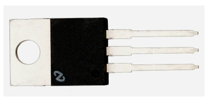
Project 13

Serial Temperature Sensor

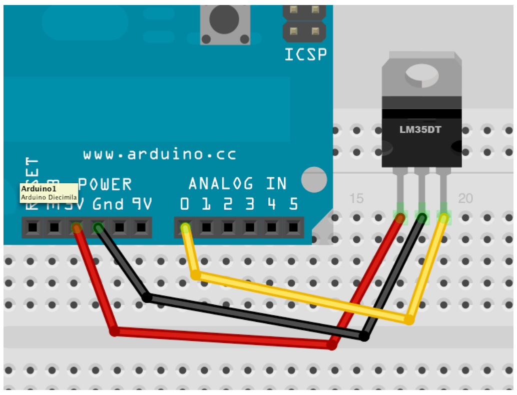
Project 13 - Serial Temperature Sensor

Now we are going to make use of the Temperature Sensor in your kit, the LM35DT. You will need just one component.

What you will need

LM35DT	

Connect it up



Enter the Code

Enter the code, then press the Serial Monitor button on the Arduino IDE. You will now get a reading every half a second(ish) that shows the analog reading from Pin 0 and also the temperature (after conversion) from the LM35DT sensor.

Leave it a little while to stabilise and then hold the sensor. You will see the temperature rise as it reads the temperature of your skin. Hold something cold against it and see it drop. The sensor can read between 0 and 100 degrees C.

```
int potPin = 0;
float temperature = 0;

void setup()
{
  Serial.begin(9600);
  Serial.println("LM35 Thermometer     ");
  analogReference(INTERNAL);
}

void printTenths(int value) {
  // prints a value of 123 as 12.3
    Serial.print(value / 10);
    Serial.print(".");
    Serial.println(value % 10);
}

void loop() {
  int span = 20;
  int aRead = 0;
  for (int i = 0; i < span; i++) {
    aRead = aRead+analogRead(potPin);
  }
  aRead = aRead / 20;

  temperature = ((100*1.1*aRead)/1024)*10;
  // convert voltage to temperature
  Serial.print("Analog in reading: ");
  Serial.print(long(aRead));
  // print temperature value on serial monitor
  Serial.print(" - Calculated Temp: ");
  printTenths(long(temperature));

  delay(500);
}
```

Project 13 - Code Overview

We begin by setting variables to store the Analog Pin we will be using and a place to store the temperature read in from the sensor.

```
int potPin = 0;
float temperature = 0;
```

Then in our setup function a Serial object is created running at 9600 baud. A message stating “LM35 Thermometer” is sent to the Serial Monitor (with a newline).

```
void setup()
{
  Serial.begin(9600);
  Serial.println("LM35 Thermometer     ");
```

Finally, we come across a new command

```
 analogReference(INTERNAL);
```

The analogReference command configures the reference voltage used for the analog inputs. When you use an analogRead() function (like we did in Project 6 to read values from a potentiometer), the function will return a value of 1023 for an input equal to the reference voltage.

The options for this function are:

- DEFAULT: the default analog reference of 5 volts
- INTERNAL: an in-built reference, equal to 1.1 volts
- EXTERNAL: the voltage applied to the AREF pin is used as a reference

In our case we have used an internal reference (of 1.1 volts) which means voltages of 1.1v or higher from the temperature sensor will give an analog reading of 1023. Anything lower will give a lower value, e.g. 0.55 volts will give 512.

We use a reference of 1.1v because the maximum voltage out from the LM35DT Temperature Sensor is 1 volt. The sensor can read between 0 Degrees C and 100 Degrees C with 0 Degrees C being an output voltage of 0 volts and 100 Degrees C being a voltage of 1 volt. If we were to not use the INTERNAL setting and leave it at the default (5 volts) then we would be reducing the resolution of the sensor readings as 100 Degrees C would only be using 20% of the resolution of the Arduino’s ADC (Analog to Digital Convertor) which can convert analog voltages between 0 and 5 volts into digital readings between 0 and 1023.

Next we create a function called printTenths (remember we can put functions before or after setup and loop).

```
void printTenths(int value) {
  // prints a value of 123 as 12.3
    Serial.print(value / 10);
    Serial.print(".");
    Serial.println(value % 10);
}
```

This function is designed to turn the integer values from analog pin 0 and show the fractions of a degree. The Arduino’s ADC reads values between 0 and 1023. Our reference voltage is 1.1 volts and so the maximum reading we will get (at 100 Degrees C) will be 931 (1024/1.1). Each of the 1024 values from the ADC increment in steps of 0.00107421875 volts (or just over 1 millivolt). The value from the ADC is an integer value so the printTenths function is designed to show the fraction part of the temperature reading.

We pass the function an integer ‘value’, which will be the reading from the temperature sensor. The function prints the value divided by 10. E.g. If the reading were 310, this would equate to 33.3 degrees (remember 100 Degrees C is a reading of 931 and 1/3 of that is 310 (the value passed to printTenths is worked out in the main loop and we will come to see how that is calculated shortly).

When the `Serial.print(value / 10)` command prints out 33.3, it will only print the 33 part of that number as the variable ‘value’ is an integer and therefore unable to store fractions of 1. The program then prints a decimal point after the whole number `Serial.print(".");`

Finally, we print out what is after the decimal point using the modulo (%) command. The modulo command works out the remainder when one integer is divided by another. In this case we calculate `value % 10` which divides ‘value’ by 10, but gives us the remainder instead of the quotient. This is a clever way of printing a floating pointer number, which was derived from an integer value.

Let’s now take a look at the main loop of the program and see what is going on here.

```
void loop() {
  int span = 20;
  int aRead = 0;
  for (int i = 0; i < span; i++) {
    aRead = aRead+analogRead(potPin);
  }
  aRead = aRead / 20;

  temperature = ((100*1.1*aRead)/1024)*10;
  // convert voltage to temperature
  Serial.print("Analog in reading: ");
  Serial.print(long(aRead));
    //  print  temperature  value  on  serial
monitor
  Serial.print(" - Calculated Temp: ");
  printTenths(long(temperature));

  delay(500);
}
```

The start of the loop sets up two local variables (variables whose 'scope', or visibility, is only between the curly braces of the function it is within) called 'span' and 'aRead'. A for loop is then set up to loop between zero and 20 (or whatever value is stored in the 'span' variable). Within the for loop the value read in from analogPin(0) is added to the value stored in aRead.

```
 for (int i = 0; i < span; i++) {
    aRead = aRead+analogRead(potPin);
  }
  aRead = aRead / 20;
```

The, after the for loop, the total value of aRead is divided by 20 (or whatever value is stored in 'span'). This gives us an average value read in from the temperature sensor, averaged out over 20 consecutive readings. The reason we do that is because analog devices, such as our temperature sensor, are prone to fluctuations caused by electrical noise in the circuit, interference, etc. and therefore each reading, out of a set of 20, will differ slightly. To give a more accurate reading, we take 20 values from the sensor and then average them out to give us a more accurate reading. The readings are taken one after the other, without any delay and therefore it will take only a tiny fraction of a second for the Arduino to perform this task.

We now have an averaged reading from the analogPin connected to the temperature sensor, which will be some value between 0 and 930 (o to 100 degrees C respectively). That value now needs to be converted into a temperature in degrees C and the next line performs that function:

```
temperature = ((100*1.1*aRead)/1024)*10;
```

This calculation multiplies the value from the digital pin by 1.1 (our reference voltage) and again by 100. What this does is stretch out or values from 0 to 930 to be a value between 0 and 1023 (100*1.1*aRead). This value is then divided by 1024 to give is a maximum value of 100, which in turn is multiplied by 10 to add an extra digit to the end, enabling the modulo function to work.

Let's look at that calculation step by step. Let us presume, for example, that the temperature being read is 50 degrees C. As we are using a reference voltage of 1.1 volts, our maximum value from the sensor will be 930 as the sensors maximum output voltage is 1 volt. 50 Degrees C will therefore be half of that, or 465.

If we put that value into our equation we get :-

(100 * 1.1 * 465.5) = 51205
51205 / 1024 = 50
50 * 10 = 500

When passed to the printTenths() function we will get a temperature of 50.0

Let's try another example. The temperature is 23.5 Degrees C. This will be read as a value of 219

(100 * 1.1 * 219) = 24090
24090 /1024 - 23.525
23.525 * 10 = 235

When passed to the printTenths() function we get 23.5

After we have calculated the temperature, the program then prints out "Analog in reading: : to the Serial Monitor, then displays the value of aRead followed by "Calculated Temp: " and the value stored in 'temperature'. (passed to the printTenths function).

The value of 'temperature' has the word long before it when we pass it to printTenths. This is an example of 'casting' or forcing one variable type to become another. The printTenths function is expecting an integer, we pass it a long type instead. Any values after the decimal point are truncated (ignored).

E.g.

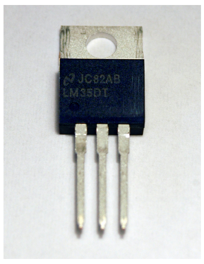
```
int i;
float f;

f = 3.6;
i = (int) f; // now i is 3
```

In this example we have cast the floating point variable f into an integer.

Finally the program delays half a second and then repeats.

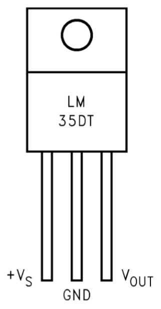
Project 13 - Hardware Overview

The hardware used for this project is very simply a LM35DT Temperature Sensor and 3 wires.



The LM35DT is an analogue temperature sensor that can read from 0 to 100 Degrees C and is accurate to within 0.5 degrees.

The device requires a power supply of anywhere between 4 to 30V DC. The output from the LM35DT will be dependent on input voltage. In our case we are giving the device 5V from the Arduino and therefore 0 Degrees C will give an output voltage of 0 volts. 100 Degrees C will give the maximum output voltage (which will match the input voltage) of 5 volts.



If we take a look at the diagram of the pinouts from the LM35DT datasheet, you can see that there are 3 legs to the device. The left hand leg (with the device number facing you and heatsink away from you) is the input voltage. The middle leg goes to ground and the right hand leg gives you the output voltage, which will be your temperature reading.