Title: Testing the temperature

Goal:

* Read LM34 temperature sensor with Atmega328p ADC port
* Transfer data to ESP8266 module through UART
* Push reading onto Thingspeak.com account

Deliverables:

The main deliverable of this assignment is a graph plotting current room temperature.

Literature Survey:

Having the ability to plot the current temperature over time from a sensor can help in maintaining a desired temperature. By integrating this device with a homes heating system it could accomplish that goal.

Components:

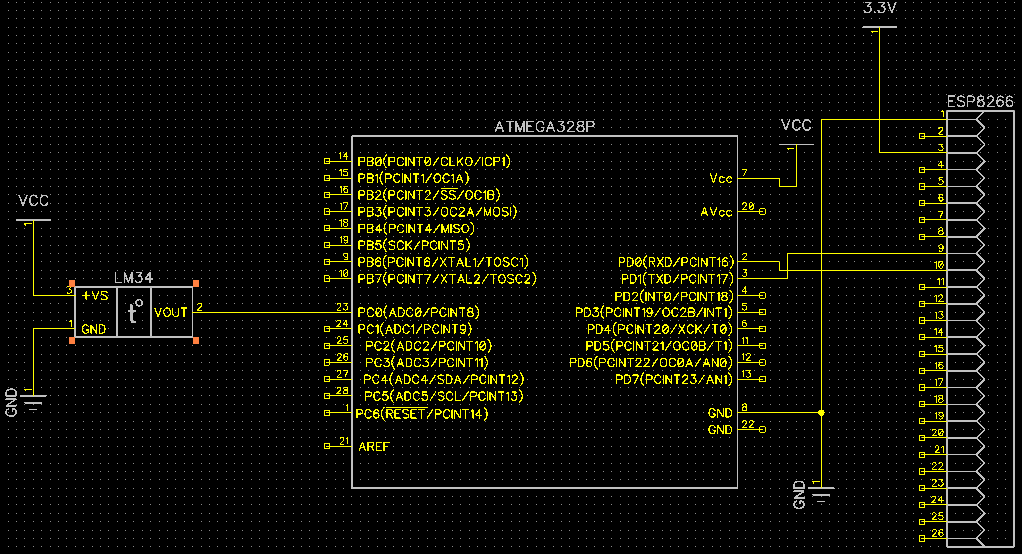
Atmega328p –Microcontroller (SPI, UART interface)

LM34 - Temperature Sensor

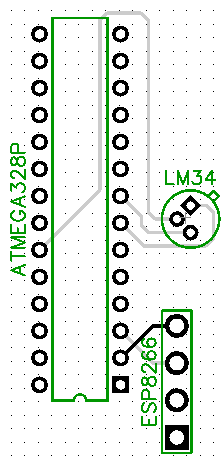
ESP8266 – WiFi module (3.3v)

USB FTDI – 3.3 V adapter

USB cable – 5 V power

Schematics: 

PCB:



Implementation:

* ATMega328P Temperature sensor via ADC port
* ATMega328P connected through UART to ESP8266
* Every 15 seconds ESP8266 pushes temperature data to web server (Thingspeak.com)

Snapshots/Screenshots:

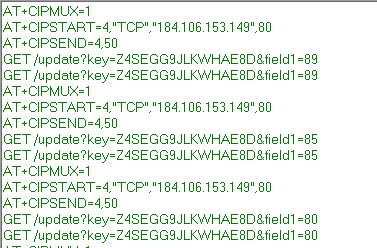


Figure 1-Screenshot of terminal reading transmitted strings as they are received by ESP8266

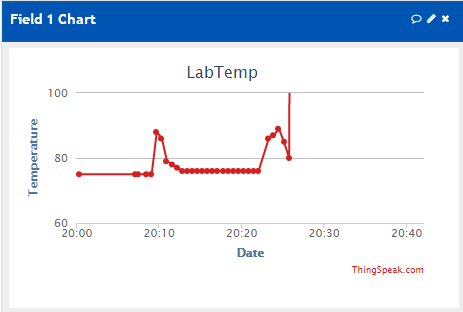


Figure 2-Screenshot of temperature being graphed. Shows spike when intentionally being warmed up and then returns back to room temperature

CODE: (with comments)

/\*

\*

\* Final.c

\* Author: battled

\*Code to read from LM-34 sensor, covert the temp and transmit it to ESP8266

\*/

#define MOSI 4 //mosi is pb3

#define SCK 6 //SCK is pb5

#define SS 3 //SS is pb2

#define THING\_SPEAK\_IP\_STR "184.106.153.149" /\* thingspeak.com IP Address \*/

#define THINK\_SPEAK\_IP\_PORT 80 /\* port number \*/

#define THING\_SPEAK\_KEY\_STR "Z4SEGG9JLKWHAE8D" /\*My API key \*/

#define THING\_SPEAK\_CHANNEL 20696 /\* channel ID \*/

#define THING\_SPEAK\_LABEL\_STR "field1"

#define F\_CPU 8000000UL //8 MHz xtal crystal

#include <avr/io.h> //

#include <avr/interrupt.h>

#include <string.h> //for string concatenation

#include <stdio.h> //for sprintf

#include <util/delay.h>

//

void uart\_init (void)

{

UCSR0B = (1<<TXEN0)|(1<<RXEN0);

UCSR0C = ((1<<UCSZ01)|(1<<UCSZ00));

UBRR0L = 8;

}

void uart\_send(char \*ch)

{

while(\*ch != '\0')

{

while(! (UCSR0A & (1<<UDRE0)));

UDR0 = \*ch;

ch++;

}

}

void uart\_prepSend(void)

{

\_delay\_ms(9000);

uart\_send("AT+CIPMUX=1\r\n");

\_delay\_ms(9000);

uart\_send("AT+CIPSTART=4,\"TCP\",\"184.106.153.149\",80\r\n");

\_delay\_ms(9000);

uart\_send("AT+CIPSEND=4,50\r\n");

\_delay\_ms(9000);

}

int main(void)

{

DDRC &= ~(1<<PORTC0); //PC0 is analog input

uart\_init(); //initialize USART

//uart\_prepSend(); //prepare string to be sent

ADCSRA = ((1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0));// ADC prescaler 128 ADEN

ADMUX = ((1<<REFS1)|(1 << REFS0)); // select internal 1.1 V Ref w/ ext cap at AREF pin and ADC0 (default)

sei(); //enable interrupts

//configure timer 1 to interrupt every second

TCNT1 = 65536 - ((double)F\_CPU/256); // set timer to overflow in 1 sec

TCCR1A = 0; //normal mode

TCCR1B = 4; //prescaler = 256

TIMSK1 |= (1<<TOIE1); //enable interrupt on overflow timer 1

while(1); //wait for interrupts

return 0;

}

ISR(TIMER1\_OVF\_vect) //timer1 overflow ISR

{

TCCR1B = 0; //stop timer 1

TIFR1 = 1; //clear overflow flag

int adc\_temp; //stores ADC

float adc\_tempf; //float for calculations

int adc\_tempi; //integer part

char TmpTemp[44];

char daTemp[5];

//read ADC

ADCSRA |= (1<<ADSC); //start conversion

while((ADCSRA &(1<<ADIF)) == 0); //wait for conversion to finish

adc\_temp = ADC; //save ADC value

adc\_tempf = (float)adc\_temp \* (1.1 / 1024) / 0.01; //(ADC\*res/.01) (lm34 sf = 10mv/degF)

adc\_tempi = (int)adc\_tempf; //integer part

uart\_prepSend();//prepare to upload the converted string

sprintf(TmpTemp,"GET /update?key=Z4SEGG9JLKWHAE8D&field1="); //store upload value command

sprintf(daTemp,"%d\r\n",adc\_tempi);

strcat(TmpTemp, daTemp); //concatonate read in value to upload value command

uart\_send(TmpTemp);

\_delay\_ms(9000);

uart\_send(TmpTemp); //command is required to be sent twice by AT standers

\_delay\_ms(9000);

//output degrees Fahrenheit to cloud

\_delay\_ms(19000);

//reset time 1 for interrupt

TCNT1 = 65536 - ((double)F\_CPU/256); //overflow in 1 sec

TCCR1A = 0; //normal mode

TCCR1B = 4; //prescaler = 256

return;

}

Reference:

<http://nodemcu.readthedocs.io/en/dev/en/upload/>

<https://github.com/nodemcu/nodemcu-firmware>

<https://www.mathworks.com/help/thingspeak/examples.html>

<https://sites.google.com/a/unlv.edu/unlvcpe301/projects>