DESİGN OF DIGITERM 03 LOW-END CONTROLLER UNİT FOR INCUBABY IB03 INCUBATOR

May 2022

OBJECTİVE:

This lab practice provides a guide to design DigiTerm 03 Low-end controller prototype simulation using an Arduino-Uno for the IncuBaby IB03 incubators.

The **low-end simulation prototype** in Proteus-ISIS for an Arduino-Uno that

i- has an anjustable desired temperature ThD. It is set through a UART command.

ii- measures the voltage of two NTC thermistors,
ThA for heater air temperature,
ThB for baby skin temperature.
It calculates temperatures `ThA` and `ThB` in Celsius, and displays the temperatures `ThA` and `ThB` on a 2-line by 16-char LCD display.

iii- starts alarm state if baby skin temperature *ThB* exceeds 37C,
starts alarm and stops heating if heater air temperature *ThA* exceeds 37C,

iv- Uses *ThD* to control *ThB* by proportional control action, with PWM that switches the heater. In the control, *ThA* shall be limited to maximum 37 C.

v- transmits necessary data to PC, for monitoring and inspecting the history of control modes, parameters, and temperature values once at every minute.

INTRODUCTION

DigiTerm-03 temperature controller unit for an infant incubator, that controls the infant body temperature (*ThB*) by powering an air heater at one of three pre specified desired-ThB temperatures (shortly denoted by *ThD*). *ThD* may get one of the three values: 30, 32, 34 C. Note that temperatures *ThA* > 37 and *ThB* > 37 are known harmful, and should not be allowed in any case. IncuBaby should start an alarm state for both ThA and ThB exceeding 37C.

1. Technical Design Requirements for DigiTerm 03:

You should design **DigiTerm-03** to have following features.

a- It must be developed on an Arduino-Uno r3 board (16MHz clock).

b- It must have a blink-alive LED (internal-LED of Arduino) flashing for 100ms once at every two seconds.

c- Sensors for temperatures *ThA* and *ThB* are two **4k7 NTC** thermistors with a suitable linearization resistor to read temperatures between **20 ... 45** degrees**.** ADC configuration should provide sufficient dynamic quantization ratio to read the sensor in full dynamic range ratio for the temperature range 20 ... 50 C. The measured temperature values shall be used as integers with 0.1 C precision, i.e., 37.5C should be stored as 375 (it is fixed point with one digit decimal fraction).

d- UART shall receive ascii commands

- "`/#d`" for desired-ThB (denote it by ThD) values, where `#` can any nonnegative decimal number such as: `320`, `340`, `360`,
- "`/#p`" for proportional gain Kp value, where `#` can be any nonnegative decimal integer number, i.e., `/340D` sets `ThD=340`, and `/20P` sets `Kp=20.`

e- Control Action

Control action requires proportional feedback gain Kp. At every minute it shall do the following tasks:
- measure both temperatures *ThA* and *ThB*. You can set the temperature of a thermistor directly by setting its resistance to corresponding value.
- calculate e = *ThB* – *ThD*
- calculate percent-Power, *PP* = *Kp* * *e*.
- calculate PWM duty time for 1 minute period, *Td*= 60**PP*/100 seconds
- keep the heater output high for *Td* seconds.

f- Display these values in a clearly recognizable format on the 16x2 LCD display:
- Temperatures *ThA* and *ThB* (integer 10*temperature.in.C)
- *ThD* (desired-ThB) temperature (integer, {320, 340, 360}, represents {32, 34, 36}C with one digit after decimal point.)
- Proportional gain Kp (integer)
- Alarm status, for air temperature *ThA*>37, make AA=1, else AA=0,

g- The UART shall transmit
- Temperatures *ThA*, *ThB*, *ThD*
- Proportional gain *Kp*,
- Button *Bc* (for cover position, low: cover closed, high: cover open),
- Alarm status AA (to display graphically the status of the incubator on the NodeRED dashboard through transfering data to a PC).

2. NTC sensor, sensor-circuit

At the absolute temperature *Tk* (in Kelvin), resistance of 4700B3900NTC sensors obeys to the expression:

$1/Tk = 1/Tko + (1/B)\ln(Rt / Ro)$ or $Rt = Ro\ \exp(B*(1/Tk-1/Tko))$

where *Ro* is normal resistance (=4700 Ohm at 25C room temperature), *Tko* is normal temperature in Kelvin (=298.15K), and B=3900 is the temperature sensitivity parameter, and *Rt* is the resistance (in Ohm) at temperature *Tk* (in K).
The precision of this %1 tolerance sensor at room temperature is 0.2 C.
Using the parameters Ro=4700 and B=3900, we get the following expression for resistance for temperature in Celcuis *Tc*:

$Rt = 4700*\exp(3900*(1/(Tc+273.15) - 1/298.15))$

$Tc = 1/(1/298.15 + (1/3900)*\ln(Rt / 4700)) - 273.15$; in ^oC.

Temperature C	0	5	10	15	20	25
Resistance Ohm	15561	12039	9398	7400	5875	4700
Temperature C	30	35	40	45	50	55
Resistance Ohm	3788	3074	2512	2065	1708	1421

3. ADC of NTC sensor output

AD conversion of both NTC sensors (*ThA* and *ThB*) should be once at every minute. Readings of *ThA* and *ThB* shall be represented as integers with 0.1 C precision (i.e., physical *ThA*=32.6C shall be represented as integer ThA=326).
Arduino ADC can use Vs=5V supply voltage as reference. In our case, since highest precision is required around 36C, NTC+Rs circuit shall have

$Rt = 4700*\exp(3900*(1/(36+273.15) - 1/298.15)) = 2951$ Ohm.

The nearest standard resistor value is 2700 Ohm.

NTC+2700 Ohm sensor circuit will deliver at 20C and 50 C the following voltages.

Vs(20C) = 5* 2700/(2700+5875) = 1.57 V
Vs(50C) = 5* 2700/(2700+1708) = 3.06 V

The ADC reading Ns, for 20 and 50 C are expected to be:

Ns(20C) = 1024*1.57 /5 = 321
Ns(50C) = 1024*3.06 /5 = 626

Now, we can calculate the dynamic quantization error of the ADC for this configuration:

D_Q = (626 – 321) /1 = 305, $D_{Q\text{-}dB}$ = 50 dB.

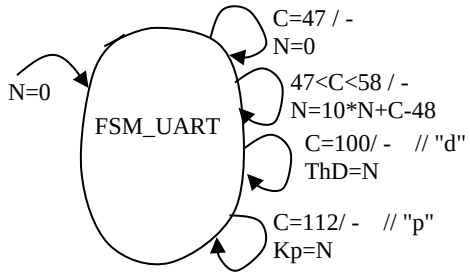
The dynamic error ratio of the NTC sensor for the same temperature is:

Ds = (50 - 20) / 0.2 = 30/0.2 = 150, $D_{s\text{-}dB}$ = 43.5 dB

We found that Dq>Ds, indicating the quantization error of ADC is better but nearly comparable with respect to sensor precision.

4. Design of UART received character FSM

UART commands such as "`/#D`" are easy to obtain by using a temporary integer `N` for composing the numeric value with an FSM. When the received ASCII character `C` is "`/`", that is `C=47`, transition on this guard should accomplish a set action `N=0`. When received character code satisfies `48 ≤ C ≤ 57`, transition should accomplish a set actions `N=N*10+(C-48)`. For ThD, if received character is `C="d"` (=100), then, set action should make `ThD=N`. For the cases of `C="p"`(=112), similarly set action shall provide `Kp=N`.



For your project, you should complete the other missing transition links as required by your project.

5. Rapid Prototyping Practice

The rapid prototyping practice on this project shall be applied to this application in the following manner.

i- The hardware and the software should be written in small steps, and should be tested after each step before starting to the next step. These simple tests for only a small step provides having multiple bugs in software and shrinks the debugging time of the software. Apply each step in incremental design/implementation approach, testing and debugging until the it works successfully.

ii- Document what you added, how you tested, and what you get as a test result for each modification. Save a copy of the design file and program code when it works successfully.

iii- In most embedded development medium there is no automatic source control system. You should apply manual source control through the test points by saving the successful tests. Do not forget saving the project after each test.

6. Overall system tests:

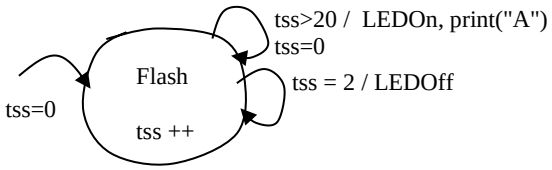
In parallel to the well documented incremental design tests, the team members shall have well documented overall system test. The test procedure shall be clearly written by the test designer into the final design report, and the test results shall be clearly documented in the report.

EXPERİMENTAL PRACTİCE

In this part we practice the implementation of simulation of the DigiTerm 03.

1. Start an Arduino project with Firmware (a-b)

- Start New Project, with name Lab07-01, at your workfolder (use desktop/lab07) for Arduino-Uno, to use Ard.AVR(Proteus) compiler.
- Set the clock rate of Arduino UNO to 16MHz.
- Start 9600 baud serial port through UART for test purpose
- For flashing internal LED, you need to count 0.1 seconds, up to 2 seconds. Declare tss to count 0.1s periods. You need to design the FSM for flashing LED

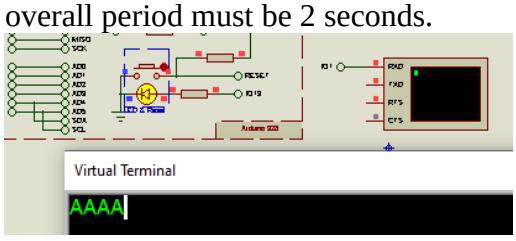


Your program should be something like this:

```
//----------declarations-----------
int tss;

void FSM_flash(){
//  FSM for flashing light.
tss++;
if(tss>=20) {
     digitalWrite(13, 1);
     tss=0;
     Serial.print("A"); // Test of transmit "A"
     }
if(tss == 2) { digitalWrite(13,0); }
}

//----------setup------------------
Serial.begin(9600);
pinMode(13,OUTPUT);
tss=0;
//----------loop-------------------
delay(100);
FSM_flash();
```

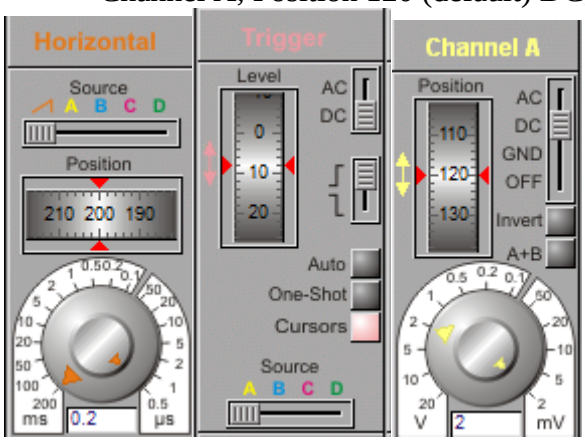
Test of the flashing led design step

Test your circuit by observing transmitted character A, and comparing the flash-on-off-period against the simulated time shown at the bottom bar of the simulator. The overall period must be 2 seconds.

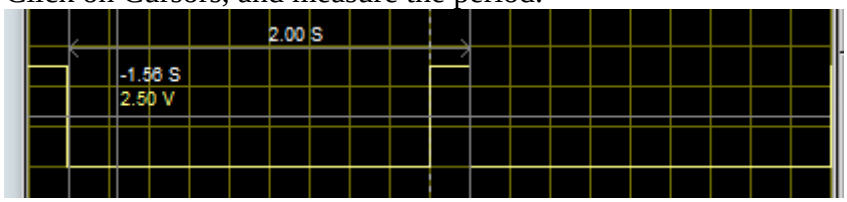


You may connect a virtual oscilloscope at IO13, and set the oscilloscope for:

- Horizontal sweep ( ), Position 200, sweep time 200ms (=0.2s) per division;
- Trigger Level 10, DC, rising edge ( ), Auto, Cursors on.
- Channel A, Position 120 (default) DC, sensitivity 2V per division.



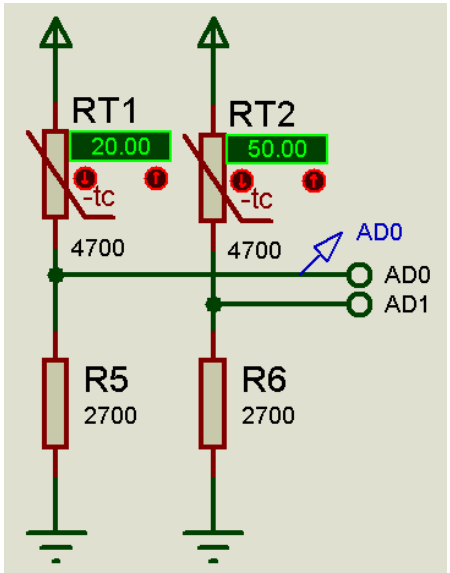
Click on Trigger -One-shot to get a frozen view of waveform.
Click on Cursors, and measure the period.



Save design step, save it by save-project-as with name Lab07-02 for the next part.

2. *Sensor Interfacing, and ADC (c)*

Use two NTC sensors (4700 Ohm B= 3900) and two 2700 Ohm resistors, connected to AD0 and AD1 of the Arduino.



Insert following red colored codes to their places.

```
//----------declarations----------
int tss;
int Ns0, Ns1, ThA, ThB, t1m;
int T1, T2, N1, N2, Tcr;

void FSM_flash(){
//  FSM for flashing light.
```

```
tss++;
if(tss>20) {
     digitalWrite(13, 1);
     tss=0;
     // Serial.print("A"); // Test of transmit "A"
     }
if(tss == 2) { digitalWrite(13,0); }
}


void ADC_AB(){
     // Default ADC resolution is 10-bit
     // Read ADC, and Calculate Temperature
     Ns0 = analogRead(A0);
     ThA = 10.*T1 +Tcr+ (10.*T2-10.*T1)*(Ns0-N1*1.)/(N2-N1*1.);
     Ns1 = analogRead(A1);
     ThB = 10.*T1 +Tcr+(10.*T2-10.*T1)*(Ns1-N1*1.)/(N2-N1*1.);
}

void UART_printT(){
     Serial.println();
     Serial.print(" N0 ");
     Serial.print(Ns0);
     Serial.print(" Ta ");
     Serial.print(ThA);
     Serial.print(" N1 ");
     Serial.print(Ns1);
     Serial.print(" Tb ");
     Serial.print(ThB);
}
```

Procedure ADC_AB reads both of ThA and ThB sensors to Ns0 and Ns1, and calculates `ThA` = 10**ThA* (i.e., *ThA*=34.5C represented as `ThA`=345) by linear interpolation.

Procedure UART_printT transmits the readings Ns0, Ns1, and, the calculated ThA and ThB values. We plan to use these values to make a plot of linearity for the sensors.

```
//----------setup------------------
Serial.begin(9600);
pinMode(13,OUTPUT);
tss=0;

analogReference(DEFAULT);
T1=20; N1=321; T2=50; N2=626; Tcr=0
```

T1, N1, T2, N2 are interpolation corner points, Tcr is the corrective shift for fine calibration of the interpolation. We placed the theoretically calculated values for T1, N1, T2, N2. Our plan is to replace them by actual readings to get better accuracy.

```
//----------loop--------------------
delay(100);
FSM_flash();
t1m++;
if(t1m >= 60) { // tasks at every minute
     t1m=0;
     ADC_AB();
     UART_printT();
     }
```

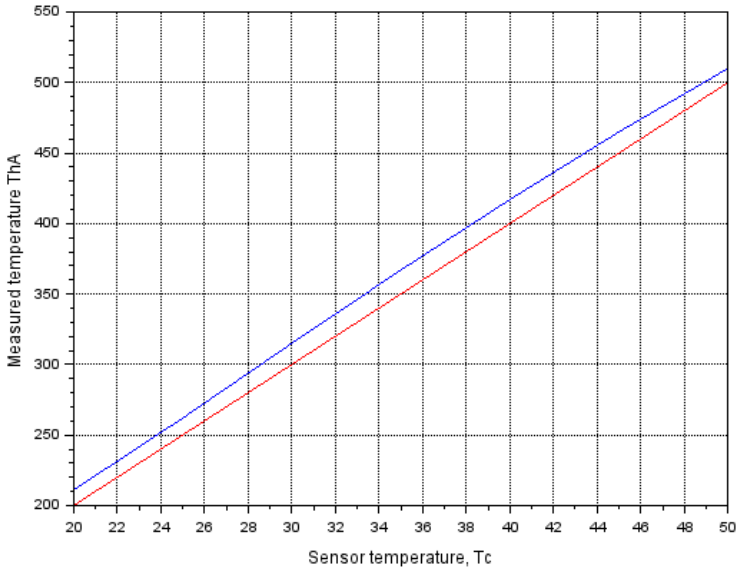
In the loop, we used a time counter that counts one minute period. ADC_AB and UART_printT are called per minute.

Test Readings:

The performance of the sensors can be obtained by plotting their response at 0, 5, 10, ... 45, 50 C temperatures using SciLab Plot.

```
TNTm=[
```

```
20 333 211
25 385 262
30 438 315
35 491 367
40 542 417
45 591 465
50 637 510
];
plot(TNTm(:,1), TNTm(:,3) );
plot([20 50],[200 500], "r" )
xlabel("Sensor temperature, Tc");
ylabel("Measured temperature ThA");
xgrid();
```

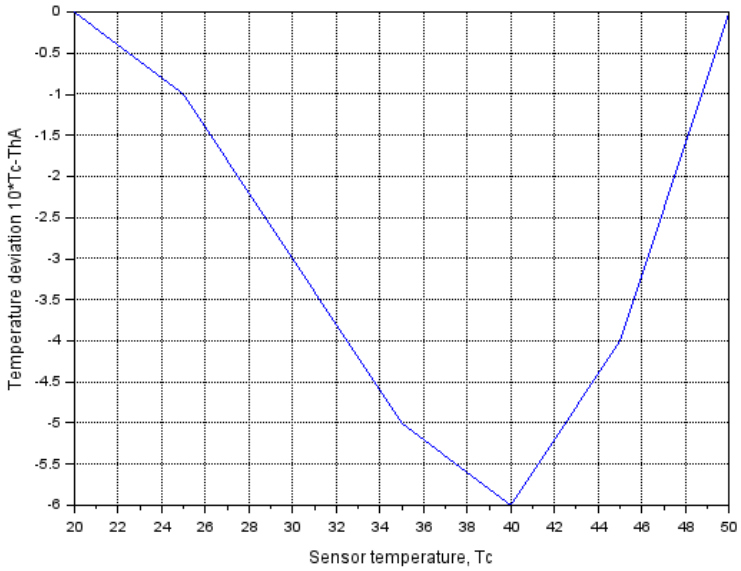


There is a shift of about 1.7 ºC on the measured temperatures. First, we should correct the readings of `Ns0` and `Ns1` at 20C and 50C.

```
T1=20; N1=333; T2=50; N2=637; Tcr=0;
```

Plot the deviation of temperature and measured temperatures.

```
TNTm=[
20 333 200
25 385 251
30 438 303
35 491 355
40 542 406
45 591 454
50 637 500
];
plot(TNTm(:,1), 10*TNTm(:,1)-TNTm(:,3) );
xlabel("Sensor temperature, Tc");
ylabel("Temperature deviation 10*Tc-ThA");
xgrid();
```



Finally, we need to correct `ThA` by adding `5` on the calculated value to get the least deviation around 35 ... 37 C.

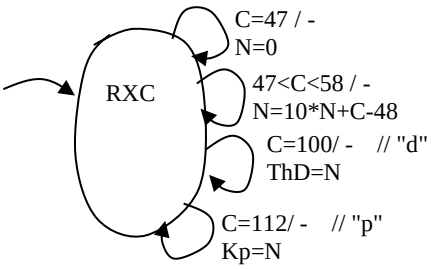
```
T1=20; N1=333; T2=50; N2=637; Tcr=5;
```

After this correction, you must observe that the deviation for 32 ... 45 C decreased to lower than 0.1C.

For the next part, save the project file, and then use save-project-as to rename it **Lab07-03**.

3. UART received character processing (d)

We should implement the following FSM on each received character. For this purpose we may use the interrupt service on received characters.



Arduino serial port works interrupt driven, to store all received characters on a buffer. It delivers each character in fifo order when you read a received character from buffer. An empty buffer returns you -1 as the returned character value.

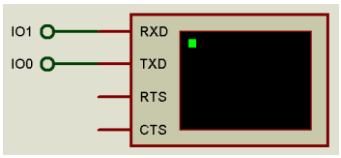
```
//----------declarations-----------
int tss;
int Ns0, Ns1, ThA, ThB, t1m;
int T1, T2, N1, N2, Tcr;
int C, N, Kp, ThD;

...

void FSM_UART(){
     // FSM for received character actions
     C=Serial.read();
     while(C!=-1){
          if(C==47) N=0; //"/"
          if(C>47 && C<58) N=10*N+C-48;
          if(C==100) ThD=N; //"d"
```

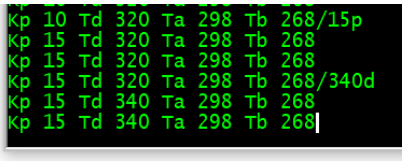
```
          if(C==112) Kp=N;  //"p"
          C=Serial.read(); }
}

//----------setup-------------------
Serial.begin(9600);
pinMode(13,OUTPUT);
tss=0;

analogReference(DEFAULT);
T1=20; N1=333; T2=50; N2=637; Tcr=-5;

Kp=10; ThD=320;

//----------loop--------------------
delay(100);
FSM_flash();
t1m++;
if(t1m >= 60) { // tasks at every minute
     t1m=0;
     ADC_AB();
     UART_printT();
     }
FSM_UART();
```

Notice that FSM_UART() is called at every 0.1s loop time.
Now, in schematics, connect TXD of virtual terminal to the RXD input (IO0) of the Arduino. This connection transmits the characters typed on terminal to Arduino RCX pin.

Test of received character FSM

Set the terminal to *echo typed characters* mode. Send strings /102p and /340d to check the change of displayed Kp and Td values (use keyboard after selecting the terminal window as active window).





After a successful test, save the project file, and save-project-as with the new name Lab07-04 for the next step of incremental test.

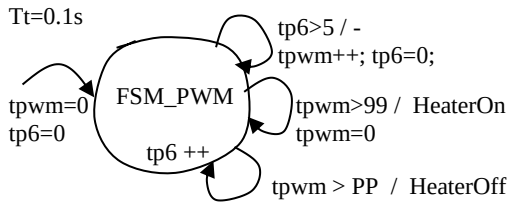
4. Control Task (f)

Control task requires proportional feedback gain Kp.
At every minute it should measure both temperatures *ThA* and *ThB*, and then calculate e = *ThB – ThD.* Next, it should calculate percent-Power, *PP = Kp * e.*
At every 0.1 second it shall trigger the PWM for heater. We may connect the heater relay to IO12 (place there a 220Ohm resistor and an LED as heater indicator.

- generate PWM duty time for 1 minute period,

  This step needs an FSM that is triggered at every 0.1 second for 1 minute time. We have such a counter, t1m, that counts 0.1 seconds in the range (0...599), total 1 minutes. We may obtain an integer tpwm that counts (0...99) by dividing t1m by 6. Or we may count tpwm up at each 6th count of 0.1 second

loop. Duty-on time can be generated by keeping the heater output high while tpwm<*PP* .



Coding needs `int PP, tp6, tpwm, Htr` to be declared globally.

It needs two procedures, i- PControl() procedure, and ii- FSM_PWM() procedure to be declared globally.

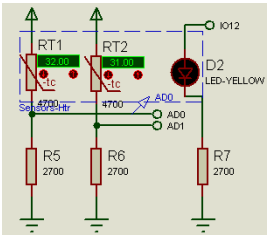
```
void PControl(){
     // PP is calculated at every minute.
     PP =  Kp * (ThD - ThB);
     Serial.print(" PP ");
     Serial.print(  PP  );
     Serial.print(" ");
}

void FSM_PWM(){
// FSM for Controller output PWM at every 0.1s
tp6++;
if(tp6>=6) { tp6=0; tpwm++; }
if(tpwm >= 100) {Htr=1; tpwm=0;}
if(tpwm == PP) {Htr=0;}
// Output to LED
digitalWrite(12, Htr);
if(tp6==0) {
     Serial.print(  tpwm  );
     if(Htr) Serial.print("+"); else Serial.print("-");
     }
}
```

The Htr output pin `pinMode(12,OUTPUT);` and variables `tpwm=0; tp6=0; Htr=0;` should be initialized at setup scope.

In the loop, The PControl() should be called at every minute, and the FSM_PWM() should be called at every 0.1 second.

```
//----------loop--------------------
delay(100);
FSM_flash();
t1m++;
if(t1m > 59) { // tasks at every minute
     t1m=0;
     ADC_AB();
     UART_printT();
     PControl();
     }
FSM_UART();
FSM_PWM();
```

You need also the Htr indicator at IO12. We may use a 220Ohm resistor, and a LED for this purpose on Schematics.



Testing Controller and PWM operation.

Now, run the simulation, set ThB 31C, and transmit /311d (or /312d) from terminal.

```
Kp 10 Td 311 Ta 320 Tb 309 PP 20 90-91-92-93-94-95-96-97-98-99-
Kp 10 Td 311 Ta 320 Tb 309 PP 20 0+1+2+3+4+5+6+7+8+9+
Kp 10 Td 311 Ta 320 Tb 309 PP 20 10+11+12+13+14+15+16+17+18+19+
Kp 10 Td 311 Ta 320 Tb 309 PP 20 20-21-22-23-24-25-26-27-28-29-
Kp 10 Td 311 Ta 320 Tb 309 PP 20 30-31-32-
```

By these settings, ThB=309, ThD=311, and Kp=10 gives PP=Kp(ThD-ThB)=20, and the heater is turned on while tpcm is less than or equal to 20, as seen at the terminal, and also observed with logicprobe at IO12 (Htr output).

After the test, you should save the project, and then use save-project-as to rename it Lab07-05 for the next step of the project.

Note that the alarm and Htr off actions are not yet implemented for ThA and ThB exceeding 37C. Htr-off on (ThA>370) is obtained by an if statement:

```
void FSM_PWM(){
// FSM for Controller output PWM at every 0.1s
tp6++;
if(tp6>=6) { tp6=0; tpwm++; }
if(tpwm >= 100) {Htr=1; tpwm=0;}
if(tpwm == PP) {Htr=0;}
if(ThA>369) Htr=0;
// Output to LED
digitalWrite(12, Htr);
if(tp6==0) {
     Serial.print(  tpwm  );
     if(Htr) Serial.print("+"); else Serial.print("-");
     }
}
```

Test of Heater off action on ThA>37

Run the simulation, and set /2p /. Then increase ThA sensor temperature to 37. While PP=22 and TdA=35.0C PWM generates from count 0 to 22 Htr=1 (shown by +), and then stops heater from count 22 to 99.

```
Kp 2 Td 320 Ta 350 Tb 309 PP 22 90-91-92-93-94-95-96-97-98-99-
Kp 2 Td 320 Ta 350 Tb 309 PP 22 0+1+2+3+4+5+6+7+8+9+
Kp 2 Td 320 Ta 350 Tb 309 PP 22 10+11+12+13+14+15+16+17+18+19+
Kp 2 Td 320 Ta 350 Tb 309 PP 22 20+21+22-23-24-25-26-27-28-29-
Kp 2 Td 320 Ta 350 Tb 309 PP 22 30-31-32-33-34-35-36-37-38-39-
```

After raising ThA=37C, in simulation ThA is calculated ThA=371, exceeding 369. Therefore Htr is set 0 independent of the value of PP.

```
Kp 2 Td 320 Ta 371 Tb 309 PP 22 90-91-92-93-94-95-96-97-98-99-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 0-1-2-3-4-5-6-7-8-9-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 10-11-12-13-14-15-16-17-18-19-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 20-21-22-23-24-25-26-27-28-29-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 30-31-32-33-34-35-36-37-38-39-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 40-41-42-43-44-45-46-47-48-49-
Kp 2 Td 320 Ta 371 Tb 309 PP 22 50-51-52-53-54-55-56-57-58-59-
```

The Htr LED works parallel to the observation on terminal. While ThA=37, Htr LED never starts to light.

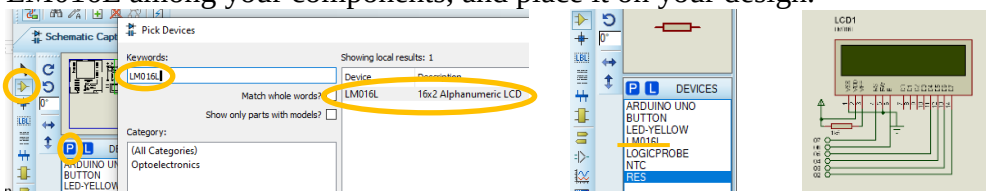
5. *LCD Display (f, g)*

At this step, first comment out the test prints of PWM testing:

```
void FSM_PWM(){
// FSM for Controller output PWM at every 0.1s
tp6++;
if(tp6>=6) { tp6=0; tpwm++; }
if(tpwm >= 100) {Htr=1; tpwm=0;}
if(tpwm == PP) {Htr=0;}
// Output to LED
digitalWrite(12, Htr);
// if(tp6==0) {
//    Serial.print(  tpwm  );
//    if(Htr) Serial.print("+"); else Serial.print("-");
//   }
}
```

Interfacing an LCD display to Proteus Arduino simulation can be obtained by many methods. The simplest method is using the Arduino library <LiquidCrystal.h>. The disadvantage of this method is LCD device becomes interfaced as an external device, and does not appear in the project column. But it has advantage of learning methods of adding external devices to your simulation circuit.

In components mode, click P to get new parts, and search for LM016L. After getting the component close the pick component window. In component mode, select LM016L among your components, and place it on your design.



Use block copy to copy the terminals IO2, IO3, IO4, IO5, IO6 and IO7, and connect them to LCD pins D7, D6, D5, D4, E and RS, as seen in the figure. Connect RW and VSS to 0V (GND), and connect VDD to 5V. For a reasonable contrast setting, place a 1.5kOhm resistor between 5V and VEE pin.

Firmware for LCD display

Arduino firmware needs LiquidCrystal.h library to drive LCD correctly. LCD should be initialized to use 16x2 character mode, and then you may use any lcd method to clear display, set cursor, and print values on the display.

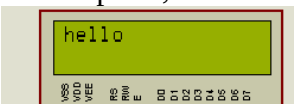
```
//----------declarations----------
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

int tss;
...
```

Arguments (7,6,5,4,3,2) configures the connection of LCD pins (RS,E,D4,D5,D6,D7) to Arduino GPIO pins.
Once `lcd` is declared for the connected pins, it should be initialized for 16x2 characters using `begin(16,2)` method. Thereafter, we may test it by writing a "Hello" at setup code.

```
void setup () {
peripheral_setup();
// TODO: put your setup code here, to run once:
//----------setup------------------
```

```
lcd.begin(16, 2);

Serial.begin(9600);
pinMode(13,OUTPUT);
tss=0;

analogReference(DEFAULT);
T1=20; N1=333; T2=50; N2=637; Tcr=-5;
Kp=10; ThD=320;

pinMode(12,OUTPUT);
tpwm=0; tp6=0; Htr=0;

lcd.print("hello");
}
```

At this point, run the simulation to test the configuration of LCD.



After passing this test, data should be displayed on the LCD. For demo purpose, I write all data without headings in two lines by an LCDdisp() procedure. clear() clears the display, home() sets the cursor to (0,0) position, print() prints strings, integers, and floats, returning the number of printed characters. setCursor(col,row) sets the cursor position.

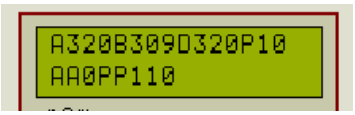
```
LCDdisp(){
     lcd.clear();
     lcd.print("A",ThA,"B",ThB,"D",ThD,"P",Kp,);
     lcd.setCursor(0,1);
     lcd.print("AA",ThA>369,"P",PP);
}
```

And in the loop, update LCD display every minute.

```
//----------loop--------------------
delay(100);
FSM_flash();
t1m++;
if(t1m > 59) { // tasks at every minute
     t1m=0;
     ADC_AB();
     UART_printT();
     PControl();
     LCDdisp();
     }
FSM_UART();
FSM_PWM();
}
```

Testing data displayed on LCD

Run and test how it displays on LCD. In the team project, it is your task to design a Display which can be easily recognized and can be useful for a Nurse or Medical Personel.



You may notice PP is not upper/lower limited, and can be any positive or negative number. It may be good practice to bound PP in the range 1 ... 99.

```
void PControl(){
     // PP is calculated at every minute.
     PP =  Kp * (ThD - ThB);
     if(PP>99) PP=99; if(PP<1) PP=1;
     Serial.print(" PP ");
```

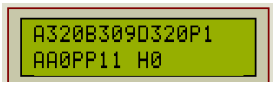
```
     Serial.print(  PP  );
     Serial.print(" ");
}
```

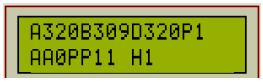
Displaying the heater on/off periods needs to call a print at every tp6==0 pass of the 0.1s loop.

```
if(tp6==0) {
     lcd.setCursor(8,1);
     lcd.print("H"); lcd.print(Htr);}
```

In this case, you may observe the heater status as H0 or H1. For example, if you set Kp=1 (by "/1p") and make temperature ThB=31, you may read on terminal and on LCD display:

```
Kp 1 Td 320 Ta 320 Tb 309 PP 11
Kp 1 Td 320 Ta 320 Tb 309 PP 11
Kp 1 Td 320 Ta 320 Tb 309 PP 11
```





H1 should appear only for 60*0.11 = 6.6 seconds of each 60-second period.

At this point, we should save the project `Lab07-05`, and use save-project-as `Lab07-06` to continue to the next step of the project.

6. UART data transmission

During the previous steps, the UART output has been already set to send the gain Kp, and the temperatures ThA ThB ThD. It additionally transmits percent power too. In the data string only the alarm condition and cover status are missing.

For the cover button, declare integer `SwCov` and `oldSwCov`, connect a switch (*SPST*) at *IO11*, and set it to `pinMode(11,INPUT_PULLUP)`. At every 0.1 second read the switch status by `SwCov=digitalRead(11);` and transmit `Serial.print(" C ");` `Serial.print(SCov);`.

The UART procedure needs the following modification:

- Modify the name UART_printT() to UARTprintAll(),
- Move the Serial.print commands for PP into `UARTprintAll()`:
- Add new Serial.print calls to transmit `AA 0` or `AA 1` depending on alarm condition ThB>369 .
- In the loop, call `UARTprintAll()` instead of `UARTprintT()`. Also move the call after calling `PControl()`.

```
void UART_printAll(){
     Serial.println();
     Serial.print("Kp ");
     Serial.print(Kp);
     Serial.print(" Td ");
     Serial.print(ThD);
     Serial.print(" Ta ");
     Serial.print(ThA);
     Serial.print(" Tb ");
     Serial.print(ThB);
     Serial.print(" PP ");
     Serial.print(  PP  );
     Serial.print(" AA ");
     Serial.print(  ThB>369  );
      SwCov=digitalRead(11);
     Serial.print(" C ");
     Serial.print(SwCov);
}
```

And at every 0.1 second in the loop we shall read the cover status. But we shall transmit it to UART only when the switch changes its status.

```
//----------loop--------------------
delay(100);
FSM_flash();
t1m++;
if(t1m > 59) { // tasks at every minute
   t1m=0;
   ADC_AB();
   PControl();
     UART_printAll();
     LCDdisp();
   }
FSM_UART();
FSM_PWM();
if(tp6==0) {
   lcd.setCursor(8,1);
   lcd.print("H"); lcd.print(Htr);}
oldSwCov=SwCov;
SwCov=digitalRead(11);
if(SwCov != oldSwCov) {
   Serial.print(" C ");
   Serial.print(SwCov);}
```

Testing UART transmit data

Run the simulation to see the terminal output:

It is taken when ThA=32C, ThB=31C, Alarm is not started.

```
Kp 10 Td 320 Ta 381 Tb 309 PP 99 AA 0 AA 0 C 0
```

Next, AA is observed by changing ThB to 37C.
Heater turns off (PP=1), and Alarm starts.

```
Kp 10 Td 320 Ta 320 Tb 309 PP 99 AA 0 AA 0 C 0
Kp 10 Td 320 Ta 320 Tb 371 PP 1 AA 1 AA 1 C 0
```

Cover is opened and closed:

```
Kp 10 Td 320 Ta 320 Tb 371 PP 1 AA 1 AA 1 C 0
Kp 10 Td 320 Ta 320 Tb 371 PP 1 AA 1 AA 1 C 0 C 1 C 0
Kp 10 Td 320 Ta 320 Tb 371 PP 1 AA 1 AA 1 C 0
```

Test results indicate alarm function is successful, cover is detected, control parameter Kp, percent power of heater PP, and all temperatures ThD, ThA, ThB are transmitted with distinctive labes such as "Kp" for proportional control gain, Td for ThD, etc.

REPORT WRİTİNG

As an ABET requirement, to get better mark you shall individually report following issues carefully. Your report should include short summary of applied procedure, plus:

i- Details of calculations for ADC dynamic quantization ratio and you should compare it to the dynamic error range of sensor for the conditions of this application.

ii- Your own drawing for the FSM diagrams for each FSM step, and firmware related to FSM diagrams.

iii- An overall testing section which checks all satisfied requirements in detail.