**ELECTRIC WATER HEATER**



*BY*

*BASSEM EL-BEHAIDY*

|  |  |
| --- | --- |
| Diploma : | Embedded Systems Diploma |
| Group : | Emb – Maadi 503 |
| E-mail : | bassem.elbehaidy@gmail.com |

***Preface***

This project is implemented as the embedded graduation project for the Embedded Diploma course held by AMIT , and hereby , I find it obligatory for me to thank AMIT team for their efforts with us to get the knowledge we were seeking for and I would send my special thanks to Eng. Ahmed El-Gaffarawy who traveled with our team through the whole learning journey starting from C-programming till the very end of Interfacing , and he was always keen to deliver the knowledge in the best way possible.

The project is an electric water heater system, which control the temperature of water within a fixed tolerance range of 5oC from a user programmable set-point.

The system is implemented using ATmega32A as the microcontroller unit in the core of the system, some side electronic circuits for handling inputs to the controller or outputs to the different parts of the system.

The Temperature control is done by:

1. The 220VAC voltage supplied to a heating element (Resistive Heater) is controlled by using a PWM control signal.
2. The 12VDC voltage supplied to Cooling element (Peltier) is controlled by using a PWM control signal.

(Hint: The Peltier is simulated by a variable resistor).

1. The Actual Temperature is read using an LM35 temperature sensor.

This project documentation is divided into two main parts: Hardware & Software.

In the end, I hope you find that the knowledge we gained during the diploma, added value to this project, and managed to fulfill all requirements from design and implementation perspectives.

Finally, my best wishes to all AMIT team, keep the good work.

*Bassem El-Behaidy*

*Wednesday 26th October, 2022*

Contents

1. Hardware Implementation 4
   1. Hardware Design Block Diagram 4
   2. System Circuit Schematics 5
      1. Whole system circuit overview 5
      2. Zero-Cross Detection Circuit 7
      3. Water Heater Control Circuit 8
      4. Water Coolant Control Circuit 9
      5. Dual 7-Segment Display Circuit 10
2. Software Implementation 11
   1. Software Architecture Block Diagram 11
   2. System Flow Chart 12
3. References 16

**1. Hardware Implementation**

The system is implemented basically using an 8-bit ATMEGA32A micro-Controller in the core of it.

***1.1. Hardware Design Block Diagram***

The block diagram of the whole system is shown in Figure 1 below.

ATMEGA32

**Heating Element Circuit**

**Cooling Element Circuit**

**7 Segment Display Circuit**

**Zero Cross Detection Circuit**

**EEPROM**

Figure 1: Hardware Block Diagram

The system consists mainly of:

1. ATMEGA32 micro-Controller with 1KB internal EEPROM.
2. Dual Seven Segment Display with all needed electronic circuit.
3. Indicating LED.
4. Three Push Button switches. (i.e.; Power [ON/OFF], Set-point Adjust [INC/DEC] ).
5. Heating element with needed electronic circuit
6. Zero Crossing Detection Circuit.
7. Cooling element with needed electronic circuit.

**1.2. System Circuit Schematic**

In this section schematics of different circuits would be shown and design criteria explained.

*1.2.1. Whole system circuit overview*

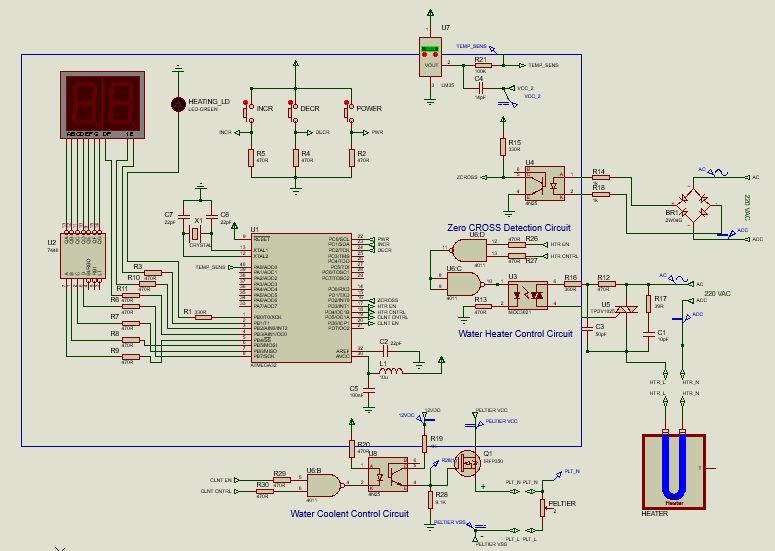


Figure 2: Whole System Schematics as Simulated on Proteus

* System Clock:

The system Clock is determined by external crystal which is 16MHz, connected as shown above with two 22pF capacitors connected to GROUND.

* *Digital Input:*

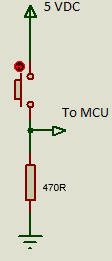


Figure 3:

Digital

Input

* All Digital Inputs are connected through an external PULL-UP resistor as shown in Figure3, and Internal PUD is disabled leaving the pin in Hi-Z (Tri-State).
* In case of any malfunction and the internal PULL-UP is enabled, the resistor value is chosen so that in the LOW state, the current flowing is just above 10.0 mA ( approx. 10.6 mA ) which is still within specs for the MCU.
* In HIGH state the 5VDC Supply connected through a 470 Ohm resistance, allowing a current of approx. 10.6 mA, reducing the current supplied from source as much as possible to minimize the Control system power requirement.
* *Digital Output:*

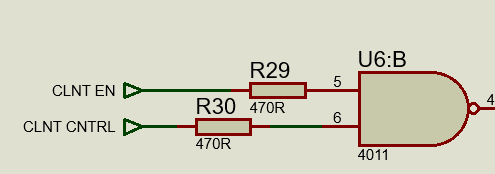


Figure 4: Digital Output

* All Digital Outputs are connected to the next circuit through a resistor whose value restricts the sourced current from MCU to be less than 20mA.
* Almost are Digital Outputs are connected to 470 Ohm resistor as shown in figure 4 , thus restricting sourced current in HIGH state to approx. 10.6mA.
* For the LED output a 330 Ohm restrictor is used which restricts the current to approx. 15.2mA, and this is only to improve the LED illumination and be brighter when ‘ON’, thus improving user experience.
* *Analog Input :*

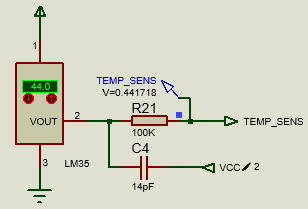


Figure 5: LM35 Analog Input

* The only Analog input we have in system is the LM35 Temperature sensor input.
* LM35 output is (10.0mV/oC) from 2oC to 150oC, having accuracy of 0.5oC, connected to VCC= 5VDC, and VSS connected to GROUND.
* The sensor output is connected to a single ended Channel ‘ADC0’ of the ADC at Pin A0 through a high impedance resistor (100K ohm), along with a capacitor (14pF) connected to ( 2.5 VDC ) as shown in figure 5.

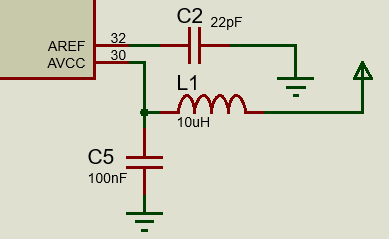


Figure 6: ADC AREF&AVCC Connections

* Using ADC Internal Reference Voltage which is (2.56 VDC) as the ADC Reference Voltage for Conversion. To immune the reference voltage against noise a 22pF Capacitor is connected between AREF pin and GROUND as shown in figure 6.
* To improve conversion accuracy, and reduce the noise level the AVCC pin is connected to digital VCC through an LC network as shown in figure 6.
* According to the above settings and connections, the ADC sensitivity is (0.25oC/STEP).
* Reading the temperature sensor once every 100 mSec, this means that every Second we get 10 different readings and averaging the results reduces the reading error to minimum.
* The final accuracy of the system is +/- 0.5oC, which is the sensor accuracy.
* *Digital I/O Isolation:*
* Whenever the Digital pin (i.e.; working as Input or Output), is interfaced with a circuit using a Supply voltage values different from the MCU Supply voltage, these pins are interfaced through an Opto-coupler to isolate the MCU circuit from the other circuit completely.
* This is implemented in the Zero Cross Detection circuit, and the Cooling Element Control Circuit using 4N25 photo transistor opto-coupler, while in the Heating Element Control circuit a MOC0321 triac driver output opto-coupler, is used to drive the triac controlling the 220VAC supply to the heater.

*1.2.2. Zero-Cross Detection Circuit:*

* The main function of the Zero-Cross detection circuit is to detect the zero voltage crossing in the 220VAC, 50Hz sinusoidal signal.

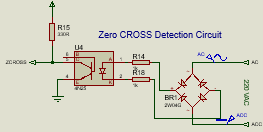


Figure 7: Zero Cross Detection Circuit

* As shown in figure 7, this is done by converting the AC voltage to DC using Diode bridge 2W04G, this DC signal has a frequency double the frequency of the supply AC voltage (i.e.; 2 x 50 = 100Hz).

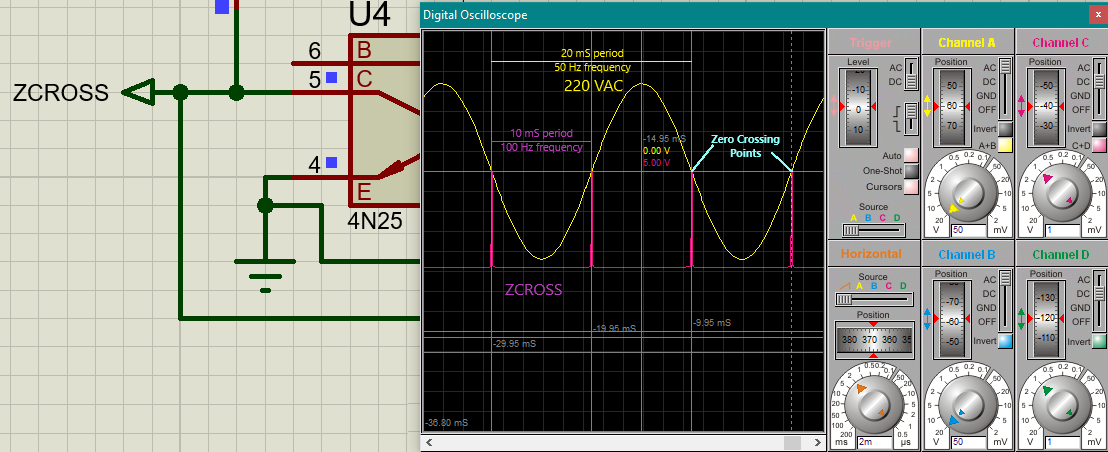


Figure 8: ZCROSS signal against Input AC voltage, showing crossing points, and frequencies

* Applying the above signal through the proper resistive network to 4N25 Opto-Coupler, the input LED diode of 4N25 is always ON except when the voltage drops below the needed value to supply Forward current needed to keep the LED ON. At that stage, the LED turns OFF, causing the output transistor to switch to OFF state as well, thus dis-connecting GROUND from ZCROSS pin, and the ZCROSS pin state becomes HIGH, as shown in Figure 8.
* ZCROSS is connected to INT0 at pin D2, and the adjusting the interrupt sense level to RISING EDGE, allows the MCU to detect every zero crossing in the input Supply voltage.
* Using this ZCROSS signal to synchronize the phase of the PWM signal used to control the heater with it, thus minimizing the transit harmonics resulting from switching effect.
* Adjusting the PWM frequency to 100 Hz, along with the above synchronization, balances the load current as it opens at the same relative phase during both positive half-period, and negative half-period of the AC source. This adds to harmonics improvement, and enhances load power factor.

*1.2.3. Water Heater Control Circuit:*

* The water heater operates on 220VAC-50Hz Supply voltage.

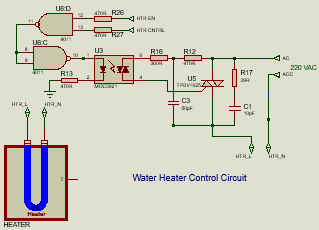
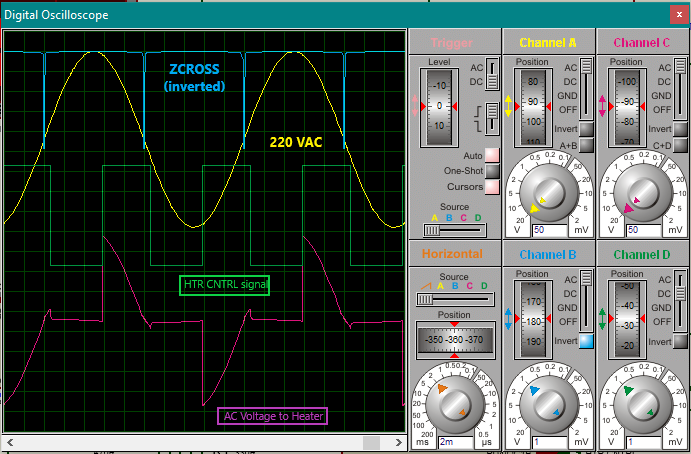


Figure 9: Water Heater Control Circuit

* The N-terminal of the supply is connected directly to one of the heater terminals, while the L-terminal is connected through TPDV1025 triac, whose gate is fired by the signal from photo triac driver opto-coupler MOC3021.
* The input LED in MOC3021 is controlled by MCU through two different control signals:

1. **HTR EN** signal: which acts as ON/OFF switch to the heater power.
2. **HTR CNTRL** signal: which is a PWM signal controlled by a PID control process, thus controlling the amount of electric power supplied to Heater when ***HTR EN*** is ***ON***.

Figure 10: Heater AC voltage control using HTR CNTRL signal



* Both HTR EN & HTR CNTRL signals are ANDED together as shown in figure 9, and the result triggers the MOC3021 input LED.

(Hint: two successive NANDs are used in series, to minimize Hardware system requirements).

* Figure 10 is showing the different signals according to the above explained circuit operation, while HTR EN signal is ON. ZCROSS signal is shown to illustrate synchronization mentioned.

*1.2.4. Water Coolant Control Circuit:*

* The cooling element chosen is TEC1-12706 thermoelectric Peltier, which needs 12VDC supply. Because of different voltages 4N25 opto-coupler is used to isolate different power rails.

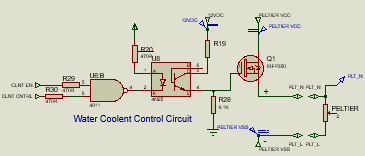


Figure 11: Water Coolant Control Circuit

* The Peltier needs up to 62 Watts for proper operation, that is why high current is supplied through IRFP350 power MOSFET, whose gate is triggered by the output of the 4N25 opto-coupler.
* The input LED of 4N25 opto-coupler is controlled by MCU through two different signals:

1. **CLNT EN**: which acts as an ON/OFF switch of the 12VDC source to Peltier.
2. **CLNT CNTRL**: which is a PWM signal controlled by a PID control process, thus controlling the amount of electric power supplied to Peltier, when **CLNT EN** is **ON**.

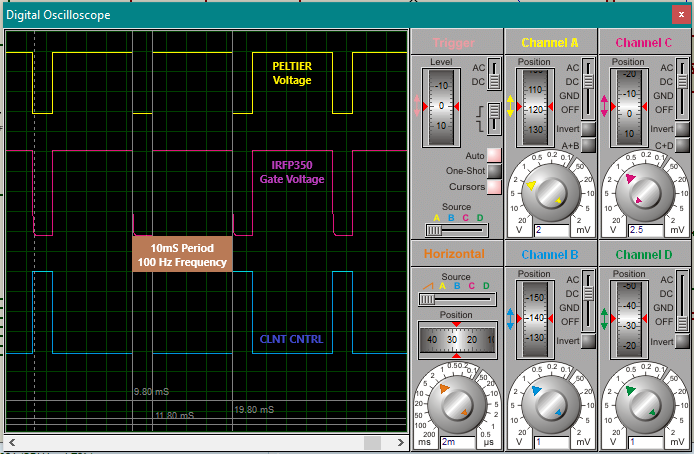


Figure 12: Coolant Control Circuit signals while CLNT EN control signal is ON

* As shown in figure 11, both CLNT EN & CLNT CNTRL are NANDED together, and the result controls input to opto-coupler.
* Figure 12 shows the coolant circuit signals during normal operation as explained above. The power supplied to Peltier is controlled by controlling IRFP350 power MOSFET.

*1.2.5. Dual 7-Segment Display Circuit:*

* As shown in Figure 13 , the UI in this project is a Dual common anode 7-Segment module, which interfaces with the MCU by seven digital output pins:

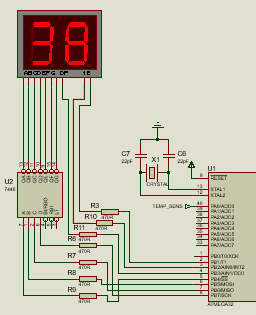


Figure 13: Dual 7-Segment Display Circuit

1. B1 pin controls Module 2 Enable.
2. B2 pin controls Module 1 Enable.
3. B3 pin controls DP Enable which is active HIGH.
4. B4:B7 pins are 4-bit binary representation of decimal digit value, where LSB appear on B4, and MSB appear on B7.

* B4:B7 are connected to ABCD inputs of 7446 Open collector BCD-To-Seven-Segment-Decoder, which converts this to the proper pattern for the 7-segment to display the decimal digit.
* Both Modules are interlaced, each module is switched ON/OFF every 10mSec by MCU through software driver. This technique creates the illusion of simultaneous illumination with no flicker, as the interlacing frequency is 50Hz which is much higher than the eye detection threshold (25 Hz).

**2. Software Implementation**

* The project is implemented based on Abstract Layer Architecture.
* During the diploma, we learnt to build the various MCU peripherals drivers, needed to build the Micro-Controller Layer (MCAL). In this project, those implemented drivers were re-used after making some amendments to the code, so no external libraries is used, and get benefit of some newly added Libraries in the SHARED layer.
* That said, excuse some parts of the code which were modified in a hurry, because of time constrains for delivering the project. I am sure some of those drivers may be implemented more nicely and more efficiently. But still, the readability is good and I tried to add as much comments as possible to make it easy to understand, and be maintainable.
* Project Architecture would be explained in the following sections in more details.

**2.1. Software Architecture Block Diagram**

* The project is built based on four software layers, along with a time management unit as shown in figure 14 below, arrows indicate the accessibility direction between layers.

**APPLICATION LAYER (APP)**

**MICRO-CONTROLLER LAYER (MCAL)**

**SHARED LIBRARIES**

**TIME MANAGEMENT UNIT**

**HARDWARE LAYER (HAL)**

Figure 14: Program Software Abstract Layers Architecture

* The four software layers are (arranged from bottom upwards) :

1. SHARED LIBRARIES: Contains essential libraries, needed by all the above layers.
2. MICRO-CONTROLLER LAYER (MCAL): Have all the libraries for required MCU peripherals, such as DIO, TIMER, EEPROM, etc…
3. HARDWARE LAYER (HAL): Have all the libraries for available hardware, required for implementing the function of the project. i.e.; LM35 sensor, Heater, Coolant, switches, etc…
4. APPLICATION LAYER (APP): Have the main function, and the major controlling functions, along with other libraries to determine system settings and performance requirements.

* TIME MANAGEMENT UNIT (TMU): Builds a tick system operation, allowing tasks to execute according to a time controlled manner, allowing more efficient operation of the system. This unit can access any layer of the software, but can only be accessed by the APP layer. In our system, the OS\_TICK is 10 msec duration.

**2.2. System Flow Chart**

* How the system operates is a matter of perspective. From the point of view of the main() function, we can have the flow chart showed in figure 15, which is totally true, but lacking of details.

STARTT

Retrieve Temperature Set-point from EEPROM

Create Tasks

Priority / Periodicity / Offset / Initial State

SCHEDULER

Figure 15: Project main() function Flow Chart

Initialize the system

* In this perspective the system starts by initializing the various hardware and software parts to initial conditions.
* Next the temperature set-point is retrieved from internal EEPROM.
* Next Tasks are created for the major controlling functions built in the APP layer, assigning for each of them Priority, Periodicity, Offset, and Initial State. At this point, it is important to arrange for functions communication using the proper Global variables, or pointers to share information.
* Finally, pass all the tasks created to the Scheduler, and then all the magic begins, and system operates within the SUPER LOOP of the scheduler.
* To have more detailed perspective, we have to dive a little bit inside the code.

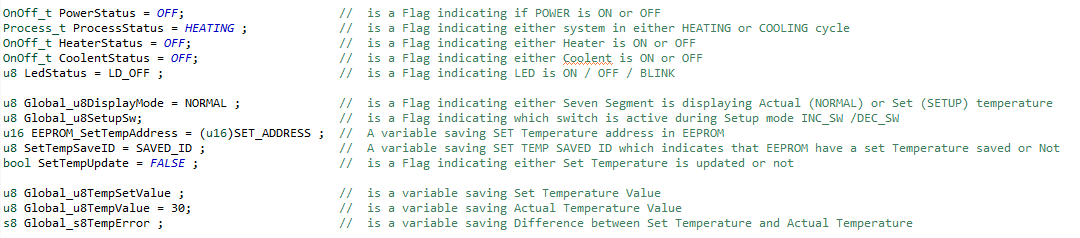


Figure 16: List of Global variables and Flags for inter-Tasks communication

* A list of all GLOBAL variables, and Flags is shown in figure 16. Those variables & flags are message carriers between tasks, as we would see later.

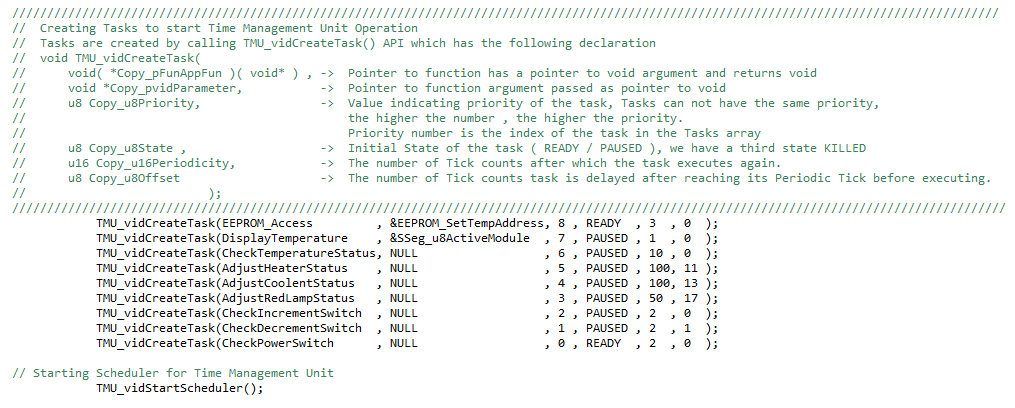


Figure 17: List of Tasks Created showing their initial states

* A list of the all created tasks (9 tasks) is shown in figure 17. Each task is responsible to access a single Hardware item, through the APIs in the library of that hardware.
* A flow chart for the whole system is shown in figure 18, explaining how tasks exchange information to perform.

START

Initialize the system

Create Tasks

**A**

NO

For TASK(8)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

EEPROM\_Access()

- Retrieve/Save Set Temperature from/to EEPROM & save it to Global\_u8TempSetValue

- Set SetTempUpdate = FALSE

- Change TASK(8) state to PAUSE

NO

For TASK(7)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

DisplayTemperature()

Displaying the Temperature Value

( Global\_u8TempValue / Global\_u8TempSetValue )

Figure 18: Flow Chart of Water Electric Heater system

* Each of the Tasks changes its status to PAUSED, if the PowerStatus flag is OFF, except the CheckPowerSwitch() Task, as it is always READY, and functioning.
* Accordingly, when the PowerStatus is OFF, the system reduces its power to minimum, and disconnect both Heater element and coolant element from their supply.

NO

For TASK(5)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

AdjustHeaterStatus()

Enable/Disable Heater According to HeaterStatus variable value, and if ON, passes Global\_s8TempError to calculate HTR\_CNTRL Duty cycle.

NO

For TASK(6)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

CheckTemperatureStatus()

\*Read the LM35 sensor value & save it to Global\_u8TempValue

\*Calculate Global\_s8TempError

\*Set the value of: HeaterStatus, CoolentStatus, LedStatus, & ProcessStatus

NO

For TASK(4)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

AdjustCoolentStatus()

Enable/Disable Coolant According to CoolentStatus variable value, and if ON, passes Global\_s8TempError to calculate CLNT\_CNTRL Duty cycle.

NO

For TASK(3)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

AdjustRedLampStatus()

Adjust Led Status According to LedStatus variable value.

ON / BLINK / OFF

Figure 18: Flow Chart of Water Electric Heater system (Cont’d)

NO

For TASK(2)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

CheckIncrementSwitch()

\*Check if INC\_SW is pressed.

First Press

Global\_u8DisplayMode = SETUP

Global\_u8SetupSw = INC\_SW

Any Successive press

Global\_u8SetTempValue is incremented by SETUP\_STEP till TEMP\_MAX\_LIMIT

\*Set SetTempUpdate = TRUE

NO

For TASK(1)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

CheckDecrementSwitch()

\*Check if DEC\_SW is pressed.

First Press

Global\_u8DisplayMode = SETUP

Global\_u8SetupSw = DEC\_SW

Any Successive press

Global\_u8SetTempValue is decremented by SETUP\_STEP till TEMP\_MIN\_LIMIT

\*Set SetTempUpdate = TRUE

NO

For TASK(0)

State == READY &&

OS\_TICK == K\*Periodicity + Offset

YES

CheckPowerSwitch()

\* Check if PWR\_SW is pressed.

\* Set the value of: PowerStatus variable.

\* if PowerStatus is ON , it resumes all system tasks.

**A**

Figure 18: Flow Chart of Water Electric Heater system (Cont’d)

* The system initially have only TASK(8) and TASK(0) are READY, and all the other tasks are PAUSED,

Accordingly the system retrieves Set Temperature Value saved in EEPROM before even the system is switched ON.

* From the above Flow chart, TASK(8): EEPROM\_Access , operates always for one time when resumed and pause again. It is only resumed when Temperature Set point is changed, to save new value in EEPROM, otherwise it is dormant.
* System turns to SETUP mode if any of the INC\_SW or the DEC\_SW is pressed, and by the end of the SETUP mode, the SetTempUpdate Flag is set, and EEPROM\_Access Task is RESUMED.

**3.References:**

1. ATMEGA32A datasheet.
2. IRFP350 Power MOSFET datasheet.
3. 4N25 photo Coupler datasheet.
4. MOC3021M Randam Phase OptoIsolator Triac Driver datasheet.
5. LM35 Precision Centigrade Temperature Sensor datasheet.
6. TEC1-12703-HB Peltier datasheet.

**\_\_\_\_\_\_**

**END**