readline()
    print (strin.decode('utf-8'))
```

Different

```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\Courses\ELEC291\Jan_2021\Lab4\Python\serial_in.py ==========
Hello, world!
```


ELEC291: LM335 Temperature Sensor



For the LM335:

+10mV/^{o}K, -40^{o}C<t<100^{o}C

For the MCP3008 :

ADC: 10-bit, 0.0V<V_{in}<4.096V

Un-calibrated temperature error: 2 to 6°C

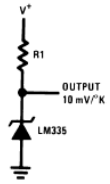
$$-40^0\,C = (273-40)^0\,K = 233^0\,K \rightarrow 2.33V$$

$$+100^0\,C = (273+100)^0\,K = 373^0\,K \rightarrow 3.73V$$

From the datasheet: “Included on the LM335 chip is an easy method of calibrating the device for higher accuracies. A pot connected across the LM335 with the arm tied to the adjustment terminal allows a 1-point calibration of the sensor that corrects for inaccuracy over the full temperature range.” NO NEED TO DO THIS: leave the ADJ pin unconnected.

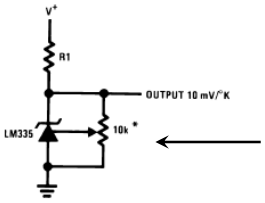
ELEC291: LM335 Temperature Sensor

Figure 15. Basic Temperature Sensor



R1=2kΩ or 2.2kΩ, if you check the datasheet, most of the specs are @ 1 ma.

Figure 16. Calibrated Sensor

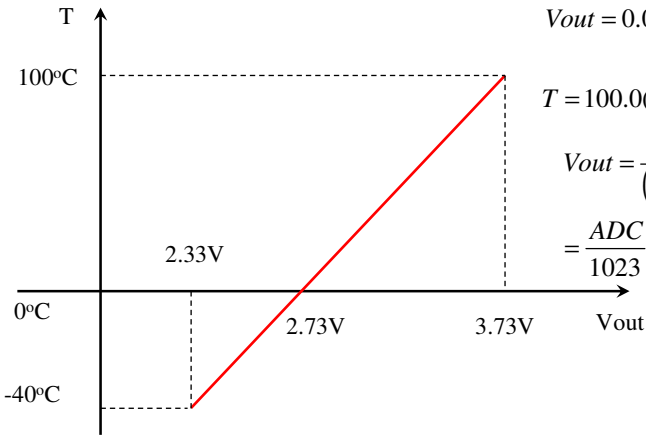


*Calibrate for 2.982V at 25°C

V^+=5V

NO NEED TO DO THIS.

ELEC291: LM335 Transfer Function



$$Vout = 0.01\frac{V}{^oC}\times T + 2.73V$$

$$T = 100.0(Vout - 2.73V)\frac{^oC}{V}$$

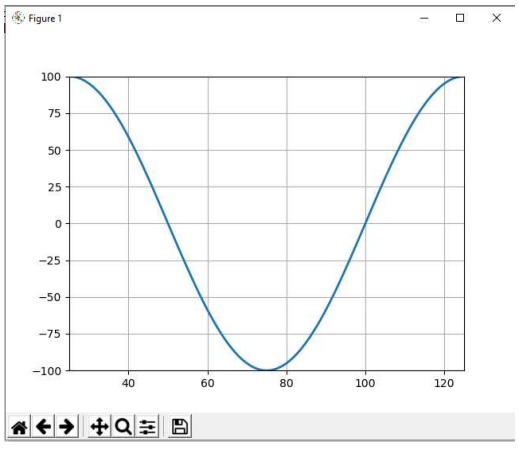
$$Vout = \frac{ADC}{(2^{10}-1)}\times V_{REF}$$

$$= \frac{ADC}{1023}\times 4.096V$$

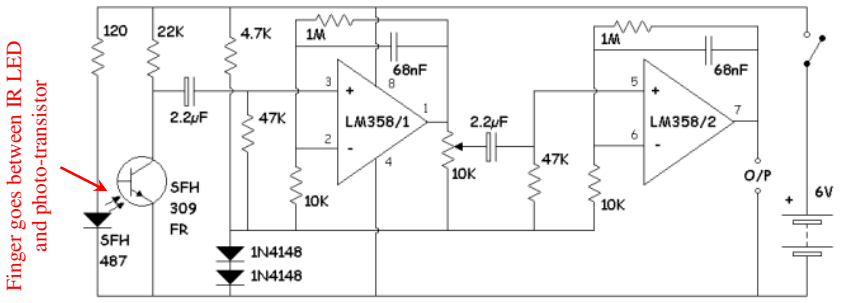
Examples Provided

- For the AT89LP51RC2:
  - Hello_AT89LP51RC2.c
  - ADC_SPI.c
  - LCD_4bit_AT89LP51RC2.c
- Python:
  - stripchart_sinewave.py
  - serial_in.py

Output of stripchar_sinewave.py

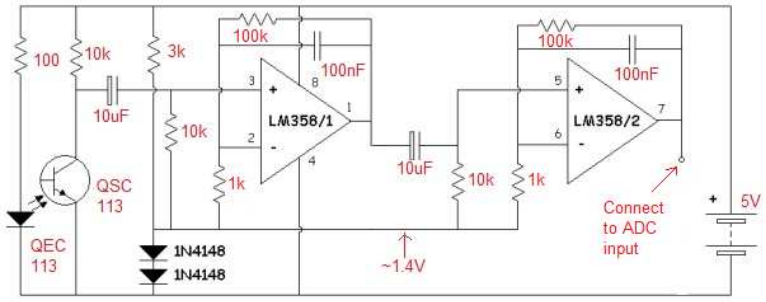


ELEC292: Heart Rate Sensor



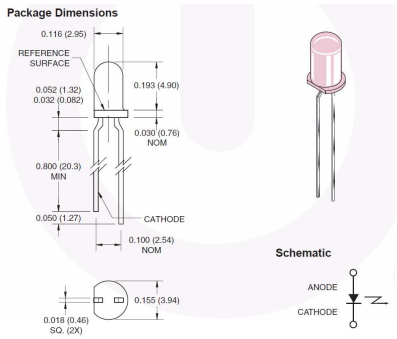
http://www.picotech.com/experiments/calculating_heart_rate/

ELEC292: Modified Circuit

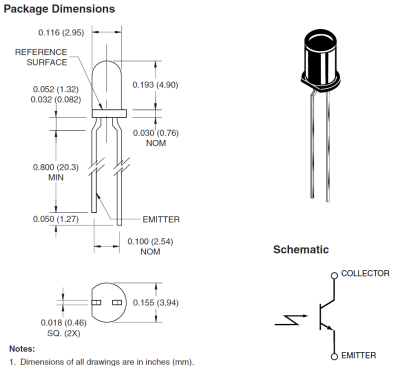


Using only components in your parts kit.
Potentiometer removed. 5V power supply.

IR LED and IR Photo-Transistor

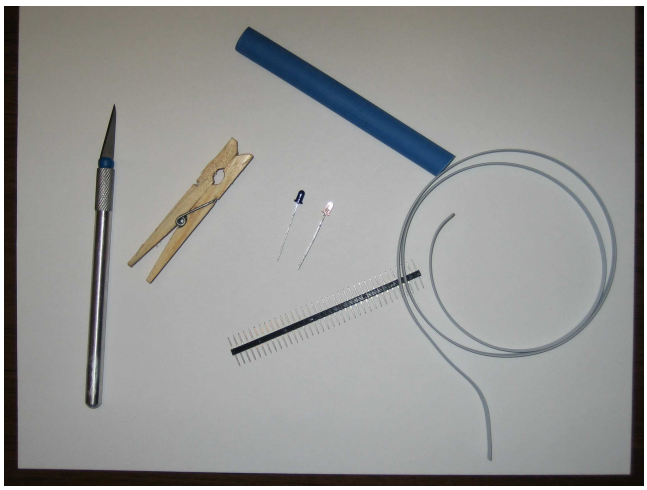


QEC113

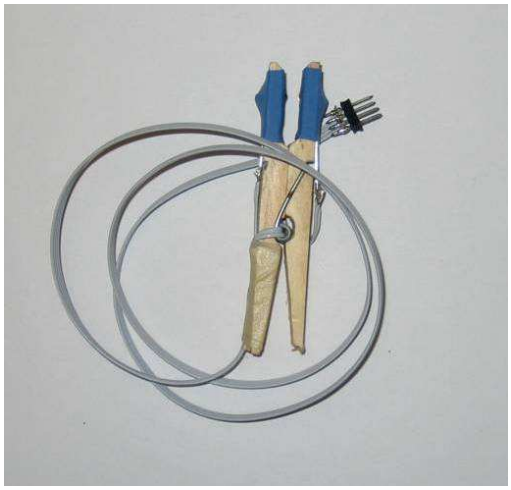


QSC113

Finger Clip: before



Finger Clip: after



Final Remarks

- Support macOS:
  - WINE. Instructions on Canvas. Works both for ARM and Intel.
  - Virtual Machine (parallels or Oracle VB)
  - Bootcamp.
  - Compile remotely using EECE virtual machines and copy hex file to local machine.
- For Linux
  - Virtual Machine (Oracle VB)
  - WINE.