Appendix A

Hardware

A.1 Ceramic Resistor

Ceramic resistors is formed of a compound and/or a complex compound containing at least four metallic or metalloid elements including Mg and Si have resistance against high voltage pulses and high power surges, and also have thermal resistance because their main constituent is ceramic. Thus, they have particular advantages which other resistors do not have.

Figure A.1 ceramic resistors

Ceramic resistors of doped barium titanate display relatively low electrical resistance in one temperature range and display very sharply increased resistance when heated to an anomaly temperature, the noted resistance levels and the anomaly temperature being characteristic of the particular resistor composition. These resistors have found wide use as thermal sensing control elements, as self-regulating electrical heaters, and as current limiting devices.

A.2 TEMPERATURE SENSORS

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1⁄4°C at room temperature and ±3⁄4°C over a full −55 to +150°C temperature range.

FigureA.2: LM35 Package

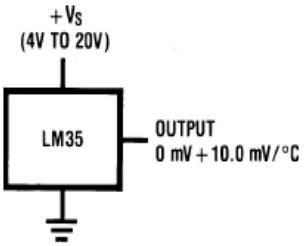

Figure A.3: LM35 pin configuration

The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy).

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01°C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is especially true for the TO-92 plastic package, where the copper leads are the

principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up all circuit with an isolated case which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die’s temperature will not be affected by the air temperature.

A.3 RS232C LEVEL CONVERTER

A standard serial interfacing for PC, RS232C, requires negative logic, i.e., logic '1' is -3V to -12V and logic '0' is +3V to +12V. To convert a TTL logic, say, TxD and RxD pins of the uC chips, thus need a converter chip. A MAX232 chip has long been using in many uC boards. It provides 2-channel RS232C port and requires external 10uF capacitors. Carefully check the polarity of capacitor when soldering the board.

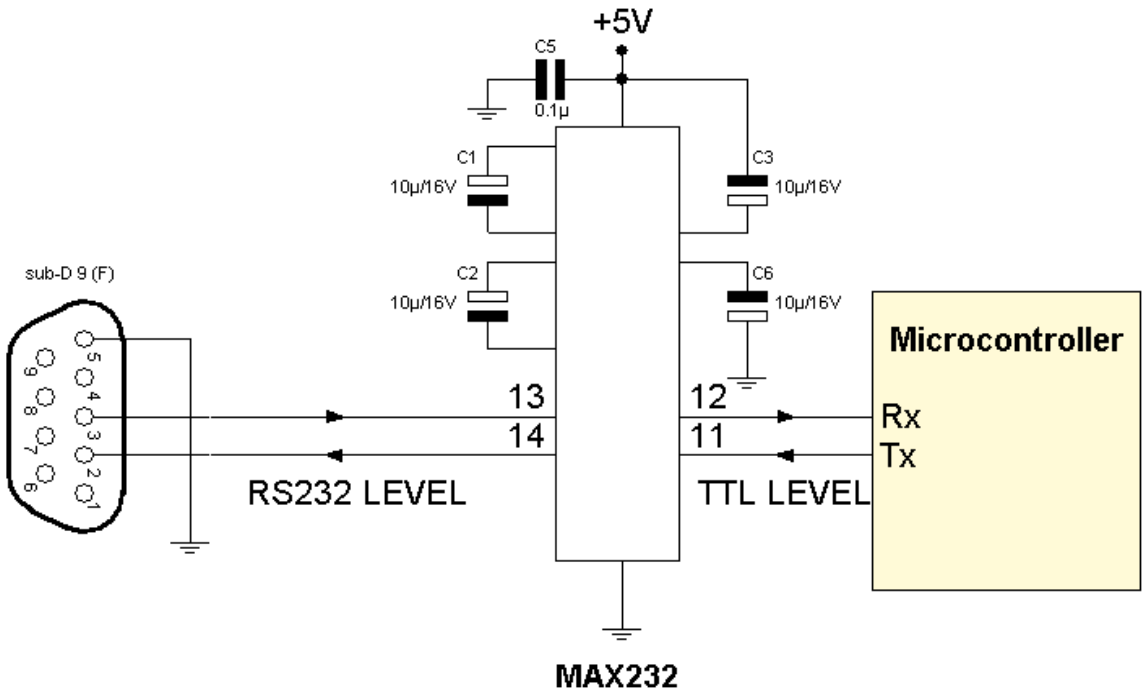

Figure A.4: MAX232 Schematic

A.5 ATMEGA16

The ATmega16 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 ATmega16 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The ATmega16 provides the following features: 16K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 512 bytes EEPROM, 1K byte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, 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), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.

The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-

power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

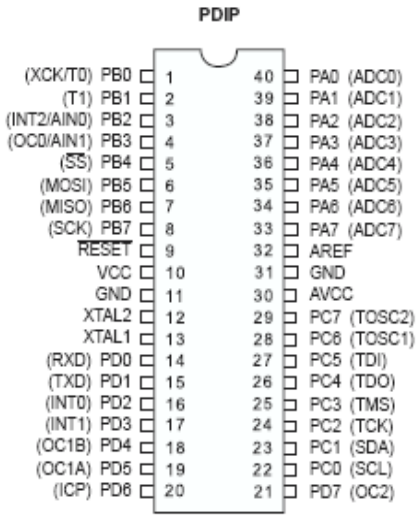

Figure A.5: ATMEGA16 PINOUT

The ATmega16 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Appendix B

Software

B.1 BASCOM-AVR Embedded Software

BASCOM is an Integrated Development Environment (IDE) that supports the 8051 family of microcontrollers and some derivatives as well as Atmel's AVR microcontrollers. Two products are available for the various microcontrollers - BASCOM-8051 and BASCOM-AVR.

BASCOM-AVR is not only a BASIC Compiler, but also a comfortable Integrated Development Environment (IDE) running under Windows 95 and Windows NT. Such a development environment supports the whole process from coding and testing a program to programming the used microcontroller.

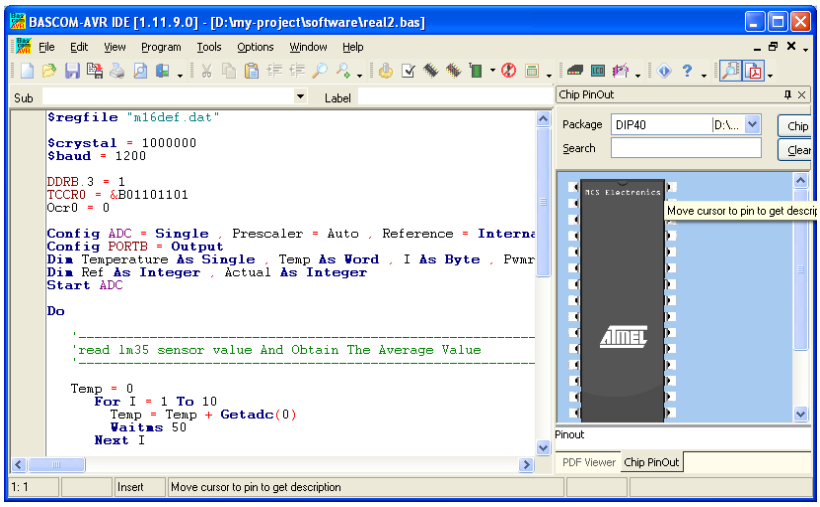

Figure B.1: BASCOM AVR Environment

After the start of BASCOM you can create a new file by selecting ***File>New*** or open an existing file by selecting ***File>Open***. In the next step, check such BASCOM Options like device selection, baud rate, clock frequency and other relating options. A detailed explanation of these options will be given in the next chapter.

Now you may edit the BASIC source and compile it afterwards. As a rule, the compiler detects here the first errors and the program must be debugged. The BASIC source must be edited as long as the compilation is without any errors. Normally, the process of editing, compiling and debugging needs to be

repeated several times. It makes no sense to debug all errors in one step. Editing several typing errors in one step is no problem. But for more difficult errors, a separate compiler run checks the validity of the changes carried out. It is always easier to debug a localized error.

With the help of the internal BASCOM Simulator the program operation can be checked without any hardware. The probably last task in a project is programming the device that is used in the application hardware, followed by an excessive test of the program on the target.

The project proves to be successful if these tests document a proper function in the target hardware. Otherwise, some steps must be repeated. Before working with the BASCOM-AVR, the development environment will be described by means of a small program example.

B.2 Data Acquisition program:

- the header of the program describe the microcontroller type and system frequency and the baud rate

```
$regfile "m32def.dat"
$crystal = 1000000
$baud = 1200
```

- this section describe the declaration of PWM signal parameter

```
DDRB.3 = 1
TCCR0 = &B01101010
OCR0 = 0
Config Input0 = Crlf , Echo = Nil
Config PORTB = Output
Dim Pwmrate As Byte
```

- Setting internal ADC channel 0 Parameter

```
Config ADC = Single , Prescaler = Auto , Reference = Internal
Dim Temperature As Single , Temp As Word , I As Byte
Start ADC
```

- Receiving PWM value from serial port and send it to actuator

```
Input Pwmrate Noecho

Waitms 10

OCR0 = Pwmrate
Waitms 100
```

- reading of current temperature and convert the value to numerical value

```
Temp = 0
   For I = 1 To 10
     Temp = Temp + Getadc(0)
     Waitms 10
   Next I

'Calculate Temperatuer And Obtain The Average Value

Temperature = Temp * 12.8 : Temperature = Temperature / 1023

Temperature = Temperature * 2
```

- send temperature value to PC serial port

```
Print Temperature
Waitms 10
```

The complete program:

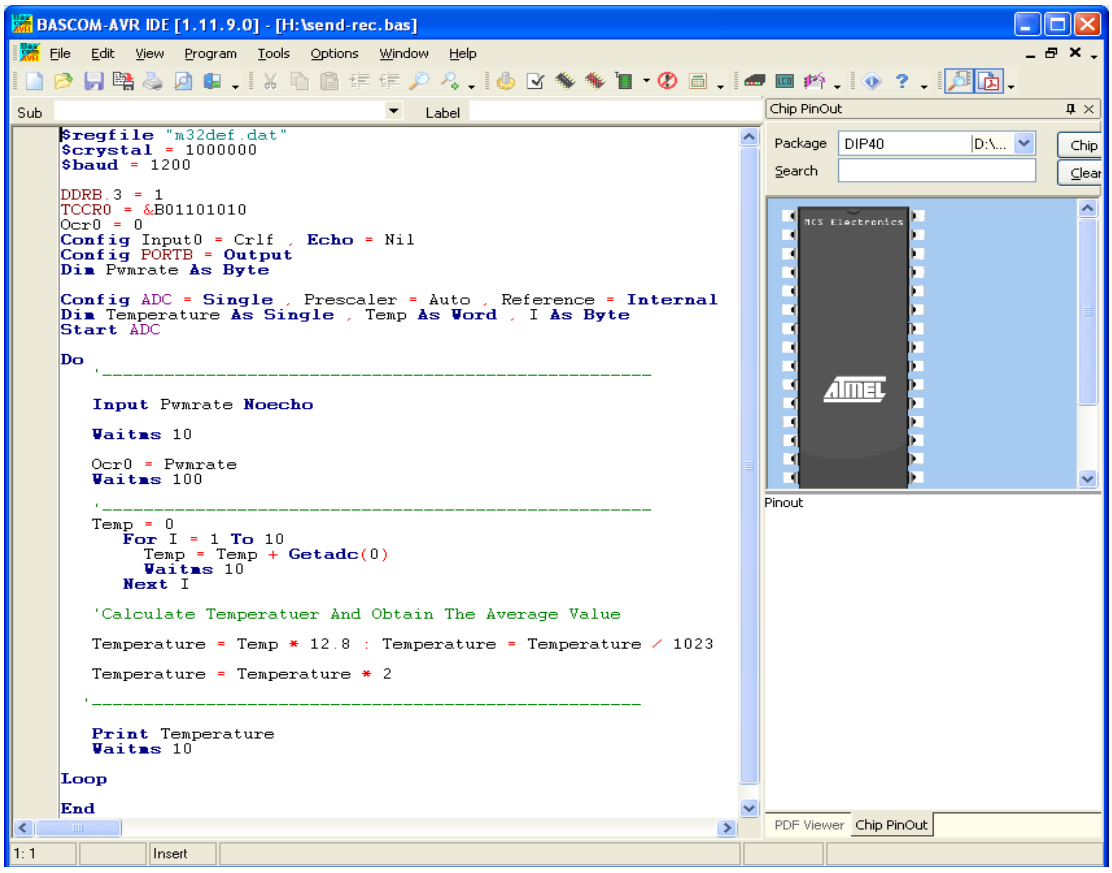

Figure B.2: microcontroller program

B.3 MATLAB Fuzzy Logic Toolbox

The Fuzzy Logic Toolbox is a collection of functions built on the MATLAB numeric computing environment. It provides tools to create and edit fuzzy inference systems within the framework of MATLAB, or by integrating fuzzy systems into simulations with SIMULINK. Also it possible to build stand-alone C programs that call on fuzzy systems which built with MATLAB. This toolbox relies heavily on graphical user interface (GUI) tools to help accomplish work, although there is alternative to work entirely from the command line.

The toolbox provides three categories of tools:

- Command line functions
- Graphical interactive tools
- SIMULINK blocks and examples

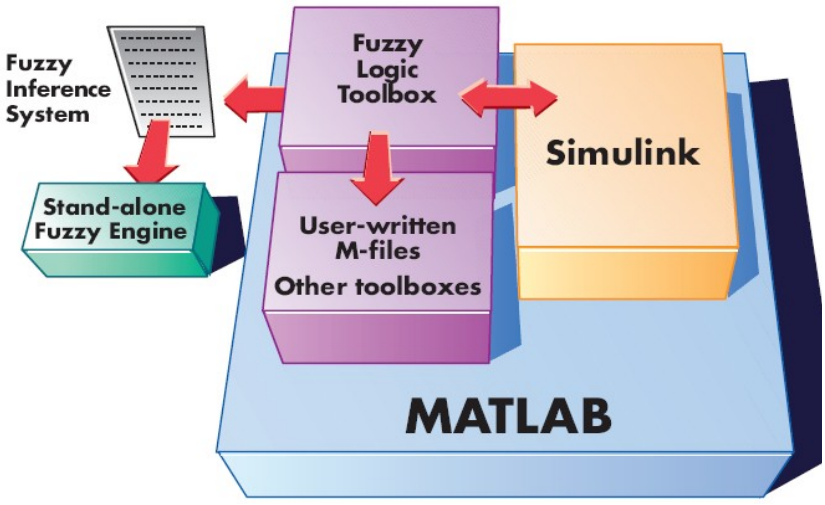

Figure B.3: fuzzy MATLAB tool box

The design will follow Graphical interactive tools approach. There are five primary GUI tools for building, editing, and observing fuzzy inference systems in the Fuzzy Logic Toolbox: the Fuzzy Inference System or FIS Editor, the Membership Function Editor, the Rule Editor, the Rule Viewer, and the Surface Viewer. These GUIs are dynamically linked, in that changes make to the FIS using one of them, can affect any of the other open GUIs.

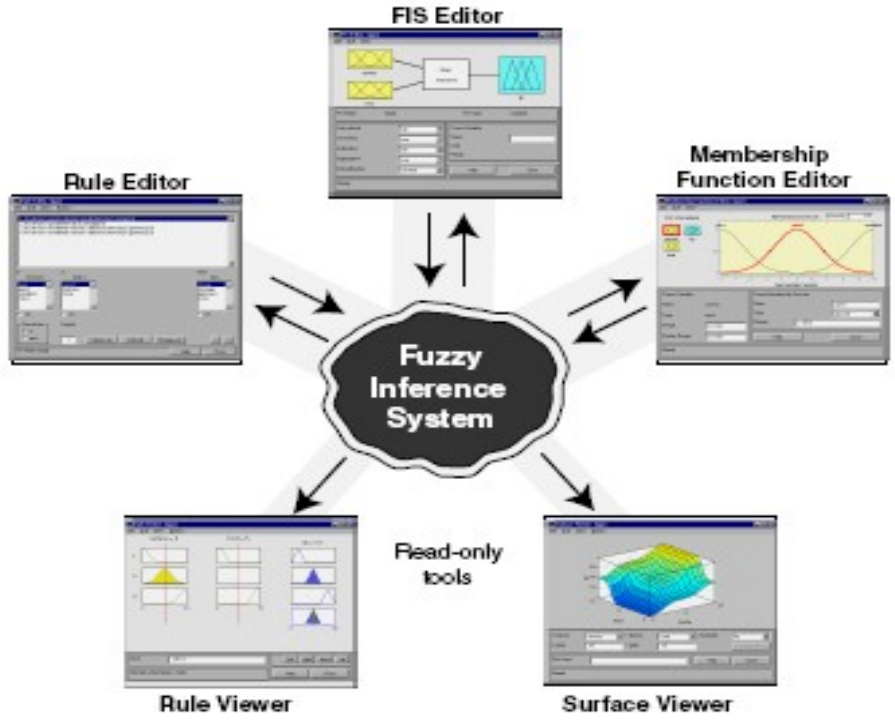

Figure B.4: MATLAB GUI tools for building fuzzy

The FIS Editor handles the high-level issues for the system: How many inputs and output variables? What are their names? The Fuzzy Logic Toolbox doesn't limit the number of inputs. However, the number of inputs may be limited by the available memory of your machine. If the number of inputs is too large, or the number of membership functions is too big, then it may also be difficult to analyze

the FIS using the other GUI tools. The Membership Function Editor is used to define the shapes of all the membership functions associated with each variable.

The Rule Editor is for editing the list of rules that defines the behavior of the system. The Rule Viewer and the Surface Viewer are used for looking at, as opposed to editing, the FIS. They are strictly read-only tools. The Rule Viewer is a

MATLAB based display of the fuzzy inference diagram shown at the end of the last section. Used as a diagnostic, it can show (for example) which rules are active, or how individual membership function shapes are influencing the results. The Surface Viewer is used to display the dependency of one of the outputs on any one or two of the inputs — that is, it generates and plots an output surface map for the system.

B.4 System without control code

```
% parameter initialize
                clc
                clear T O
                SetTemp=40;
                error= 0;
                ActualTemp= 0;
                pwm=0;
% control program
%create serial object
                s = serial('COM2','BaudRate',1200, 'terminator', 'CR/LF');
                set(s, 'TimeOut', 1);
                fopen(s);
% control loop
                for i=1:200
                clc
                fwrite(s, pwm, 'char'); % send PWM signal to microcontroller
                fprintf(s,'\r');
                ActualTemp =(fscanf(s,'%g')); % read temperature
                T(i)= ActualTemp;
                error= SetTemp - ActualTemp; % calculate error
                pwm = error;
                O(i)=pwm;
                pwm=int2str(pwm);
                pause(0.51)
                end
% plot the graph
                fwrite(s, '0', 'char');
                fprintf(s,'\r');
                fclose(s)
                delete(s)
                clear s
                plot(0:199,T,0:199,O,0:199,SetTemp);
```

B.5 System with PD-fuzzy control code

- **Initialization:**

```
clc
clear T O
SetTemp=50;
PreviousError=0;
error= 0;
DelError=0;
NextError=0;
ActualTemp= 0;
pwm=0;
```

- **fuzzy controller Gain parameter**

```
ge =1.5; error
gde =1;  change of error
gu = 1;  out put
```

- **create com port for serial communication**

```
s = serial('COM2','BaudRate',1200, 'terminator', 'CR/LF');
set(s, 'TimeOut', 1);
fopen(s);
```

- **control loop for 150 cycle**

```
for i=1:150
clc
```

- **send pwm rate to Atmega32**

```
fwrite(s, pwm, 'char');
fprintf(s,'\r');
```

- **read current temperature from Atmega32**

```
ActualTemp =(fscanf(s,'%g'));
T(i)= ActualTemp;
```

- **calculate error and change of error rate**

```
error= SetTemp - ActualTemp;
DelError = error - PreviousError ;
DelError = DelError ./1;
```

- **send error and change of error rate as input to fuzzy controller and receive pwm as out put**

```
A = READFIS('wall’);
FUZZPWM = GU*EVALFIS([GE*ERROR    GDE*DELERROR],A);
PWM = ROUND((FUZZPWM*255)./100);
IF PWM >=255
PWM =255;
        END
        O(I)=PWM;
        PWM=INT2STR(PWM);
        PREVIOUSERROR = ERROR;
        PAUSE(0.51)
        END
```

- **plotting the output of controller**

```
        FWRITE(S, '0', 'CHAR');
        FPRINTF(S,'\R');
        FCLOSE(S)
        DELETE(S)
        CLEAR S
        PLOT(0:149,T,0:149,O,0:149,SETTEMP);
```

B.6 System with PID fuzzy control code

```
clc
clear T O
SetTemp=45;
PreviousError=0;
error= 0;
DelError=0;
NextError=0;
ActualTemp= 0;
pwm=0;
prvinterror=0;
%fuzzy controller gains
ge =1;
gde = 1;
gu = 1;
ki=1;
s = serial('COM2','BaudRate',1200, 'terminator', 'CR/LF');
set(s, 'TimeOut', 1);
fopen(s);
for i=1:200
clc
fwrite(s, pwm, 'char');
fprintf(s,'\r');
ActualTemp =(fscanf(s,'%g'));
T(i)= ActualTemp;
SetTemp;
error= SetTemp - ActualTemp;
DelError = (error - PreviousError) ;
interror = ki*(error + PreviousError)/2;
%DelError = DelError ./0.4;
a = readfis('last');
fuzzpwm =
(gu*evalfis([ge*error gde*DelError],a))+(interror+prvinterror);
prvinterror=interror;
pwm = round((fuzzpwm*255)./100);
```

```
    if pwm >=255
       pwm =255;
    end
    O(i)=pwm;
    pwm=int2str(pwm);
    PreviousError = error;
    pause(0.51)
    end
    fwrite(s, '0', 'char');
    fprintf(s,'\r');
    fclose(s)
    delete(s)
    clear s
    plot(0:i-1,T,0:i-1,SetTemp
```

B.7 System with PID control code

```
clc
clear T O
SetTemp=45;
PreviousError=0;
error= 0;
ActualTemp= 0;
pwm=0;
previous_Integral = 0;
%pid controller gains
kp =300;
kd =3;
ki =400;
s = serial('COM2','BaudRate',1200, 'terminator', 'CR/LF');
set(s, 'TimeOut', 1);
fopen(s);
for i=1:200
clc
fwrite(s, pwm, 'char');
fprintf(s,'\r');
ActualTemp =(fscanf(s,'%g'));
T(i)= ActualTemp;
error= SetTemp - ActualTemp;
pid_integral= ki*(error+PreviousError)/2 ;
pwm = kp*error + (pid_integral+ previous_Integral) + kd*(error -
      PreviousError);
previous_Integral = pid_integral;
if pwm >=255
   pwm =255;
elseif pwm < 0
   pwm = 0;
end
O(i)=pwm;
pwm=int2str(pwm);
PreviousError = error;
pause(0.51)
```

```
    end
    fwrite(s, '0', 'char');
    fprintf(s,'\r');
    fclose(s)
    delete(s)
    clear s
    plot(0:i-1,T,0:i-1,SetTemp);%0:i-1,O,
```

Appendices