FACULTY OF ENGINEERING – AIN SHAMS UNIVERSITY

COMPUTER ENGINEERING AND SOFTWARE SYSTEMS

CSE411 REAL TIME EMBEDDED SYSTEMS Design

Final Project

Team 13

Authors

Kareem

Ahmed

Mohammed Makram

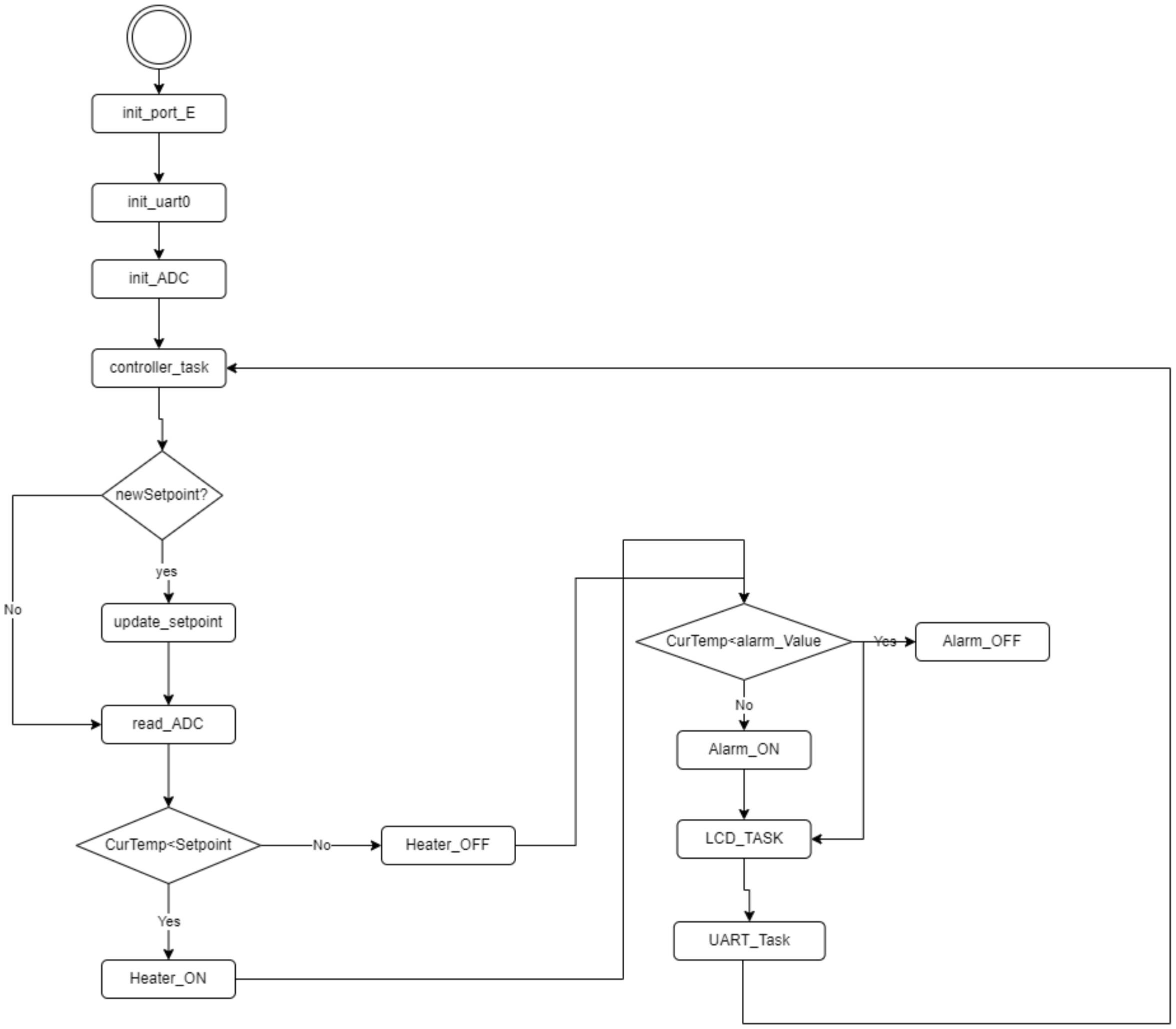
Ahmed

Mohamed

Table of Contents

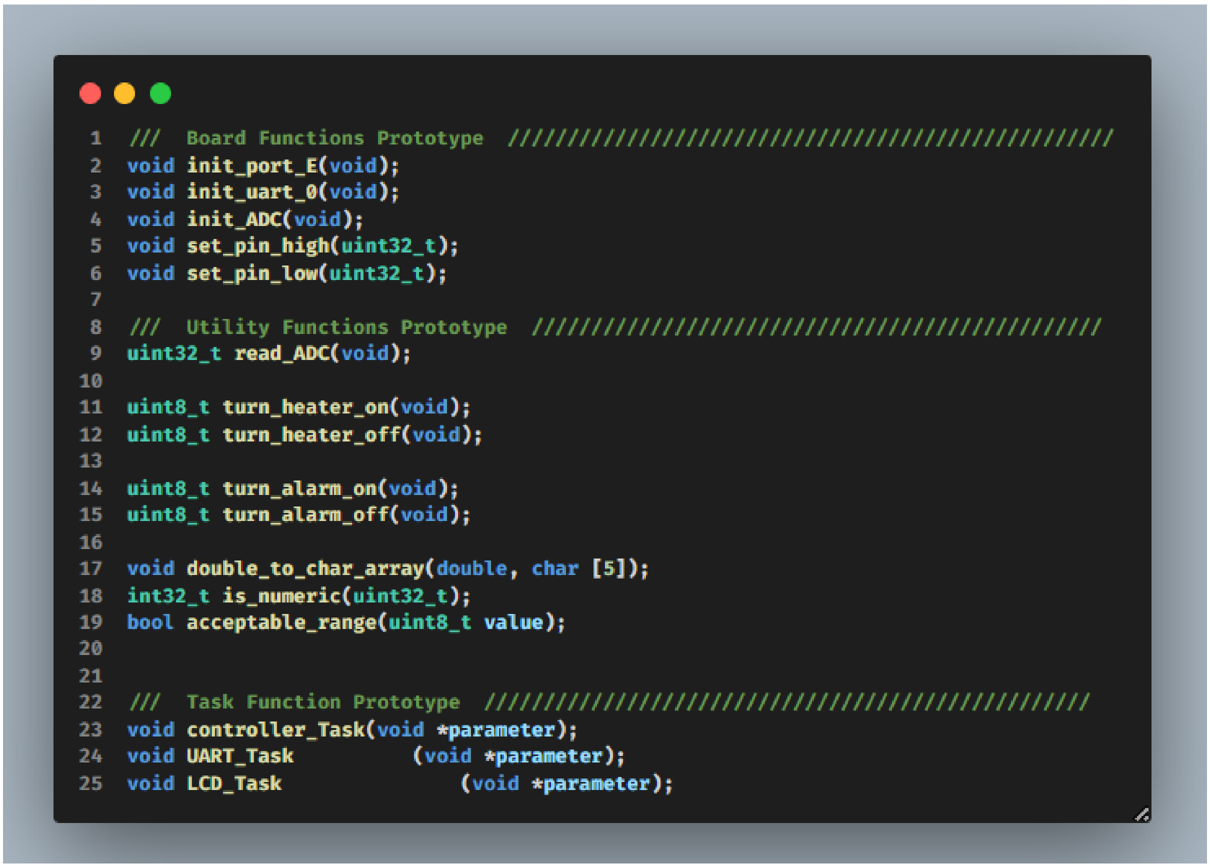
System Flow chart

The following flow chart represents the system in the simplest way.
We focus function callings and main conditions in the system as representing the whole system in a flow chart is impossible, so some implementation details are removed from the flow chart for simplicity and in the next chapter we will discuss our implementation in much more details.



Functions

API functions



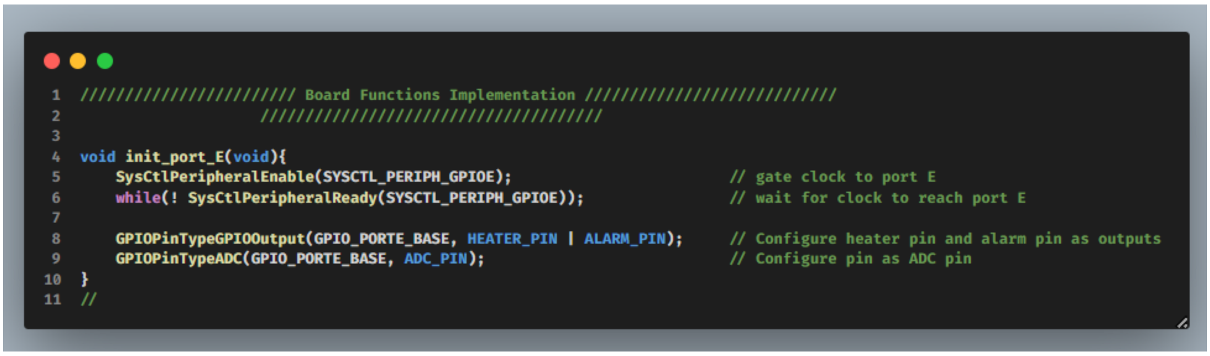
```
///  Board Functions Prototype  //////////////////////////////////////////////////
void init_port_E(void);
void init_uart_0(void);
void init_ADC(void);
void set_pin_high(uint32_t);
void set_pin_low(uint32_t);

///  Utility Functions Prototype  ////////////////////////////////////////////////
uint32_t read_ADC(void);

uint8_t turn_heater_on(void);
uint8_t turn_heater_off(void);

uint8_t turn_alarm_on(void);
uint8_t turn_alarm_off(void);

void double_to_char_array(double, char [5]);
int32_t is_numeric(uint32_t);
bool acceptable_range(uint8_t value);


///  Task Function Prototype  //////////////////////////////////////////////////
void controller_Task(void *parameter);
void UART_Task          (void *parameter);
void LCD_Task               (void *parameter);
```

Functions Descriptions

Define prototypes for all used functions and data structures in our system.

Board functions

`Void init_port_E(void);`

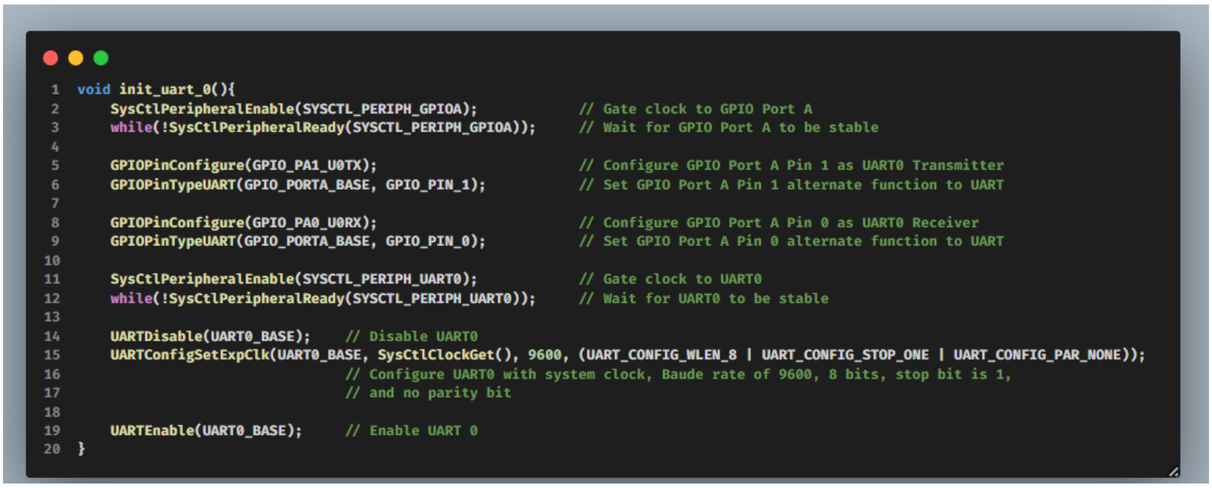


```c
/////////////////////// Board Functions Implementation /////////////////////////
                    ////////////////////////////////////

void init_port_E(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);                     // gate clock to port E
    while(! SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE));             // wait for clock to reach port E

    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, HEATER_PIN | ALARM_PIN);  // Configure heater pin and alarm pin as outputs
    GPIOPinTypeADC(GPIO_PORTE_BASE, ADC_PIN);                        // Configure pin as ADC pin
}
//
```

Function Description:

This function is the init of port E and it starts by gating clock to port E then it waits for clock to reach port E. We configure the heater pin and alarm pin as outputs and then we configure the pin as ADC pin.

`Void init_uart_0_E(void)`

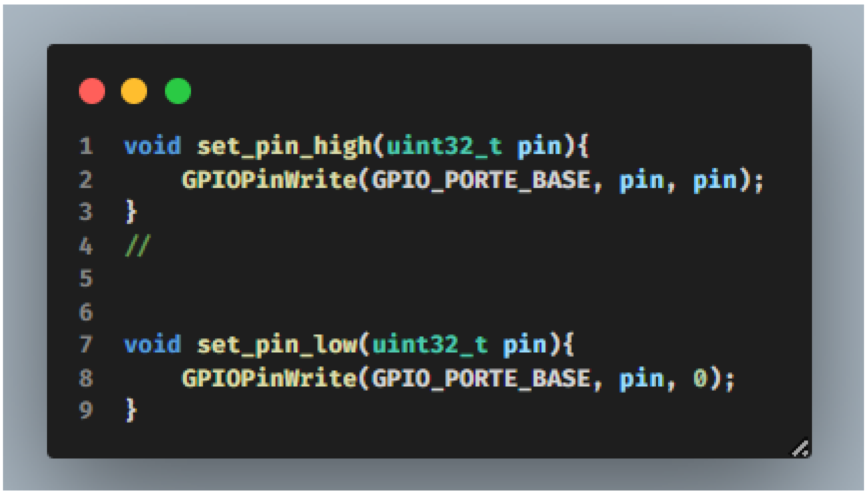


```c
void init_uart_0(){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);           // Gate clock to GPIO Port A
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));    // Wait for GPIO Port A to be stable

    GPIOPinConfigure(GPIO_PA1_U0TX);                       // Configure GPIO Port A Pin 1 as UART0 Transmitter
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_1);          // Set GPIO Port A Pin 1 alternate function to UART

    GPIOPinConfigure(GPIO_PA0_U0RX);                       // Configure GPIO Port A Pin 0 as UART0 Receiver
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0);          // Set GPIO Port A Pin 0 alternate function to UART

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);           // Gate clock to UART0
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0));    // Wait for UART0 to be stable

    UARTDisable(UART0_BASE);    // Disable UART0
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 9600, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
                                // Configure UART0 with system clock, Baude rate of 9600, 8 bits, stop bit is 1,
                                // and no parity bit

    UARTEnable(UART0_BASE);     // Enable UART 0
}
```

Function Description:

This is the init of uart0 and it starts by gating the clock to GPIO port A and then wait for GPIO port A to be stable. We start configuring the GPIO port A pin 1 as UART0 transmitter. We set the GPIO port A pin 1 alternate function to UART. We start configuring port A pin 0 as UART receiver and set its alternate function to UART. We gate clock to UART0 and wait for it to be stable the disable it and configure the UART0 with system clock and the baute rate of 9600, 8 bits, stop bit is 1 and no parity bit. Finally be enable UART A.

Void set_pin_high(unint32_t) and set_pin_low(uint32_t);

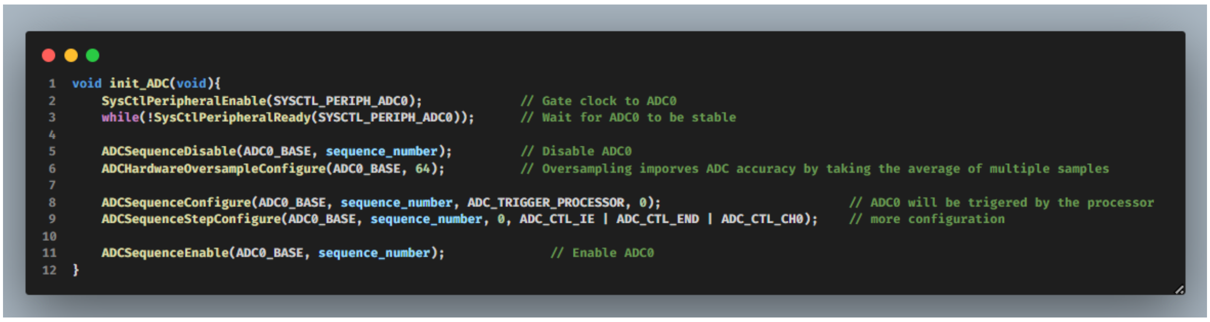


```
void set_pin_high(uint32_t pin){
    GPIOPinWrite(GPIO_PORTE_BASE, pin, pin);
}
//


void set_pin_low(uint32_t pin){
    GPIOPinWrite(GPIO_PORTE_BASE, pin, 0);
}
```

Function Description:

First function we set a given pin high and second function we set a given pin low.

init_ADC(void)



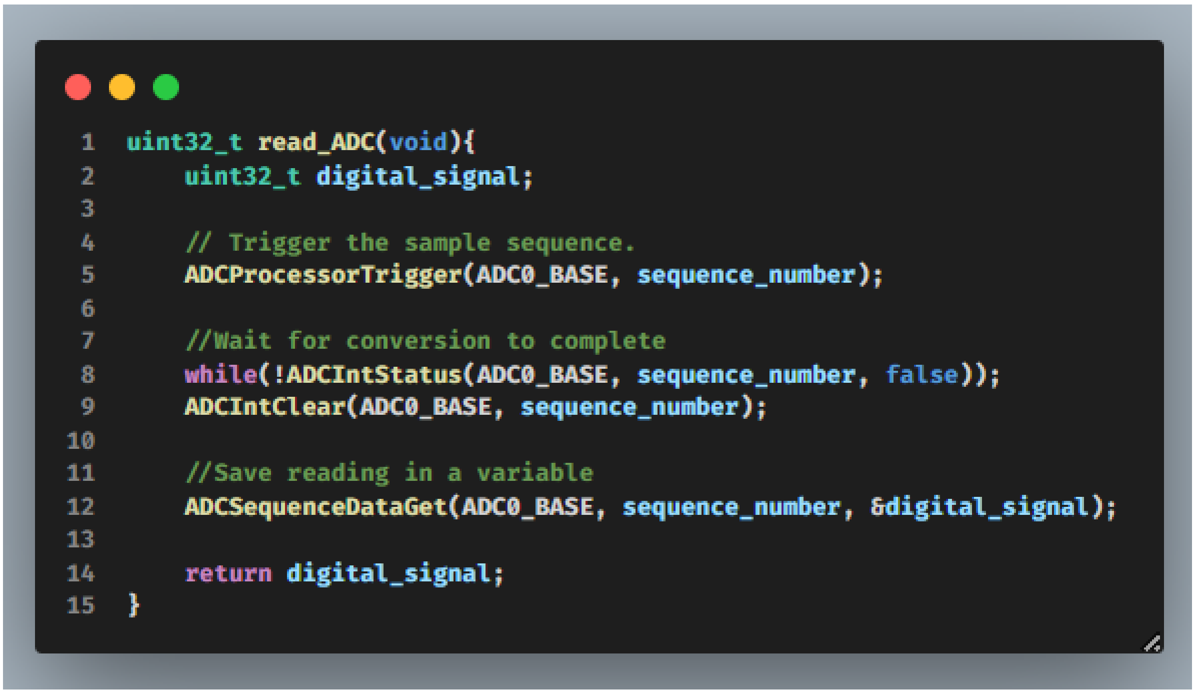
```
void init_ADC(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);             // Gate clock to ADC0
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0));      // Wait for ADC0 to be stable

    ADCSequenceDisable(ADC0_BASE, sequence_number);         // Disable ADC0
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);          // Oversampling imporves ADC accuracy by taking the average of multiple samples

    ADCSequenceConfigure(ADC0_BASE, sequence_number, ADC_TRIGGER_PROCESSOR, 0);                 // ADC0 will be trigered by the processor
    ADCSequenceStepConfigure(ADC0_BASE, sequence_number, 0, ADC_CTL_IE | ADC_CTL_END | ADC_CTL_CH0);    // more configuration

    ADCSequenceEnable(ADC0_BASE, sequence_number);              // Enable ADC0
}
```

Function Description:

This function is the init of the ADC as we start we gate clock to ADC0 then we wait for ADC0 to be stable. Once it is stable we disable the ADC0 and then we do oversampling to improve ADC accuracy by taking the average of multiple samples. ADC0 the will be triggered by the processor, after more configuration we enable ADC0.

Utility functions

`uint32_t read_ADC(void);`

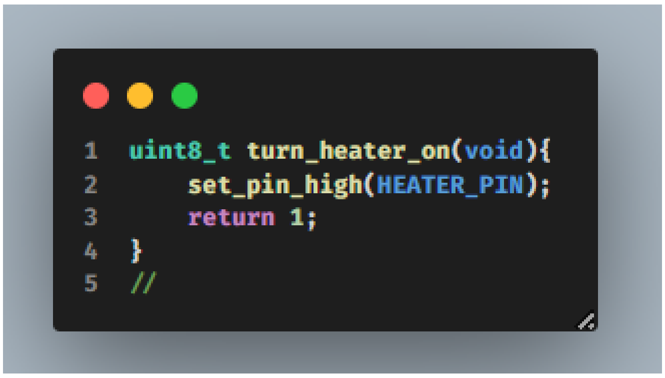


```c
uint32_t read_ADC(void){
    uint32_t digital_signal;

    // Trigger the sample sequence.
    ADCProcessorTrigger(ADC0_BASE, sequence_number);

    //Wait for conversion to complete
    while(!ADCIntStatus(ADC0_BASE, sequence_number, false));
    ADCIntClear(ADC0_BASE, sequence_number);

    //Save reading in a variable
    ADCSequenceDataGet(ADC0_BASE, sequence_number, &digital_signal);

    return digital_signal;
}
```

Function Description:

This function reads ADC and starts by triggering the sample sequence then wait for conversion to complete and then finally save the reading in a variable.

`uint8_t turn_heater_on(void);`

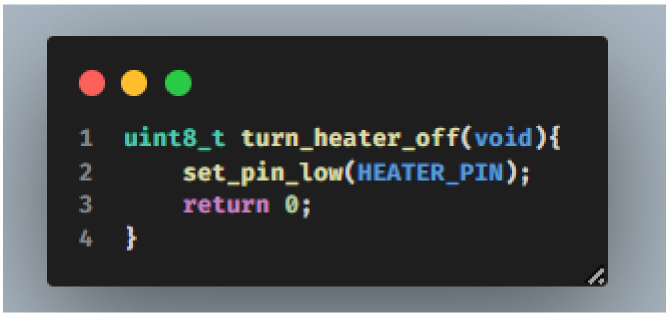


```c
uint8_t turn_heater_on(void){
    set_pin_high(HEATER_PIN);
    return 1;
}
//
```

Function description:

This function turns the heater on by setting the heater pin high.

`uint8_t turn_heater_off(void);`

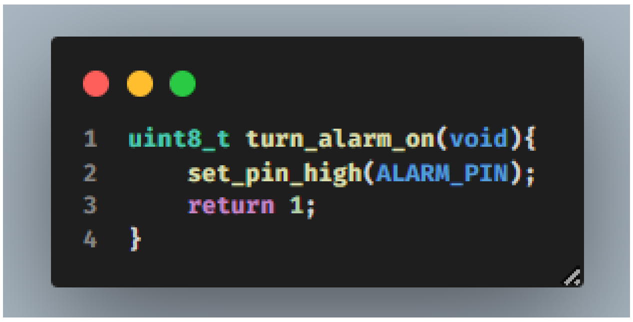


```c
uint8_t turn_heater_off(void){
      set_pin_low(HEATER_PIN);
      return 0;
}
```

Function Description:

This function turns the heater off by setting the heater pin low.

`uint8_t turn_alarm_on(void);`

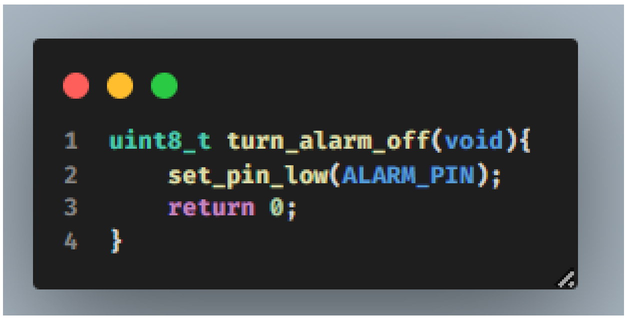


```c
uint8_t turn_alarm_on(void){
      set_pin_high(ALARM_PIN);
      return 1;
}
```

Function Description:

This function turns the alarm on by setting the alarm pin high.

`uint8_t turn_alarm_off(void);`

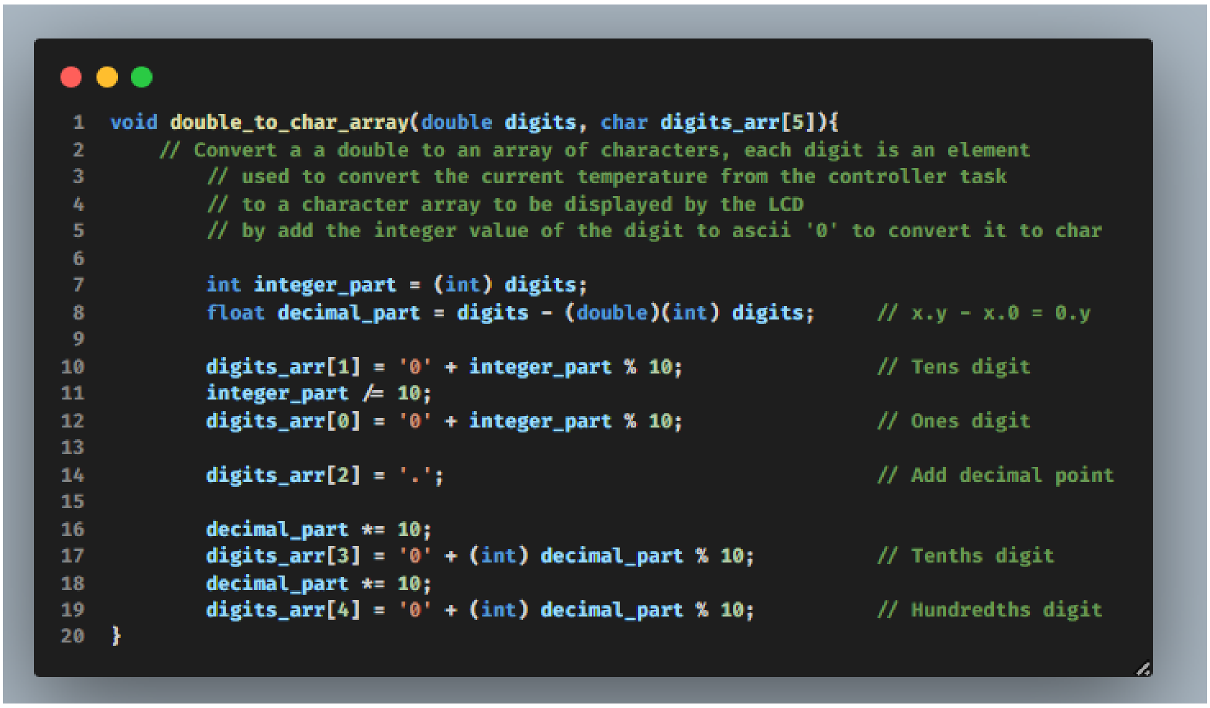


```c
uint8_t turn_alarm_off(void){
      set_pin_low(ALARM_PIN);
      return 0;
}
```

Function Description:

This function turns the alarm off by setting the alarm pin low.

```
void double_to_char_array(double, char [5]);
```

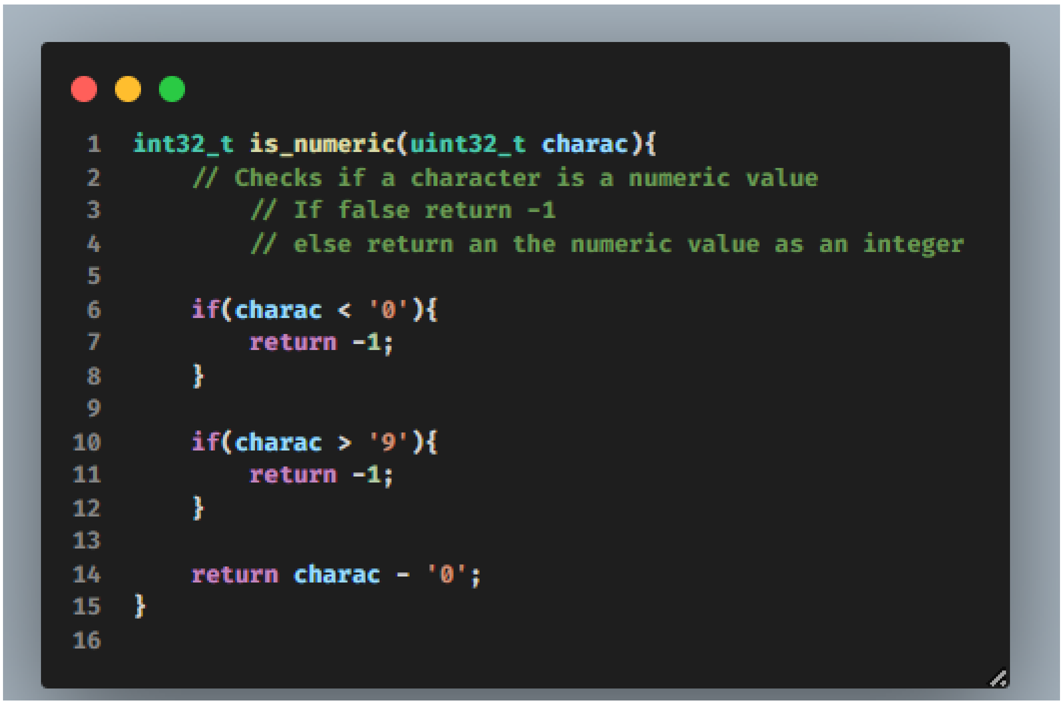


```
void double_to_char_array(double digits, char digits_arr[5]){
    // Convert a a double to an array of characters, each digit is an element
        // used to convert the current temperature from the controller task
        // to a character array to be displayed by the LCD
        // by add the integer value of the digit to ascii '0' to convert it to char

        int integer_part = (int) digits;
        float decimal_part = digits - (double)(int) digits;     // x.y - x.0 = 0.y

        digits_arr[1] = '0' + integer_part % 10;                // Tens digit
        integer_part /= 10;
        digits_arr[0] = '0' + integer_part % 10;                // Ones digit

        digits_arr[2] = '.';                                    // Add decimal point

        decimal_part *= 10;
        digits_arr[3] = '0' + (int) decimal_part % 10;          // Tenths digit
        decimal_part *= 10;
        digits_arr[4] = '0' + (int) decimal_part % 10;          // Hundredths digit
}
```

Function Description:

This function double to char array takes two parameters that is digits and digits array. The function convert a double to an array of characters, each digit is an element used to convert the current temperature from the controller task to a character array to be displayed by the LCD by adding the integer value of the digit to ascii '0' to convert it to char. This function starts by the following x.y - x.0 = 0.y then we tens digit, ones digit, add decimal point, tenths digit, and finally hundredths digit.

`int32_t is_numeric(uint32_t);`

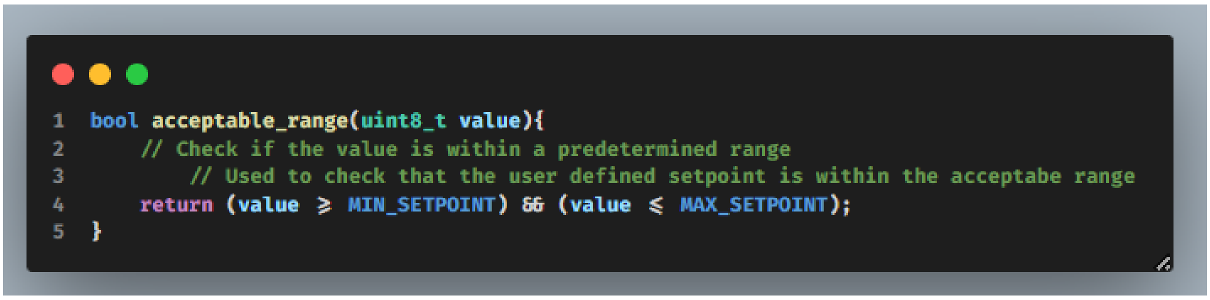


```c
int32_t is_numeric(uint32_t charac){
    // Checks if a character is a numeric value
        // If false return -1
        // else return an the numeric value as an integer

    if(charac < '0'){
        return -1;
    }

    if(charac > '9'){
        return -1;
    }

    return charac - '0';
}

```

Function Description:

This function is numeric checks if a character is a numeric value and if false return -1 else return the numeric value as an integer.

`bool acceptable_range(uint8_t value);`



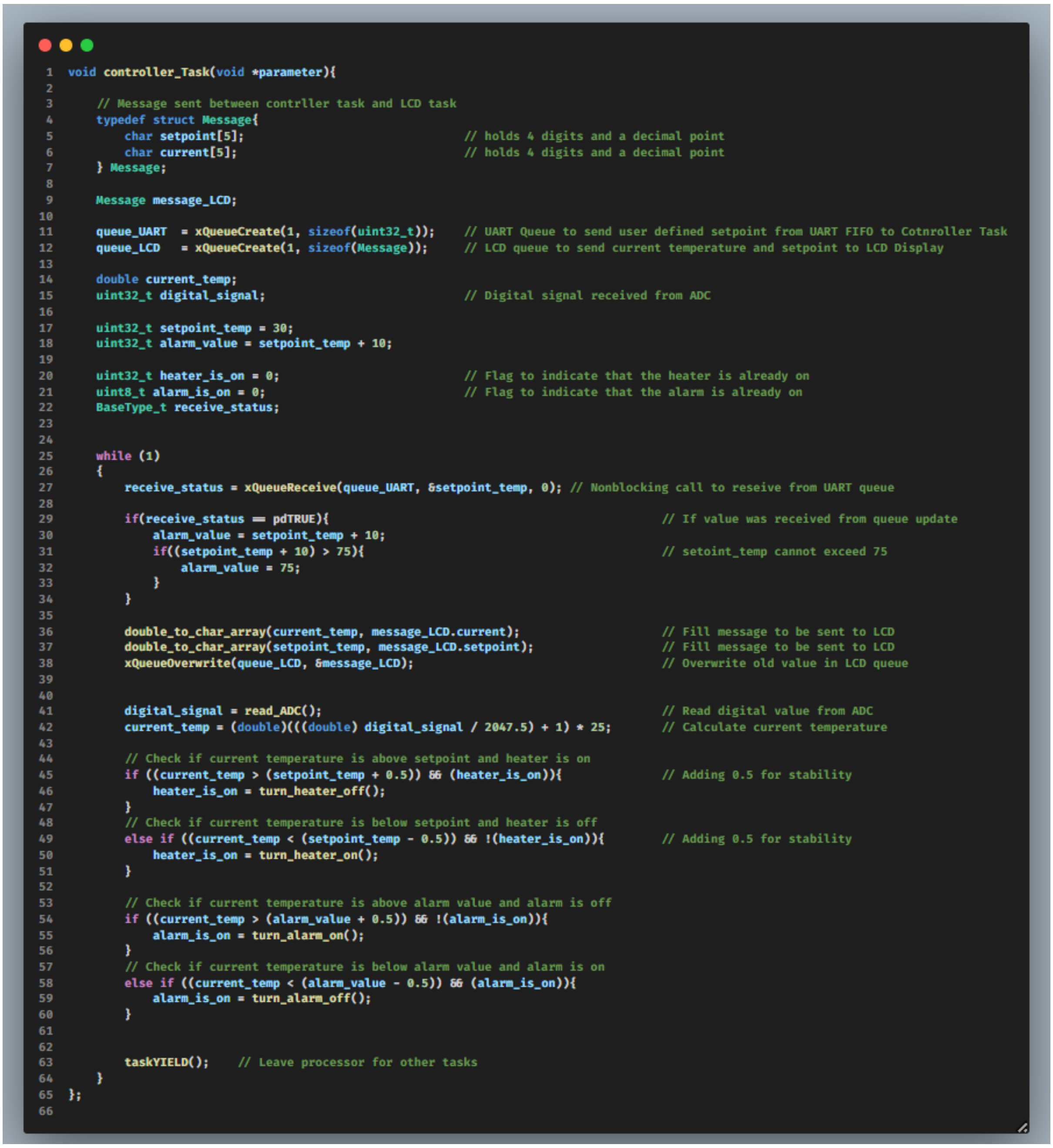
```c
bool acceptable_range(uint8_t value){
    // Check if the value is within a predetermined range
        // Used to check that the user defined setpoint is within the acceptabe range
    return (value >= MIN_SETPOINT) && (value <= MAX_SETPOINT);
}
```

Function Description:

This function acceptable_range checks if the value is within a predetermined range then used to check that the user defined setpoint is within the acceptable range.

Task Functions

`void controller_Task(void *parameter);`

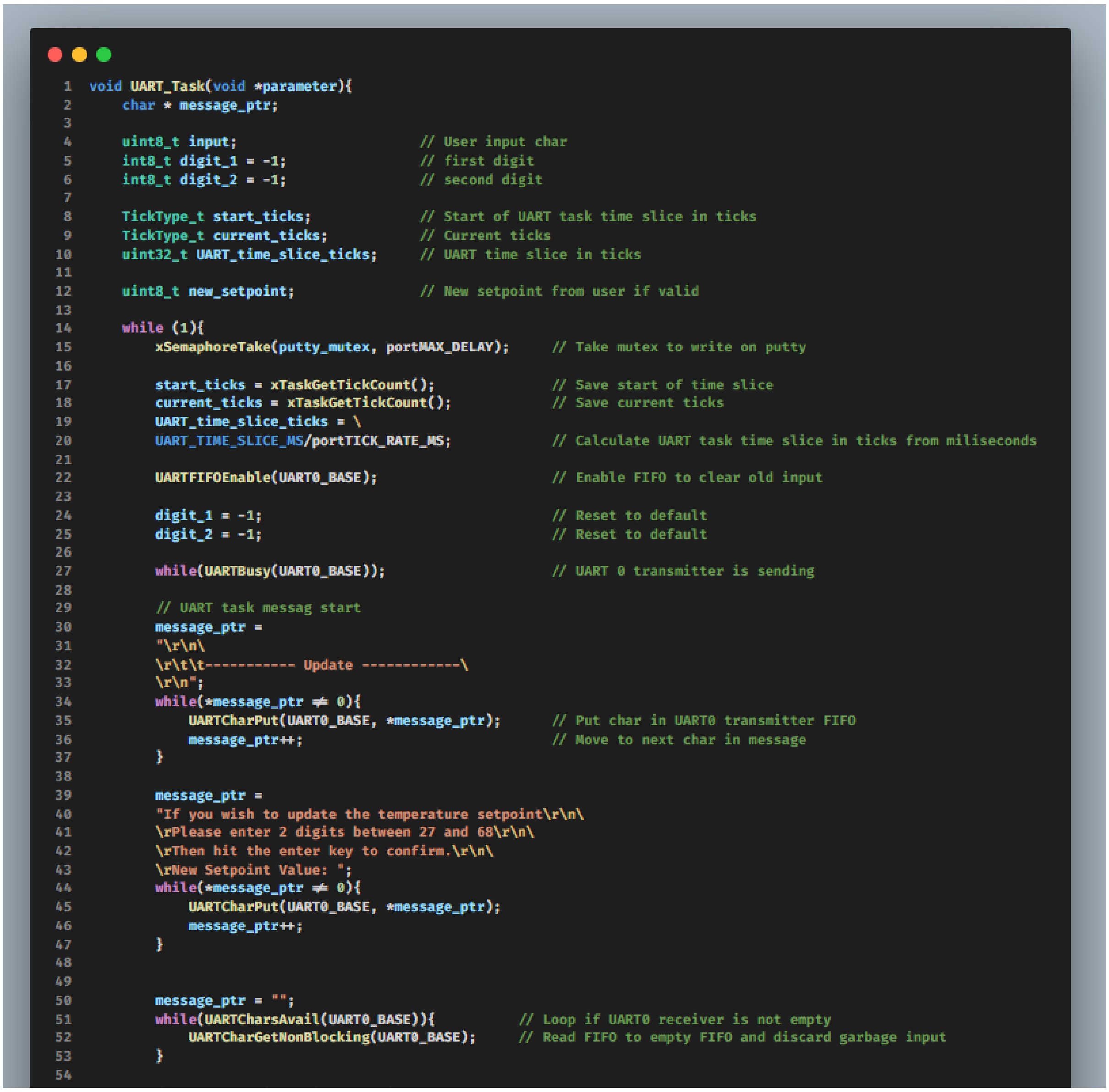


```c
void controller_Task(void *parameter){

    // Message sent between contrller task and LCD task
    typedef struct Message{
        char setpoint[5];                               // holds 4 digits and a decimal point
        char current[5];                                // holds 4 digits and a decimal point
    } Message;

    Message message_LCD;

    queue_UART  = xQueueCreate(1, sizeof(uint32_t));    // UART Queue to send user defined setpoint from UART FIFO to Cotnroller Task
    queue_LCD   = xQueueCreate(1, sizeof(Message));     // LCD queue to send current temperature and setpoint to LCD Display

    double current_temp;
    uint32_t digital_signal;                            // Digital signal received from ADC

    uint32_t setpoint_temp = 30;
    uint32_t alarm_value = setpoint_temp + 10;

    uint32_t heater_is_on = 0;                          // Flag to indicate that the heater is already on
    uint8_t alarm_is_on = 0;                            // Flag to indicate that the alarm is already on
    BaseType_t receive_status;


    while (1)
    {
        receive_status = xQueueReceive(queue_UART, &setpoint_temp, 0); // Nonblocking call to reseive from UART queue

        if(receive_status == pdTRUE){                                               // If value was received from queue update
            alarm_value = setpoint_temp + 10;
            if((setpoint_temp + 10) > 75){                                          // setoint_temp cannot exceed 75
                alarm_value = 75;
            }
        }

        double_to_char_array(current_temp, message_LCD.current);                    // Fill message to be sent to LCD
        double_to_char_array(setpoint_temp, message_LCD.setpoint);                  // Fill message to be sent to LCD
        xQueueOverwrite(queue_LCD, &message_LCD);                                   // Overwrite old value in LCD queue


        digital_signal = read_ADC();                                                // Read digital value from ADC
        current_temp = (double)(((double) digital_signal / 2047.5) + 1) * 25;       // Calculate current temperature

        // Check if current temperature is above setpoint and heater is on
        if ((current_temp > (setpoint_temp + 0.5)) && (heater_is_on)){              // Adding 0.5 for stability
            heater_is_on = turn_heater_off();
        }
        // Check if current temperature is below setpoint and heater is off
        else if ((current_temp < (setpoint_temp - 0.5)) && !(heater_is_on)){        // Adding 0.5 for stability
            heater_is_on = turn_heater_on();
        }

        // Check if current temperature is above alarm value and alarm is off
        if ((current_temp > (alarm_value + 0.5)) && !(alarm_is_on)){
            alarm_is_on = turn_alarm_on();
        }
        // Check if current temperature is below alarm value and alarm is on
        else if ((current_temp < (alarm_value - 0.5)) && (alarm_is_on)){
            alarm_is_on = turn_alarm_off();
        }


        taskYIELD();    // Leave processor for other tasks
    }
};

```

Function Description:

The controller task is the main task in our system, it is responsible for implementing the ON-OFF control system by controlling the heater, it first checks if the user requested a change in the setpoint temperature if so it updates it, it then uses the ADC to read the current temperature of the environment and based on the current value it controls the heater, also it is responsible for alerting the user if the system is overheating, our definition of over heating is if the current temperature exceeds the set point by 10 degrees. If so the system turns a buzzer on to alert the user.

`void UART_Task(void *parameter);`

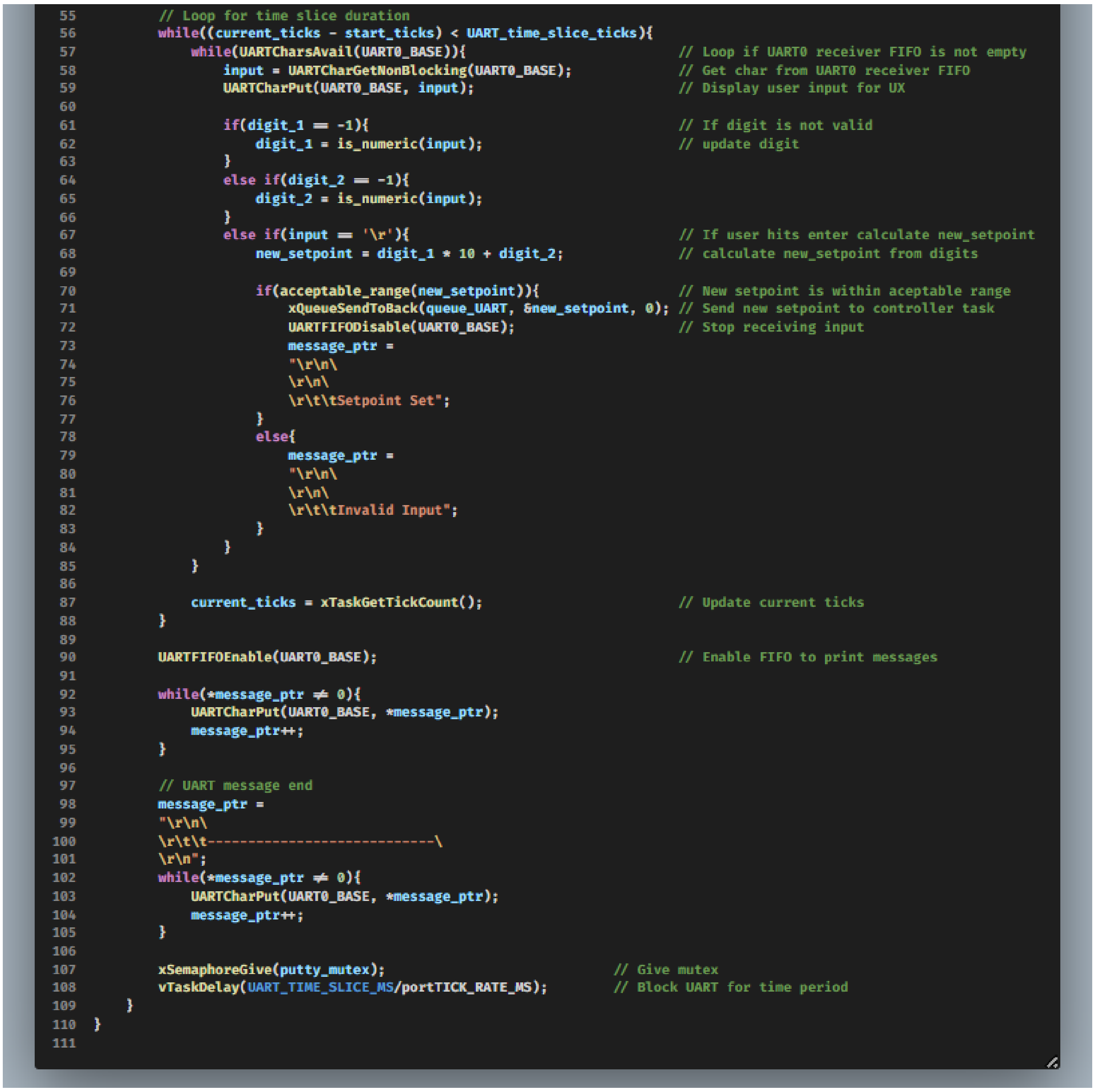


```c
void UART_Task(void *parameter){
    char * message_ptr;

    uint8_t input;                      // User input char
    int8_t digit_1 = -1;                // first digit
    int8_t digit_2 = -1;                // second digit

    TickType_t start_ticks;             // Start of UART task time slice in ticks
    TickType_t current_ticks;           // Current ticks
    uint32_t UART_time_slice_ticks;     // UART time slice in ticks

    uint8_t new_setpoint;               // New setpoint from user if valid

    while (1){
        xSemaphoreTake(putty_mutex, portMAX_DELAY);     // Take mutex to write on putty

        start_ticks = xTaskGetTickCount();              // Save start of time slice
        current_ticks = xTaskGetTickCount();            // Save current ticks
        UART_time_slice_ticks = \
        UART_TIME_SLICE_MS/portTICK_RATE_MS;            // Calculate UART task time slice in ticks from miliseconds

        UARTFIFOEnable(UART0_BASE);                     // Enable FIFO to clear old input

        digit_1 = -1;                                   // Reset to default
        digit_2 = -1;                                   // Reset to default

        while(UARTBusy(UART0_BASE));                    // UART 0 transmitter is sending

        // UART task messag start
        message_ptr =
        "\r\n\
        \r\t\t----------- Update ------------\
        \r\n";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);      // Put char in UART0 transmitter FIFO
            message_ptr++;                              // Move to next char in message
        }

        message_ptr =
        "If you wish to update the temperature setpoint\r\n\
        \rPlease enter 2 digits between 27 and 68\r\n\
        \rThen hit the enter key to confirm.\r\n\
        \rNew Setpoint Value: ";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }


        message_ptr = "";
        while(UARTCharsAvail(UART0_BASE)){          // Loop if UART0 receiver is not empty
            UARTCharGetNonBlocking(UART0_BASE);     // Read FIFO to empty FIFO and discard garbage input
        }

```

—

The responsibilities of this function is

1. Display prompt to the user to input new setpoint
2. Take input from the user as char
3. Convert the input from char format to an integer
4. Pass the user input in integer format to the controller task
5. Repeat procedure every 5 seconds

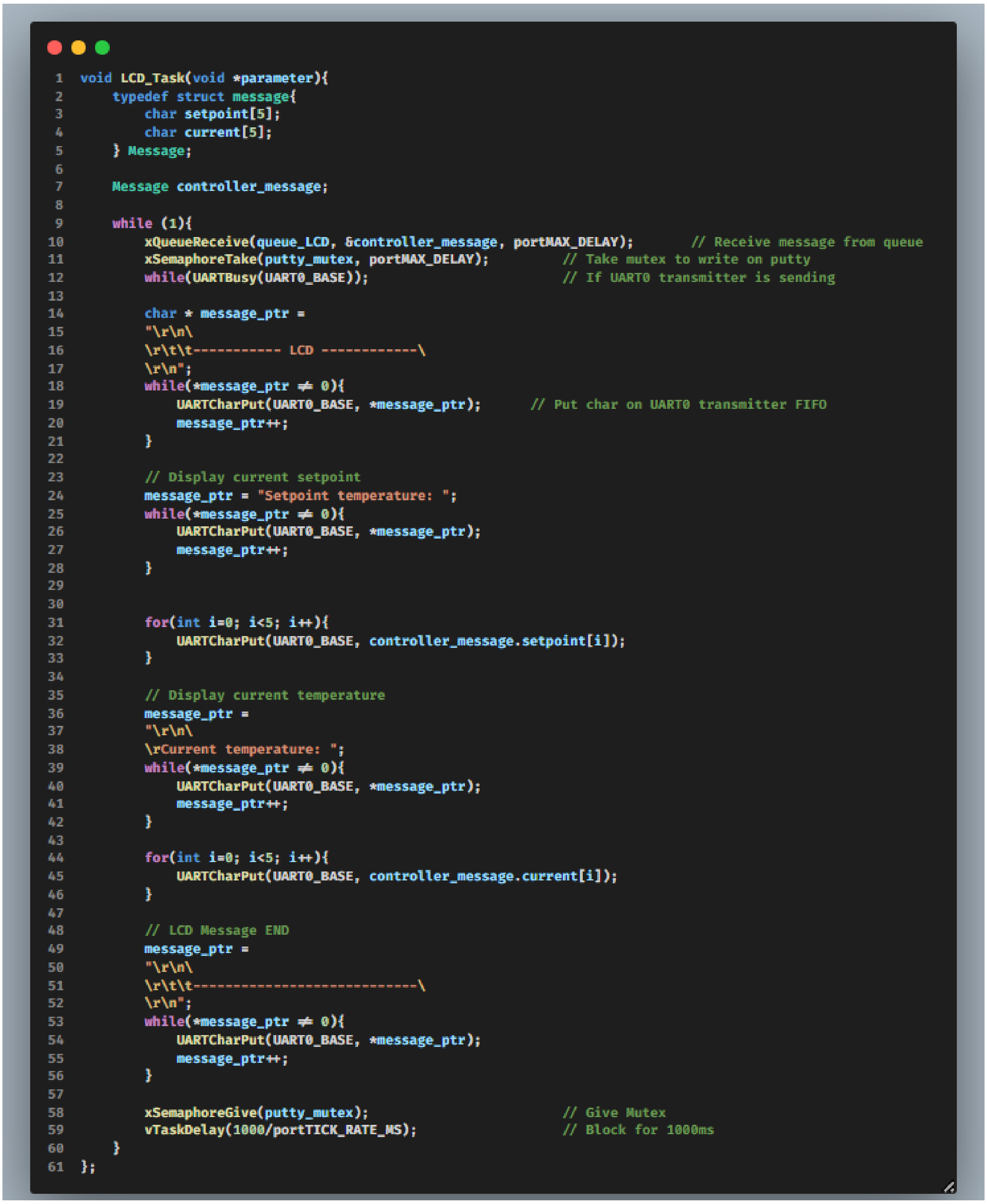


```c
        // Loop for time slice duration
        while((current_ticks - start_ticks) < UART_time_slice_ticks){
            while(UARTCharsAvail(UART0_BASE)){                          // Loop if UART0 receiver FIFO is not empty
                input = UARTCharGetNonBlocking(UART0_BASE);             // Get char from UART0 receiver FIFO
                UARTCharPut(UART0_BASE, input);                         // Display user input for UX

                if(digit_1 == -1){                                      // If digit is not valid
                    digit_1 = is_numeric(input);                        // update digit
                }
                else if(digit_2 == -1){
                    digit_2 = is_numeric(input);
                }
                else if(input == '\r'){                                 // If user hits enter calculate new_setpoint
                    new_setpoint = digit_1 * 10 + digit_2;              // calculate new_setpoint from digits

                    if(acceptable_range(new_setpoint)){                 // New setpoint is within aceptable range
                        xQueueSendToBack(queue_UART, &new_setpoint, 0); // Send new setpoint to controller task
                        UARTFIFODisable(UART0_BASE);                    // Stop receiving input
                        message_ptr =
                        "\r\n\
                        \r\n\
                        \r\t\tSetpoint Set";
                    }
                    else{
                        message_ptr =
                        "\r\n\
                        \r\n\
                        \r\t\tInvalid Input";
                    }
                }
            }

            current_ticks = xTaskGetTickCount();                        // Update current ticks
        }

        UARTFIFOEnable(UART0_BASE);                                     // Enable FIFO to print messages

        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }

        // UART message end
        message_ptr =
        "\r\n\
        \r\t\t---------------------------\
        \r\n";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }

        xSemaphoreGive(putty_mutex);                            // Give mutex
        vTaskDelay(UART_TIME_SLICE_MS/portTICK_RATE_MS);        // Block UART for time period
    }
}
```

Function Description:

This function takes mutex to write on putty and save the start of a time slice and the current takes. It then calculates the UART task time slice in ticks. After that the function enables the FIFO to clear the old output. The function then loops if the UART receiver is not empty and reads the FIFO to make it empty and discard garbage input. and it will update the digit if it is only valid.
If the user hit enter calculate new_setpoint and if it was within the acceptable range, it will be set to controller task and it will stop receiving input. At the end, it will give the mutex and it will block the UART for a time period.

`void LCD_Task(void *parameter);`



```c
void LCD_Task(void *parameter){
    typedef struct message{
        char setpoint[5];
        char current[5];
    } Message;

    Message controller_message;

    while (1){
        xQueueReceive(queue_LCD, &controller_message, portMAX_DELAY);       // Receive message from queue
        xSemaphoreTake(putty_mutex, portMAX_DELAY);         // Take mutex to write on putty
        while(UARTBusy(UART0_BASE));                        // If UART0 transmitter is sending

        char * message_ptr =
        "\r\n\
        \r\t\t----------- LCD ------------\
        \r\n";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);      // Put char on UART0 transmitter FIFO
            message_ptr++;
        }

        // Display current setpoint
        message_ptr = "Setpoint temperature: ";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }


        for(int i=0; i<5; i++){
            UARTCharPut(UART0_BASE, controller_message.setpoint[i]);
        }

        // Display current temperature
        message_ptr =
        "\r\n\
        \rCurrent temperature: ";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }

        for(int i=0; i<5; i++){
            UARTCharPut(UART0_BASE, controller_message.current[i]);
        }

        // LCD Message END
        message_ptr =
        "\r\n\
        \r\t\t----------------------------\
        \r\n";
        while(*message_ptr != 0){
            UARTCharPut(UART0_BASE, *message_ptr);
            message_ptr++;
        }

        xSemaphoreGive(putty_mutex);                        // Give Mutex
        vTaskDelay(1000/portTICK_RATE_MS);                  // Block for 1000ms
    }
};
```

Function Description:

This function simply recieve a message from the queue and make the mutex write on putt in the condition that UART0 transmitter is sending. The function then put a char0 on URAT0 transmitter's FIFO and display the setpoint, current temperature and the mutext is given back and the function block for one second.