PROJECT REPORT

ON

MICROCONTROLLER BASED TEMPERATURE SENSOR

PREPARED BY

VRUNDA A. DAVE (CE-28)

&

MANTHAN H. DOSHI (CE-34)

CLASS: COMPUTER ENGINEERING (5th SEM)

GUIDED BY

ASST PROF. PARTH DAVE

ASST PROF. JAYDEEP TRIVEDI

ASST PROF. NIKUNJ VADHER

SUBMITTED TO

FACULTY OF TECHNOLOGY

DHARMSINH DESAI UNIVERSITY

NADIAD – 387001

2012-2013

INDEX

DHARAMSINH DESAI UNIVERSITY

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that Mr. Manthan Doshi (CE-34) and Ms. Vrunda Dave (CE-28) of B.TECH C.E. Semester V of DHARAMSINH DESAI UNIVERSITY, NADIAD have successfully completed their term project in subject MICROPROCESSOR FUNDAMENTALS AND PROGRAMMING during the academic year 2012-2013. During the whole term they were found sincere, hardworking and punctual. They made MICROCONTROLLER BASED TEMPERATURE SENSOR.

Staff in Charge

Date :

Head Of Department

Date :

PREFACE

Gaining the theoretical knowledge and conceptualizing it, into practical utilities and day-to-day life, must be the correct approach to any subject. With the above ideology, in order to understand Microprocessors' Fundamentals more precisely, this project has been chosen. It not only aims to clear microprocessor fundamentals, but also clears basics of microcontroller.

The sentence "Experience is the best teacher" is very true in every field. The project provides the experience that takes very important part in our life. So Project is arranged in third year of engineering to develop the skill.

ACKNOWLEDGEMENT

This report is an outcome of the project we have done in our Microprocessor Fundamentals and Programming subject.

We are very thankful to our respected professors, Prof. Parth Dave and Prof. Jaydeep Trivedi for allowing us to undertake this project.

We extend our sincere gratitude to Prof. Nikunj Vadher for providing necessary guidance and information during project work.

We are also thankful to all those persons who have directly or indirectly helped us in completing our project work.

Vrunda Dave

Manthan Doshi

ABSTRACT

In this era of digital signals, data processing and manipulating has become a very basic requirement in almost all real world tasks like mobile communication, instrument control in industries, aerospace operations, power grids, marine projects and so on.

Our Project intends to give an insight of real time working of above mentioned fields. Project serves as a very fundamental step towards achieving remote access to instruments sensitive to temperature. Objective is to display the current room temperature using the LCD Display. Based on this temperature , further processing can be done.

Components involved are:

Temperature sensor

Microcontroller

LCD

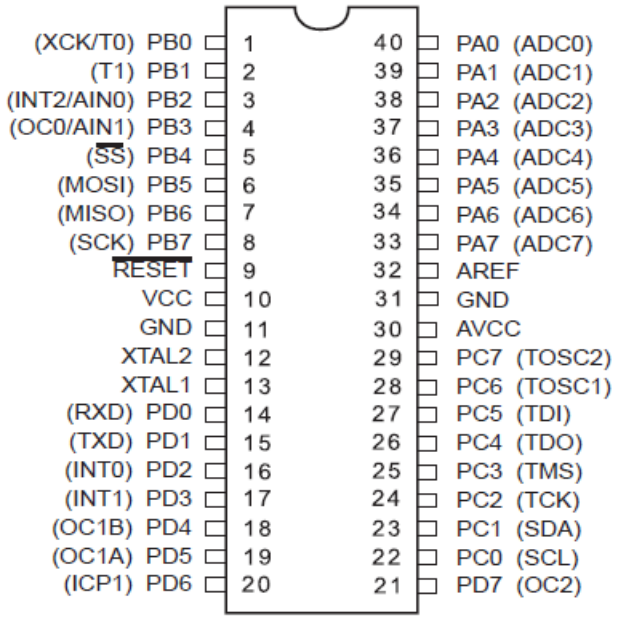
ADC

HARDWARE UTILITIES

Atmega32 Microcontroller

Overview: The ATmega32 is a low-power CMOS *8-bit microcontroller* based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Pin Diagram:



The ATmega32 provides the following features: 32K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024 bytes EEPROM, 2K byte SRAM, 32 general purpose I/O lines, 32 general purpose working registers,, On-chip Debugging support and programming, three flexible

Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, **an 8-channel**, **10-bit ADC** with optional differential input stage with programmable gain (TQFP package only), an SPI serial port, and six software selectable power saving modes.

The Onchip ISP Flash allows the program memory to be *reprogrammed* in-system through an *SPI serial interface*, by a conventional non volatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation.

Pin Descriptions

- **VCC:** Digital supply voltage.
- **GND:** Ground.
- **Port A (PA7..PA0):** Port A serves as the analog inputs to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port.
- **Port B (PB7..PB0):** Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).
- **Port C (PC7..PC0):** Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).
- **Port D (PD7..PD0):** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).
- **RESET:** Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset.
- **XTAL1:** Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.
- **XTAL2:** Output from the inverting Oscillator amplifier.
- **AVCC:** AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.
- **AREF:** AREF is the analog reference pin for the A/D Converter.

Temperature Sensor LM35

By interfacing different types of sensors with our MCU we can sense the environment and take decisions, in this way we can create "smart" applications. There are wide variety of sensors available. But here we have used a popular sensor LM35 which is precision *centigrade temperature sensor*. It can be used to measure temperature with *accuracy of 0.5 degree centigrade*. We can interface it easily with AVR MCUs and can create thermometers, temperature indicator.

LM35 by National Semiconductor is a popular and low cost temperature sensor. It is also easily available.

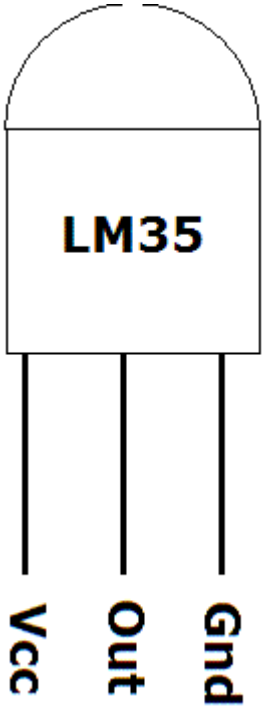


Fig - LM35 Pin Configuration

The Vcc can be from 4V to 20V as specified by the datasheet. To use the sensor simply connect the Vcc to 5V, GND to Gnd and the Out to one of the ADC (analog to digital converter channel). The output linearly varies with temperature. The output is **10MilliVolts per degree centigrade.**

So if the output is 310 mV then temperature is 31° C.

Analog to Digital Convertor(ADC)

The ATmega32 features a **10-bit** successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3 V from VCC.

Operation:

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.
If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

ADC Conversion Result:

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).
For single ended conversion, the result is:

$$ADC = \frac{V_{IN} * 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference.

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – REFS1:0: Reference Selection Bits

These bits select the voltage reference for the ADC, as shown in Table. The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Bit 5 – ADLAR: ADC Left Adjust Result

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted.

Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels.

ADC Control and Status Register – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – ADEN:ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off.

Bit 6 – ADSC: ADC Start Conversion

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Bit 5 – ADATE: ADC Auto Trigger Enable

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR.

Bit 4 – ADIF: ADC Interrupt Flag

This bit is set when an ADC conversion completes and the Data Registers are updated. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

Bit 3 – ADIE: ADC Interrupt Enable

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

The resolution of AVRs ADC is 10-bit and for reference voltage we are using 5V so the resolution in terms of voltage is 5/1024 = 5mV approx.

So if ADC reading is 10 it means : 10 x 5mV = 50mV.

You can get read the value of any ADC channel using the function : ReadADC(ch); Where ch is channel number (0-7).

If you have connected the LM35's output to ADC channel 0 then call adc_value = ReadADC(0) this will store the current ADC reading in variable

adc_value. The data type of adc_value should be int as ADC value can range from 0-1023.

As we saw ADC results are in factor of 5mV and for 1 degree C the output of LM35 is 10mV, So **2 units of ADC = 1 degree.**

So to get the temperature we divide the adc_value by two
temperature = adc_value/2;

Finally you can display this value in LCD Module.

16x2 LCD Display

LCD Modules can present textual information to user. It's like a cheap "monitor" that you can hook in all of your gadgets. They come in various types. The most popular one can display 2 lines of 16 characters. These can be easily interfaced to MCU's, thanks to the API(Functions used to easily access the modules) we provide.

The LCD module can be easily connected to the AVR MCU ATmega32. The diagram below shows the LCD 16×2.

BreadBoard and Wires

Connecting all above components require some base. This purpose is served by breadboard.

For connecting various voltage and i/o pins wires are used.

SOFTWARE UTILITIES

AVR Studio:

AVR Studio® 4 is a free Integrated Development Environment (IDE) for writing and debugging AVR applications in Windows 98/XP/ME/2000/7 and Windows NT environments.

AVR Studio 4 provides a project management tool, source file editor, simulator and In-Circuit Emulator interface for the powerful AVR 8-bit RISC family of microcontrollers.

It features Integrated AVR GCC compiler support, advanced data and program breakpoints, RTOS plug-in and window docking system.

Features:

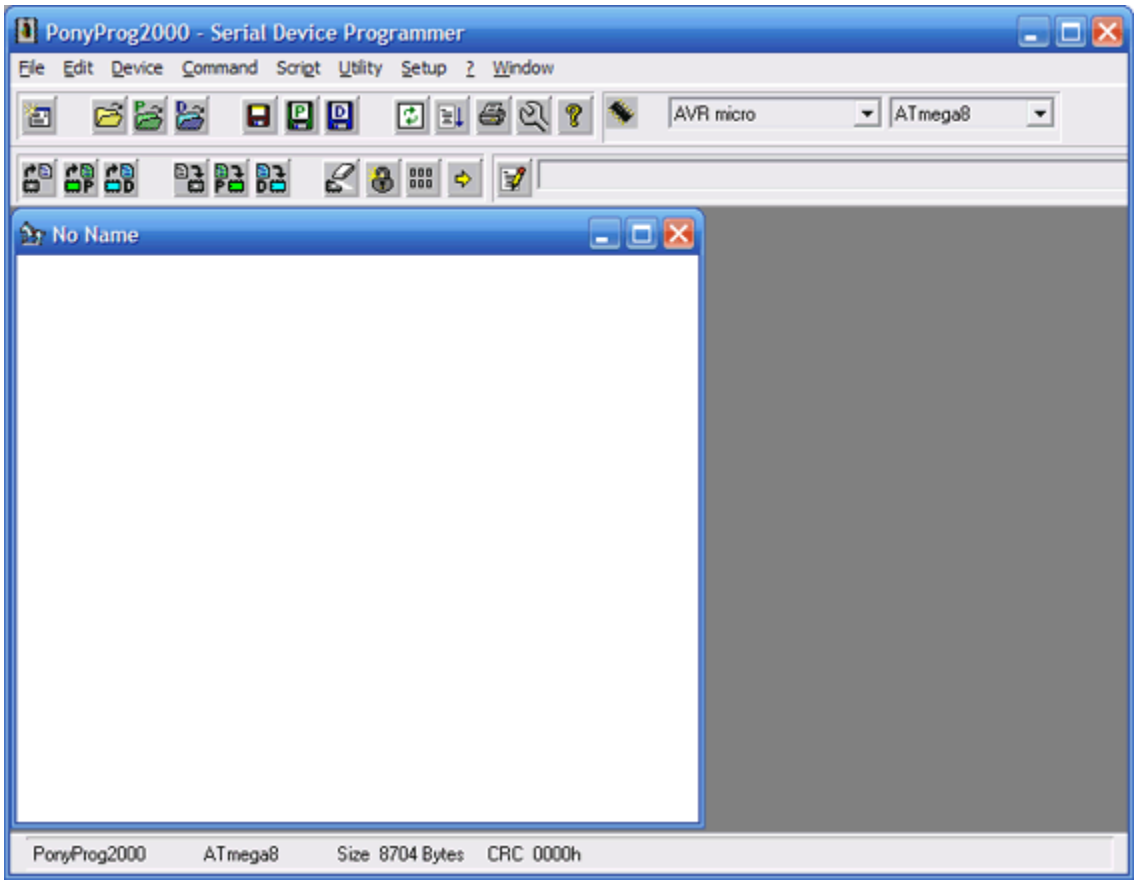
- Integrated Development Environment
- Write, Compile and Debug
- Fully Symbolic Source-level Debugger
- Configurable Memory Views (SRAM/EEPROM/Flash/Registers and I/O)
- Unlimited Number of Break Points
- Trace Buffer and Trigger Control
- Online HTML Help
- Variable Watch/Edit Window with Drag-and-Drop Function
- Extensive Program Flow Control Options
- Simulator Port Activity Logging and Pin Input Stimuli
- File Parser Support: COFF/UBROF6/UBROF8 and Hex Files
- Language support: C, Pascal, BASIC, and Assembly

Pony Prog:

A powerful but simple *serial device programmer software* with a user friendly GUI framework. Its purpose is reading and writing every serial device. At the moment it supports I^2C Bus, Microwire, SPI eeprom, the Atmel AVR and Microchip Micro.

It is the best way to perform AVR ISP programming. It works even in low voltage systems (3V). To select it choose "AVR ISP I/O" or "AVR ISP API" from the Options - Setup menu and the parallel checkbox.

You can connect directly to the target system (ISP) through the 10 pin connector; alternatively you can connect a PonyProg adapter card for AVR and feed external power to the device. On Windows2000/XP you have to select "AVR ISP I/O" and use a standard PC LPT port. Snapshot of Ponyprog screen is shown in the figure.



Implementation

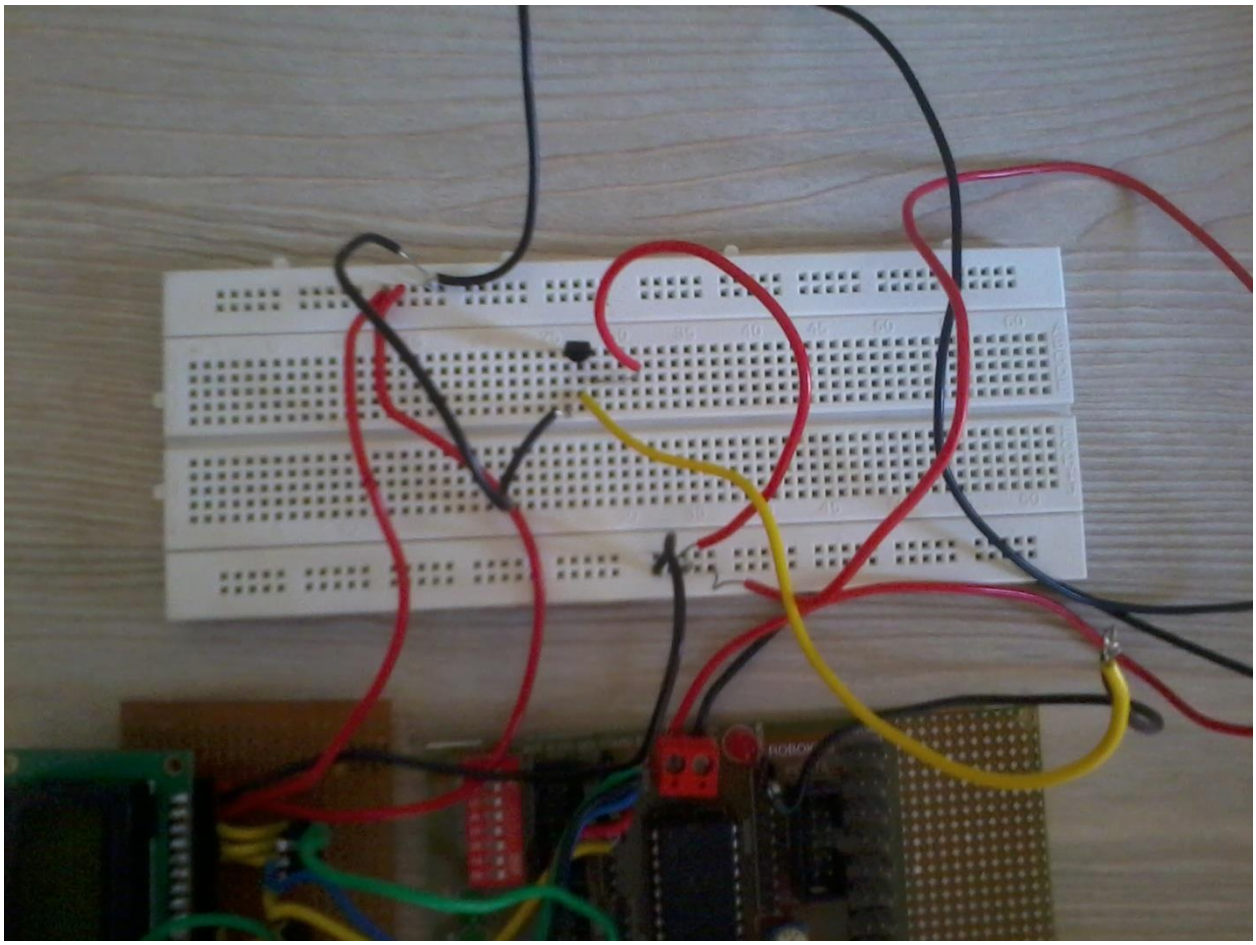
Component Connections

Atmega32 is available with all its components on a single PCB. It looks as follows:

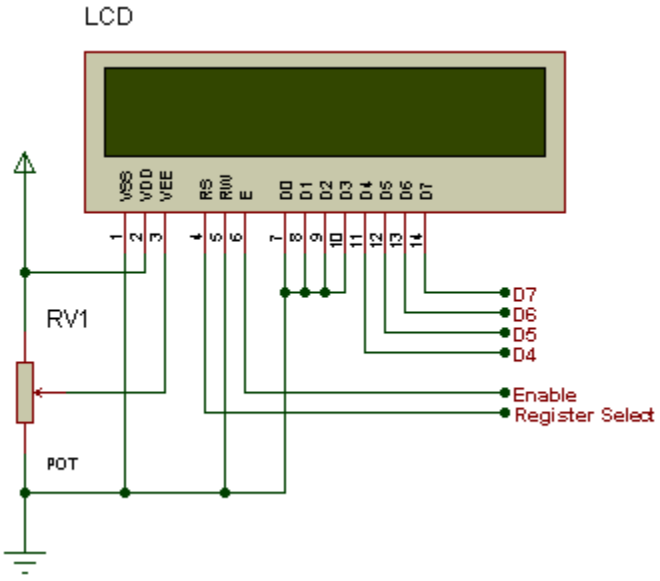
Different ports are available but we'll be using only ports B and D of above SDK.

1. Connect LM35 Sensor onto breadboard and supply 5v to its input pin using output 5v of SDK.
2. Wire output pin of LM35 to port A pin A0, as we will use channel 0.
3. Connect a ground of LM35 with that available from SDK onto breadboard as follows:

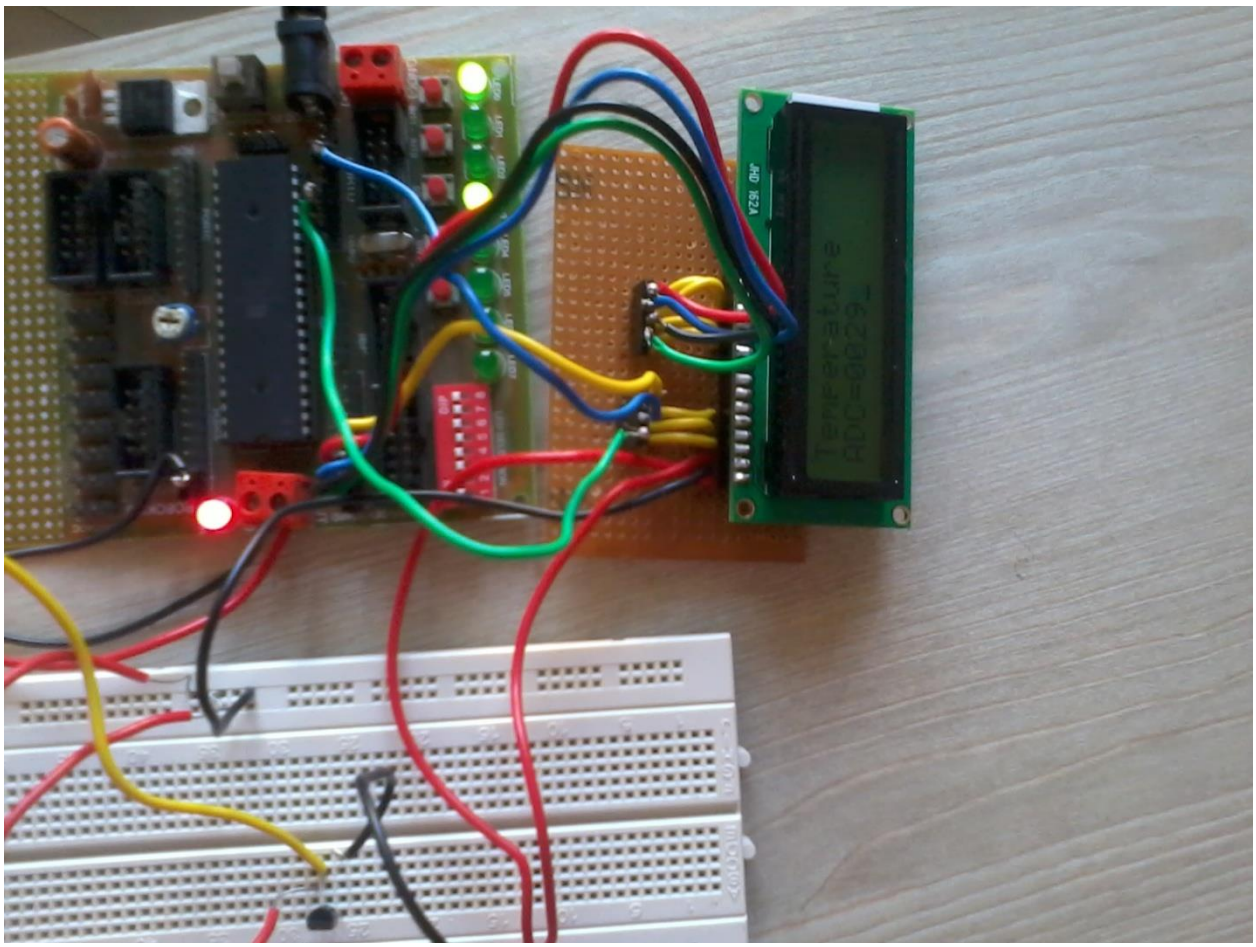
   As shown below yellow wire is output from LM35 which is in analog form. Red wire connected with LM35 is Vcc supply to it and black wire is ground. These are connected with that from SDK



4. Now to connect LCD we need to understand its pin diagram as follows:



5. Vss and Vee pins needs to get 0v and Vdd pin 5v. so connect these pins onto breadboard with those already available from SDK. These connections are visible from previous snap of breadboard.
6. Connect pins 4,5,6 with PD3,PD6 and PB4 port pins of SDK respectively. These 3 pins are used to configure LCD for functions like: selecting line, selecting block in line, to enable cursor blink ,etc. These all are handled by header file for LCD module.
7. To send data onto LCD pins numbered 11,12,13,14 are used. These are connected with PB0,PB1,PB2,PB3 pins of SDK as port B will be containing output temperature. Following snap shows this:



Three wires green, blue and yellow are pins 4,5,6 of LCD which are connected as mentioned above.
Also four wires together, namely, green, black, blue, red are pins 11,12,13,14 of LCD respectively.

NOTE: *The brown board in previous snap is PCB on which LCD module is connected using male-female connector pins using soldering.*

8. All the components are now connected and ready to be used.
9. Supply external voltage to SDK using its adapter. However, you'll not get any output on LCD..!!!!

This is because microcontroller hasn't been loaded with logic code. Lets see how MCU is programmed.

Interfacing LM35

Basically no programming needs to be done to LM35. But output of LM35 is taken at pin PA0. So to read voltage level from PA0 following code needs to be developed:

```
uint16_t ReadADC(uint8_t ch)
{
   //Select ADC Channel ch must be 0-7
   ch=ch&0b00000111;
   ADMUX|=ch;

   //Start Single conversion
   ADCSRA|=(1<<ADSC);

   //Wait for conversion to complete
   while(!(ADCSRA & (1<<ADIF)));

   //Clear ADIF by writing one to it
   //Note you may be wondering why we have write one to clear it
   //This is standard way of clearing bits in io as said in datasheets.
   //The code writes '1' but it result in setting bit to '0' !!!

   ADCSRA|=(1<<ADIF);

   return(ADC);
}
```

To initialize ADC to start receiving output from LM35, following code is needed:

```
void InitADC()
{
ADMUX=(1<<REFS0);                              // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor
=128
}
```

Interfacing 16x2 LCD Display

Once voltage value has been manipulated into temperature, it needs to be displayed on LCD. To do so we first need to initialize LCD cursor as follows:

```
//Initialize LCD
   LCDInit(LS_BLINK|LS_ULINE);
   LCDClear();
```

These are methods available in a header file named lcd.h
Once LCD is ready to write onto LCD following method can be used:

```
LCDWriteStringXY(0,1,"temp=");
LCDWriteIntXY(4,1,adc_result,4); //Print the value in 4th column second line
```

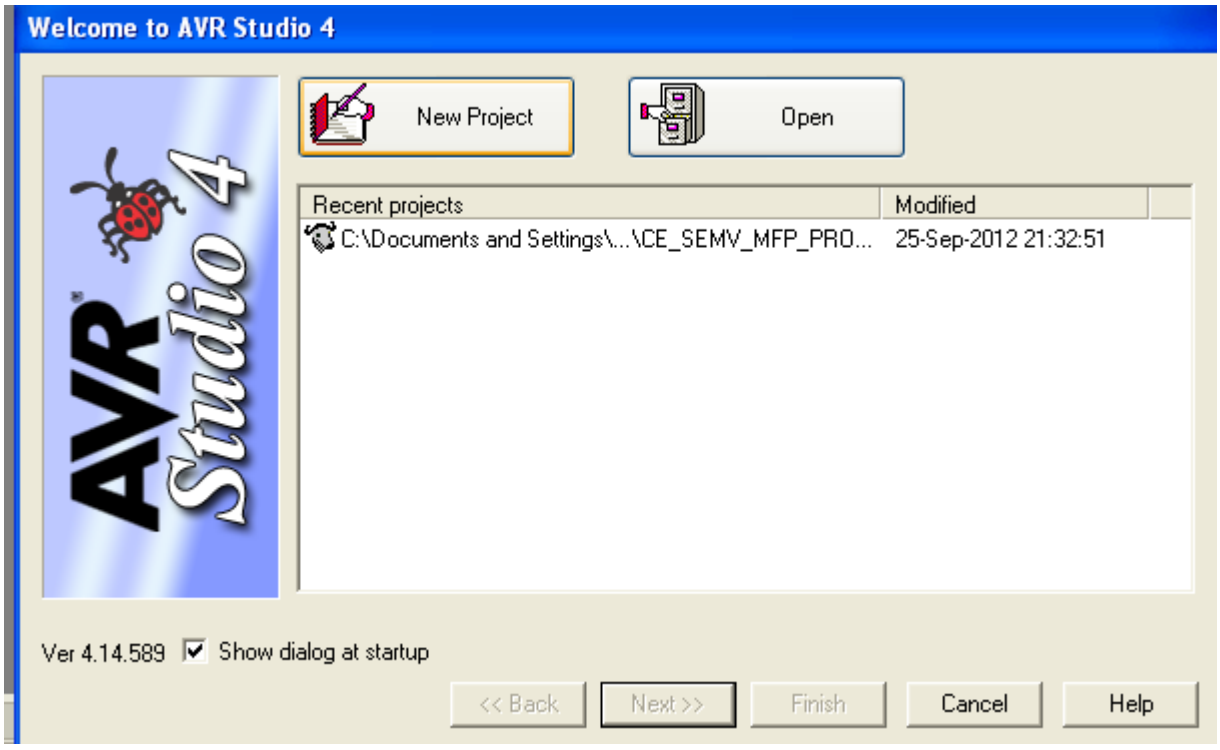
Complete Code

```
#include <avr/io.h>

#include "lcd.h"

void InitADC()
{
ADMUX=(1<<REFS0);                              // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor
=128
}

uint16_t ReadADC(uint8_t ch)
{
   //Select ADC Channel ch must be 0-7
   ch=ch&0b00000111;
   ADMUX|=ch;

   //Start Single conversion
   ADCSRA|=(1<<ADSC);

   //Wait for conversion to complete
   while(!(ADCSRA & (1<<ADIF)));

   //Clear ADIF by writing one to it
   //Note you may be wondering why we have write one to clear it
   //This is standard way of clearing bits in io as said in datasheets.
   //The code writes '1' but it result in setting bit to '0' !!!

   ADCSRA|=(1<<ADIF);

   return(ADC);
}

void Wait()
{
   uint8_t i;
   for(i=0;i<20;i++)
      _delay_loop_2(0);
}

void main()
{
   uint16_t adc_result;

   //Initialize LCD
   LCDInit(LS_BLINK|LS_ULINE);
   LCDClear();

   //Initialize ADC
   InitADC();

   //Put some intro text into LCD
```

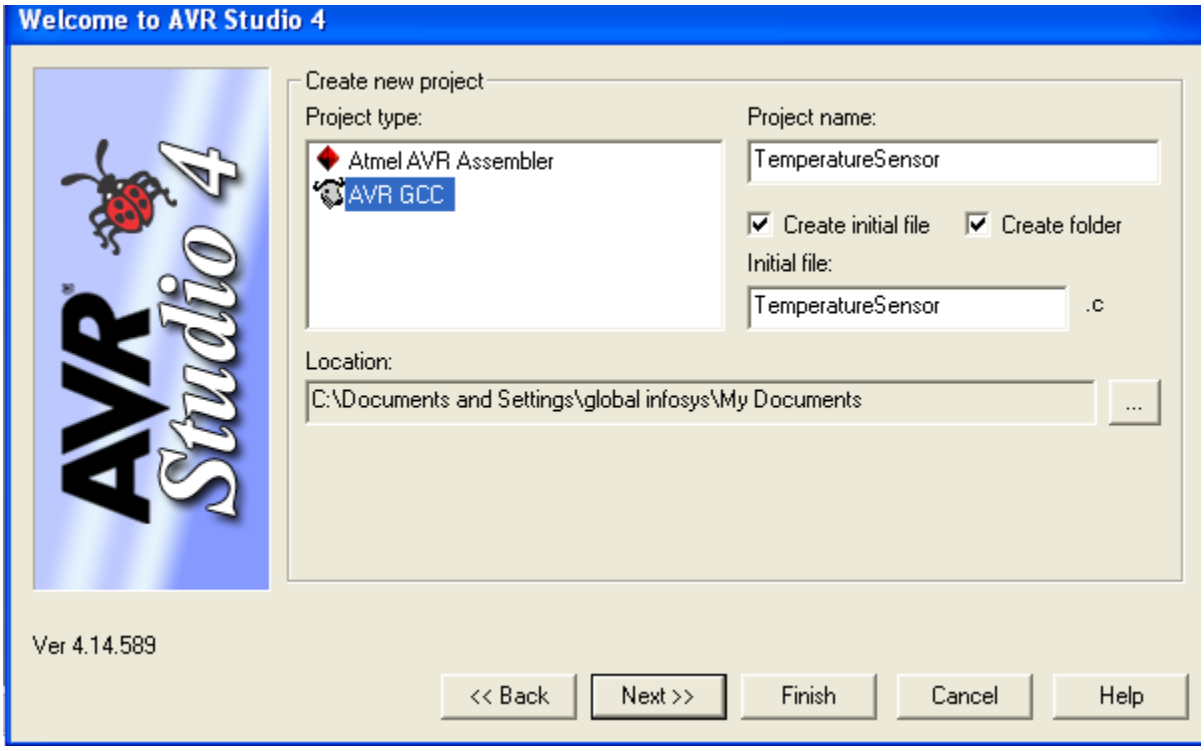
```
    LCDWriteStringXY(0,1,"temp=");

    while(1)
    {
       adc_result=ReadADC(0);            // Read Analog value from channel-0

         // Voltage to temperature conversion logic

      adc_result*=25;
      adc_result=adc_result/1000;

          LCDWriteIntXY(4,1,adc_result,4); //Print the value in 4th column
       second line
       Wait();
    }
}
```

Compiling code in AVR Studio

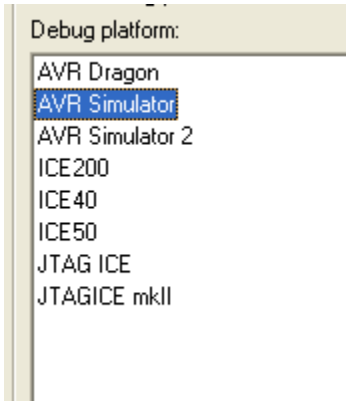
1. Open AVR studio. Click NewProject:

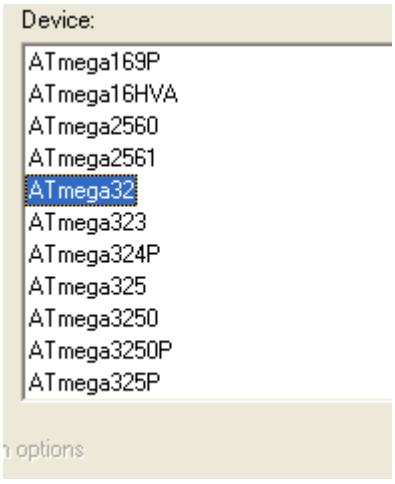


2. Select project type and name as follows and click on Next.

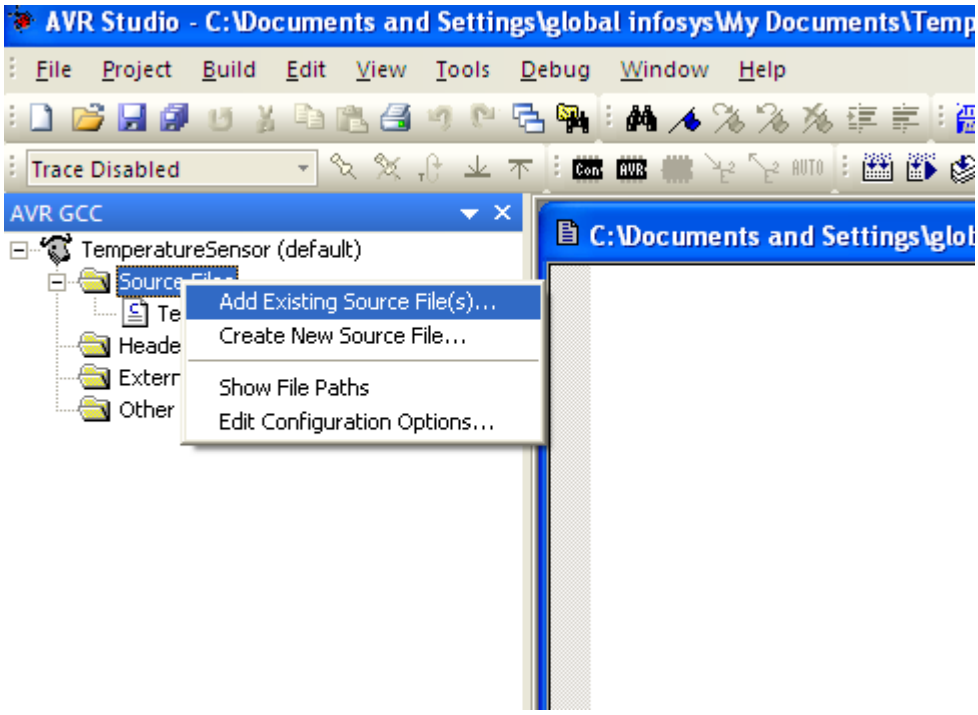


3. Select Debug platform and MCU device and click Finish as follows:

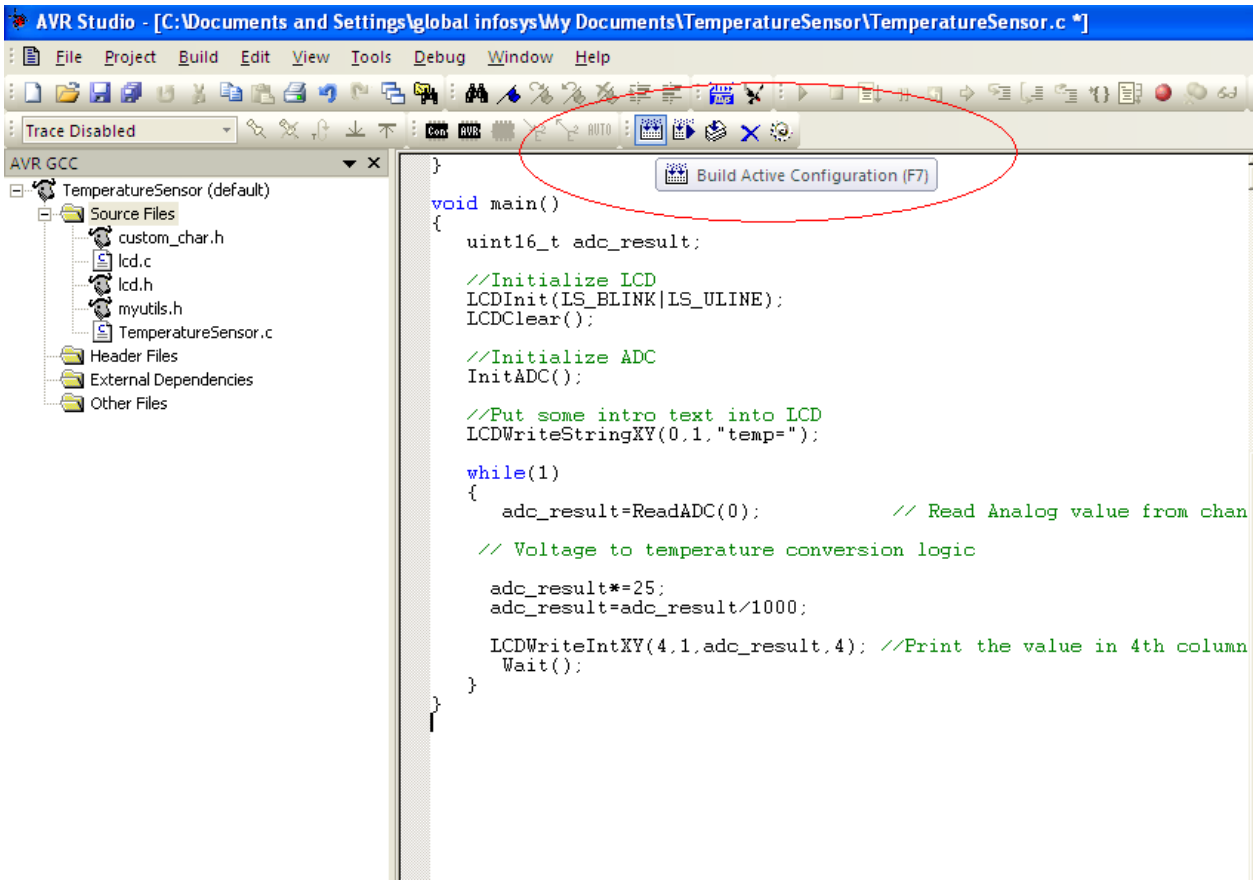




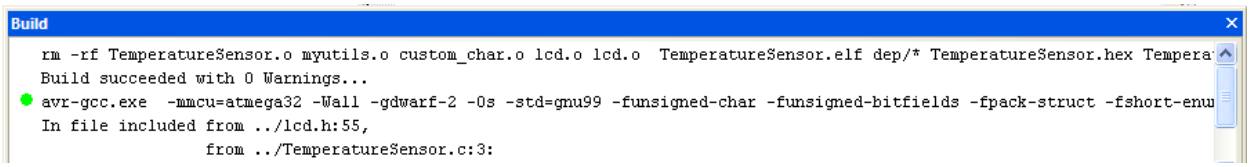
4. Add header files to folder by right-click as follows and also add these files by copying them to folder where your current AVR project is located:



5. Add header file lcd.h from your file-system and copy code onto new available in center window and then compile code using following button to generate hex file:



6. When you click on build following message appears at bottom of your window:



CODE BURNING ON ATMEGA32

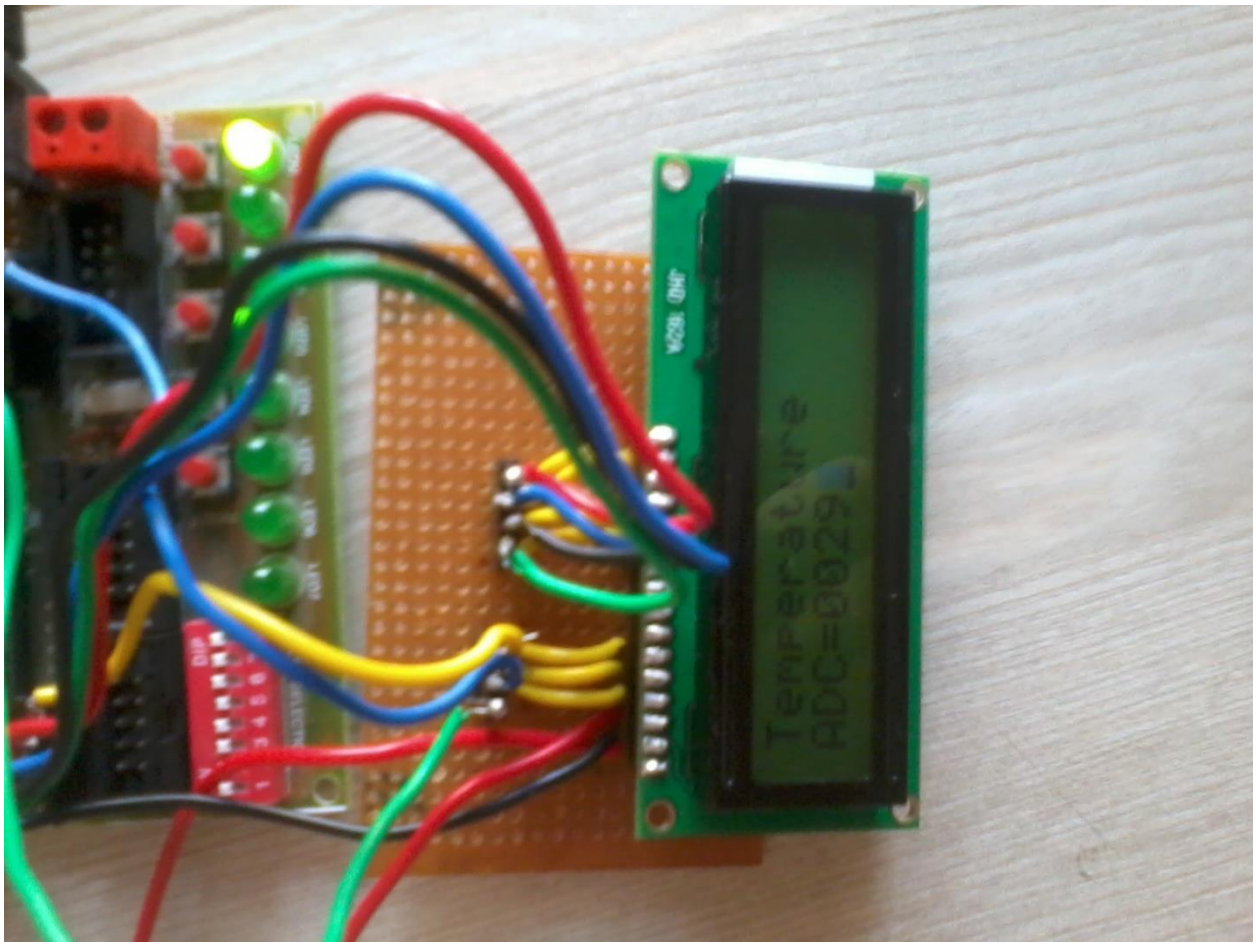
Once compilation is done an object file is generated in “default” folder located at path where your current AVR project is located.

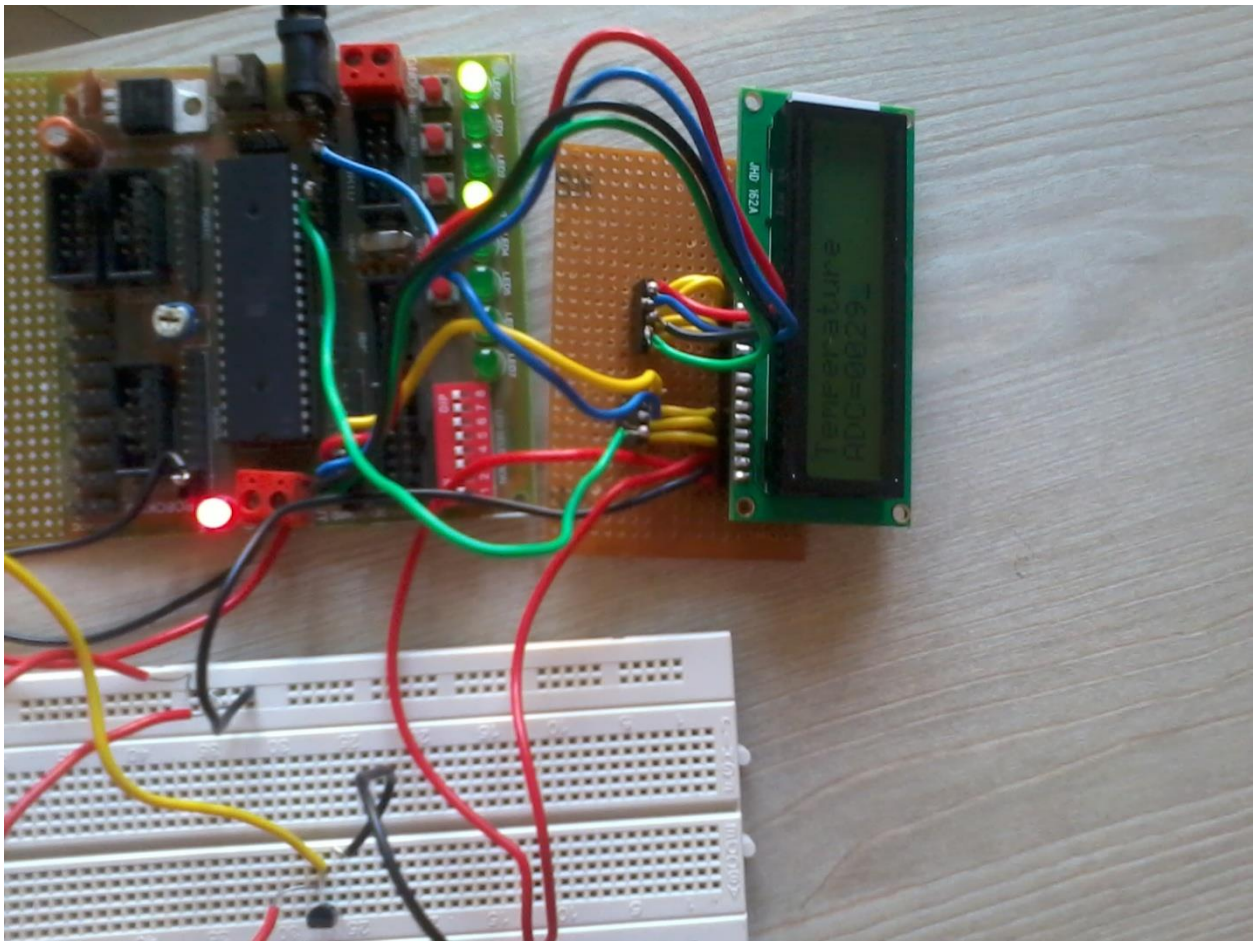
Open Pony Prog software and load this object file in it. Connect Atmega32 with computer using serial connector cable. Click on Load button of Pony Prog to mount the code onto MCU.

Once done , turn on power supply of SDK and ……..!!!!

LCD displays Current room Temperature…

Snaps of Working Temperature Sensor





APPLICATIONS

This mini project can be employed in various fields where temperature sensing is required. This sensor can be used in various appliances which may either home appliances or industrial equipments.

Some of Real World Appliances using this Sensor are:

Air-Conditioners	Refrigerators
Microwave Ovens	Boilers

CONCLUSION

Objective intended to be served is to automate the industrial processing units as well as home appliances. All those areas where temperature plays a sensitive role and involves risks for mankind, a temperature sensor serves as a vital option.

Controlled environment are those in which certain actions need to be performed based upon certain conditions. These conditions need to be checked at regular intervals of time. One such environment could be a microwave oven. Sensing temperature inside can't be done manually. Also automation is required to satisfy customers, to turn off oven automatically when desired temperature is achieved.

Though not much useful as standalone device , but can be used to achieve controlled environment with other large equipments and devices. Also the same project without any additional hardware can used as fire-alarm by just modifying the code to constantly check current temperature with predefined temperature and beep alarm when the two matches.

Thus , this mini project can serve as a strong base in many real world applications.