DT4012 GR3 Project

Abd Alrahman Atieh, Abdulfattah Morad

January 5, 2023

Abstract

This project involved the development of a greenhouse system with a calendar, temperature recording and presentation, sun tracking and light control, temperature alarms, and a fast mode for testing. The system was implemented using a microcontroller, a keypad, a display, a SysTick timer, a temperature sensor, a photosensor, an LED, and motor-controlled mirrors.

The calendar and time-stamping functions were implemented using the SysTick timer to increment a count of seconds and update the date and time displayed on the screen. The temperature recording function used a temperature sensor to measure the temperature at regular intervals and store the time-stamped data in a linked list. If the memory buffer became full, the oldest data in the linked list was deleted to make room for the new data. The first section is an Introduction. The second section describes the requirements. The last section is a conclusion. The project report contains figures and diagrams on the last pages.

1 Introduction

1.1 Functional Block Diagram

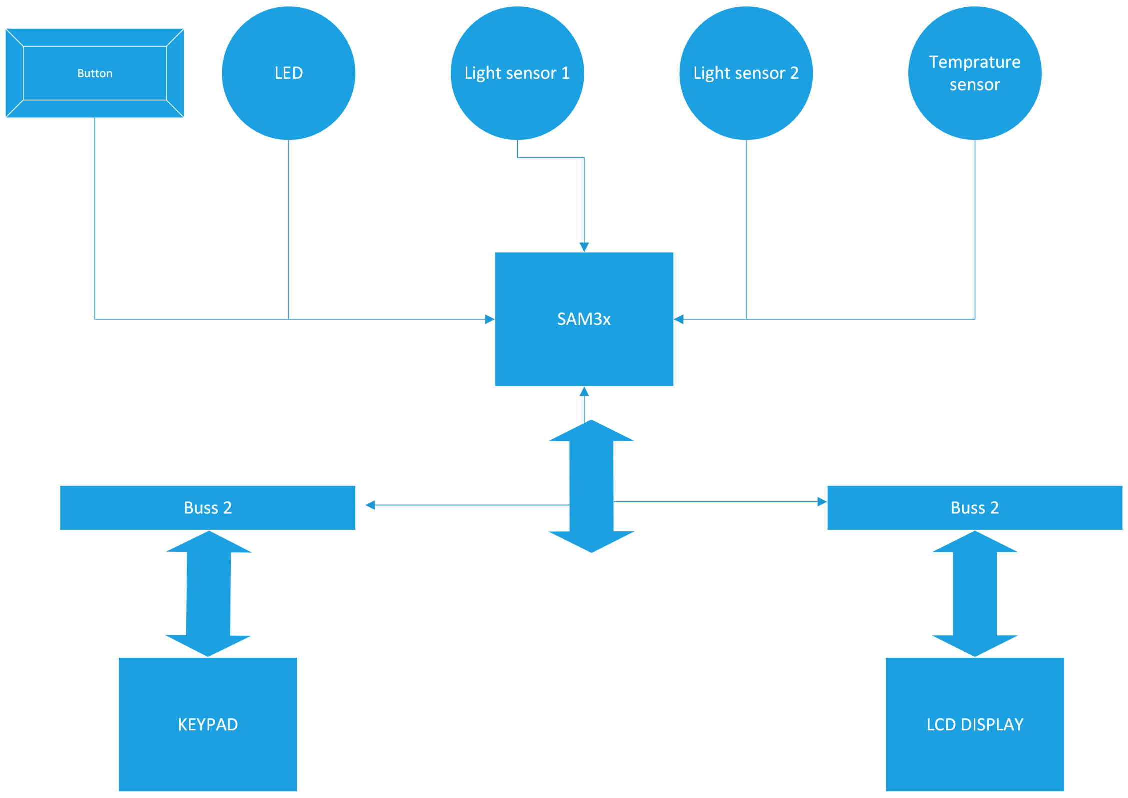


Figure 1: An overview of the system components

1.2 Light sensors

2 light sensors were used to sense the sunlight, The power management controller (PMC) was enabled for the port where the sensors were connected, port A. The PMC was also enabled for the ADC (Analog-Digital Converter). The digital pins were configured as inputs to allow for data to be read from them. The pull-up resistors were also disabled. The clock frequency for the ADC was set using a pre-scaler to divide the main clock frequency (MCK) by a value of (pre-scaler + 1) * 2. Setting the pre-scaler value is done in the mode register (ADC_MR). The pre-scaler value was 6. The specific channels on the ADC where the sensors were connected were then enabled, and the ADC the conversion process was started, the channels are 1 and 2. The ADC was then enabled by starting it in the control register (ADC_CR). The NVIC function for the light sensor was also enabled. To start a measurement with the ADC, the start bit in the control register was set and the status register was read to clear any pending interrupts. The EOC interrupt was then enabled and triggered before the converted data was read. In other words, the interrupt occurs when the conversion is ready. The interrupt calls the ADC handler. In the ADC handler function, it was checked whether the CH1 or CH2 interrupts were enabled and a flag was set to indicate that the data was ready to be read. The interrupt was then turned off until the next data was ready to be read.

1.2.1 Reading the value of the light sensors

When the end of the conversion register is set, the interrupt calls the TC handler. Then The TC Handler sets a flag, indicating the ability to read the values. Then the values are read from the channel data registers 1 and 2.

1.3 Temprature sensor

1.3.1 Configuring the temperature sensor using a time counter

First, enable the PMC for the digital pin and for the Time Counter (TC). In (PMC_PCER) we set the bit 12 for port B, and bit 27 for TC0. The (Timer_Clock1) was selected in the channel mode register (TC_CMR). After selecting the clock, it is enabled and reset in the channel control register (TC_CCR). The timestamp for the falling edge is set to be stored in RA (register A). The timestamp for the rising edge is set to be stored in RB (register B). The digital pin was enabled by enabling the pin's peripheral (PER) control. The pull-up resistor was then disabled.

The temperature sensor was initialized by sending a reset pulse and enabling the TC module to use nested vector interrupts.

1.3.2 Getting a measurement from the temperature sensor using a time counter

A start measurement function was created to start the temperature calculation cycle. The counter was reset to start counting from the beginning and the TC status register (TC_SR) was read to clear any pending interrupts. The TC interrupt enables register (TC_IER) waited for the calculations to be done and then enabled the interrupts. The interrupt was set on the rising edge. If the rising edge of the temperature was detected, it entered the TC0_Handler function automatically. If the rising edge was detected in the handler function, it meant that enough information had been collected to calculate the temperature. A flag called "Global Rising Edge Temp" was set to 1 to indicate that the temperature could be calculated.

An is_measurement_ready function checked the Global Rising Edge Temp flag to see if enough information had been collected to calculate the temperature. If the flag was set to 1, the measurement was ready. If the flag was set to 0, the measurement was not yet ready.

$$\text{Temperature in celsius } = \frac{\text{time for rising edge} - \text{time for falling edge}}{42 * 5} - 273.15$$

A get_measurement function used the calc_temp(RA) falling edge and calc_temp(RB) rising edge values to calculate the temperature using a formula involving the difference between the two edges and a constant value. The Global Rising Edge Temp flag was then set to 0 to indicate that the measurement had been taken and the process could start again.

1.4 Servo

A servo motor was used to track the sun's movement, the glass roof. A digital pin (pin 62 on port B) was connected to a pulse width modulation (PWM) signal by first enabling the power management controller (PMC) for the PWM and port B. The digital pin was then disabled so that the PWM could control it, rather than the digital pin being controlled as a general-purpose input/output (GPIO) pin. This was done by using the pin's alternate function mode register (ABMR) to select the peripheral B.

The specific PWM channel (channel 1) that was connected to the servo was enabled, and the prescaler value (MCK/32) was selected to determine the clock frequency for the PWM. The conversion period (cycle) was then set and the duty cycle (e.g. 1500) was enabled.

The position of the servo is updated using the Channel Duty Cycle Update Register. The the value given for that register is calculated with this formula:

$$\text{value} = \frac{700}{3}(\text{angle}/10) + 1300$$

1.5 LCD display

The pins on the LCD that controls the display were enabled by the flag pins LCD function. Pins 6 and 12 to 20 on port C. These pins may have been general-purpose input/output (GPIO) pins.

1.5.1 Read the status of the display

read_status_display() return a byte of data, called status word. The function is_display_ok() checks if the display was functioning properly, is_ display_ok(), returned a value of 1 if the display was ok.

```
// functions defined
// sends a byte command from. Commands are predefined
void write_command_2_display(unsigned char command)

// sends a 1 byte of data to display
void write_data_2_display(unsigned char data)
void init_display()
```

The databus from the LCD to the SAM3X chip was set as an input by the make databus as input function, while the databus from the SAM3X chip to the LCD was set as an output by the make databus as an output function. The values on the DB0 to DB7 pins on the LCD (the databus) were read by the data_bus function and returned as a byte of data (8 bits). The values on the databus were cleared by the clear databus function. A byte of data (1 byte, 8 bits) was sent to the DB0 to DB7 pins on the LCD by the set data to databus function.

1.6 Keypad

1.6.1 Configuring the keypad

First, enable the PMC for ports C and D. The columns use the bits 7-9 in port C. The rows use 2-5 in port C. The keypad is connected through a bus. It can be activated and deactivated through a digital pin controlled with bit 2 in port D.

1.6.2 Read a value from the keypad

The polling technique is used here because keys are not connected to separate pins, they share pins. To know which button is pressed:

1. Activate the bus connecting the keypad with the processor.
2. Set all columns as output, enable columns as output.
3. Give all columns value 1.
4. Go through all columns, one column at a time.
   (a) Give column value 0.
   (b) Read all rows. And see which row has the value 0.
   (c) If there is no row with the value 0, it means there is not any pressed key.

    (d) Then the pressed key value is computed as row number * 3 + column number +1;
    (e) Give column value 1.

5. Set all columns as input, enable columns as input.

1.6.3 Implement keys as click buttons, i.e. read the value once the button is released

```
int button_released()
{
    int value = pressed_key();
    // wait until a key is pressed.
    while (value == 0) { value = pressed_key(); }
    // wait until the key is released.
    while (pressed_key() != 0);
    return value;
}
```

1.7 LED and Button

The button and LED are both in port d. The PMC for port D was enabled. Both led and the button was enabled by setting the bits PIO_PER, peripheral enable register. The pull-up resistors were disabled for both buttons and led by setting the bits in PIO_PUDR, pull-up disables register. The LED is set as an output by setting the bit in PIO_OER, output enable register. The button is set as an input by setting the bit in PIO_OER, output enable register. The interrupt was enabled for the button, in order to be used in later functions.

2 GUI

2.1 4 pages for text and 1 page for graphics

The home addresses for the 4 pages of text: [0, 16x30, 16x30 x 2, 16x30 x 3], because the text display has a place for 16x30 characters. Pages 1,2,3 and 4 are assigned for the menu, Greenhouse info, and Temperature table and edit the options, respectively. To choose a page, we use the command SET_GRAPHIC_HOME_ADDRESS from the LCD's datasheet. The main functions for printing on the display are setting the address pointer and then sending the data to the display. The set_address_pointer gives you access to every location on the display. First, it is told which page to print on, then the x and y coordinates. The formula for calculating the address of a given x and y coordinates is as follows:

$$address = base + (y * 30) + x$$

where 30 is the number of columns for the display. We can see that the address will consist of 2 bytes. The LCD can read 1 byte at a time, hence we send first the low byte, then we logically shift to the right 8 bits to send the more significant byte.

```
void set_address_pointer(unsigned short address, char col, char row)
{ if (col > 30 || row > 16)
        return;
    address += (row * 30) + col;
    run_command_op2(SET_ADDRESS_POINTER, address & 0xff,
                               (address & 0xff00) >> 8);}
```

After deciding the location for printing, we either print a word or a character. The printing of a character function is simply by sending the data of the character and using one of these commands DATA_WRITE_AND_INCREMENT_ADP or DATA_WRITE_AND_NON_VARIABLE_ADP. The printing of a word function uses the functionality of printing a character but using a for loop to print a char array instead.

To choose the graph page, we need to change the display mode. The function display_mode helps us change the mode very easily. For example, display_mode(1,1,1,0) means, set the cursor to ON, the blinking of the cursor to ON, text mode to ON, and graph mode to OFF.

```
void display_mode(char cursor, char blinking_cursor, char text, char graph)
{ write_command_2_display(DISPLAY_MODE  |
                            (graph << 3)  |
                            (text << 2)   |
                            (cursor << 1) | (blinking_cursor));}
```

2.2 Keylistner

The interaction with the system. The keypad is the main component to talk with the processor. We have a limited set of keys to use, 12 keys. The system has a lot of functionality (e.g. change the time and date, change the temperature limits...). We introduce (modes) to solve that problem. Modes are a set of flags, each function that needs the use of keys gets a mode assigned to it. The system has a total of 12 modes. So, each time a key is pressed the Keylistner will check the mode. The mode tells the Keylistner what to do with the key, i.e which function needs this key at the moment. A snippet of function Keylistner:

```
void key_listner(int key)
{
...
    switch (MODE)
    {
...
        case MODE_PAGE_3_OPTIONS_VIEW:
            controll_page_3_options();
            break;

        case MODE_TEMP_LOW_EDIT:
            edit_temprature_lower();
            break;

        case MODE_TEMP_HIGH_EDIT:
            edit_temprature_higher();
            break;
...}}
```

2.3 Graph Mode

We introduce canvas, a char array of length 3840. The canvas is usefull for saving the information locally, the same information which is plotted on the LCD. To set a bit in canvas from a given x and y location, we do this:

$$canvas\left[30y + \frac{x}{8}\right] \mid = (1 << (7 - (x\%8)))$$

Note, the OR operations are used to keep the old data written. Then the whole canvas is displayed on the display, by opening the stream (AUTO DATA READ) then sending each byte. After the whole array of bytes is sent, close the stream.

3 Project Requirements

3.1 Timestamp: (Req 1)

The calendar is based on one SysTick timer, which increments every millie seconds. The calendar has seconds, minutes, hours, days, months, and years counter. The seconds counter updates when the millie seconds counter becomes equal to or more than 1000. The minutes counter updates when the seconds counter becomes 60. and so on... We introduce a one-byte variable TICK. TICK has 8 bits, so we can assign 8 flags in TICK, bit 0: seconds, bit1: minutes, bit2: hours, bit3: days. By checking each bit, we can see if a new (second/minute/hour/day) has started.

Updating the time and date by the user. edit_date() and edit_time() are implemented to read values from the user. Then counters are updated accordingly. The current time is displayed on page 1, i.e by clicking 1 from the menu page.

```
void edit_date()
{
if (MODE != MODE_DATE_EDIT)
    return;
// the options page
unsigned short local_address = PAGE_3_ADDRESS;
print_info("Date", 4, local_address);
set_address_pointer(local_address, 0, ROW);
print_word("Date: dd/mm/yyyy", 17);
int key = 0;
// to store the input from the user
char date[10] = "00/00/0000";
for (char i = 0; i < 10; i++)
{
    if (i == 2 || i == 5) // jump over "/"
        continue;

    key = read_digit(i + 6, ROW);
    if (key == -1)
    {
        errase_info(4, PAGE_3_ADDRESS);
        display_mode(0, 0, 1, 0);
        MODE = MODE_PAGE_3_OPTIONS_VIEW;
        return;
    }
    date[i] = key + '0';
}
update_date(date);
errase_info(4, PAGE_3_ADDRESS);
display_mode(0, 0, 1, 0); // cursor blinking = off
MODE = MODE_PAGE_3_OPTIONS_VIEW;}
```

3.2 Periodic Temperature Recording (reqs 2 and 9)

The user has the ability to set how many samples in a minute should the system record, N, which can be done on page 3 from the menu. After setting the N value, or just using the default N = 10, the time between each record is $60000/N$. 60000 ms = 60 s = 1 minute. Then after getting those N samples, the mean value is calculated, let us call it average1. Both the N samples and the average1 are stored in their own linked-lists. Then we calculate the new minimum, maximum and average value of the day. Notice, The minimum, maximum, and average of the day consist of the averages in the minutes.

The first flow chart in Figure **??** describes the procedures which are done on a daily bases. Incrementing a day counter tells us which day of the week it is. It is incremented using this, $day = (day\%7) + 1$, it is like a loop, never goes over 7. We also, store the date. These dates are used for the table diagram when presenting the records. A very important step is displaying the actual data. This is done once a day. At last, some flags are set or cleared, they control the flow of the program.

The second flow chart in Figure 2. It is simply a for loop that works using time. n is incremented each 60000/N if a measurement is done. Then after getting all N samples we can compute the average of that minute.

The third flow chart, the bottom left one, is in Figure 2. This one is for controlling the temperature sensor. If the ms counter has reached the value 60000/N, start the measurement cycle for the sensor. And clear the ms counter. Then after starting the temperature sensor, wait until the temperature sensor calls an interrupt handler. Then the handler will set a flag to tell us the measurement is ready. At last, get the measurement and store it in the linked-list.

Memory is a very important thing. We need to check if we have enough place in memory before storing new values. The 4:th flow chart in Figure 2, the bottom right one, describes the function of checking if memory is available to us. First, create the element which you want to store. If the element returned to be a null value means that element has not been created due to lack of memory. This problem is solved by removing the old data of that LinkedList, and freeing that memory place. We may do that repeatedly. Eventually, we will have a free space in memory to store our new records.

The configuration of the temperature sensor is paragraph 1.3.

3.3 Display of Periodic Temperature Recordings (reqs 3 and 11)

From the menu, press 2 you will see a table that consists of 7 rows. Each row represents a day of the week. The first column has the date. The second column has the minimum of the day and its timestamp. The third column has the max. The last column has the average. see figure 6. We are only printing days that have passed, so if today is Thursday we print for Monday. It is only on Mondays you will see a full week printed, the last week.

The plot is accessible from the menu by pressing 5. There you can see three graphs. The first one represents the maximum for each day. The second graph represents the average under a day. The third one represents the minimum. Each graph fits for 7 days. The same technique is followed here. You only see a full plot on Mondays, i.e the last week's data. The timestamps for each day are printed on page 2.

3.4 The Greenhouse: (reqs 4 and 10)

The greenhouse system requires 3 important components. One is the light sensors. Two is the servo. Three, is the LED, i.e. the light system. The configurations for these components are in paragraphs 1.2, 1.4 and 1.7, respectively. Another component is the shades. These shades can be simulated with printing a text message on scren. The flow chart in figure 3 describes the functions of the greenhouse. First, the hours in sunlight-, the hours in the light system- and hours in darkness counter are defined. The next step, clear those counters at the start of each day. The system is designed so that the greenhouse gets a controlled amount of light and darkness. precisely, 16 hours of light and 8 hours of darkness. The light system works as a supplementary sun, on winter days when the sun has a short life span. On the other hand, the shades block the sun if the hours on the sunlight counter have reached the limit, which is 16 hours.

Getting the angle of the sun. It is a challenging requirement. First, we compute the difference between the two light sensors. The value of a light sensor can be interpreted as the distance from the light source to the sensor. If the result of subtracting a-b is ≈ 0 the distances from the light source to the sensor are almost equal, thus the light source is 90 degrees above the sensor. If the result of a-b is negative. The light source is to the left of the sensors. If the result of a-b is positive. The light source is to the right of the sensors. where a and b are the two light sensors.

```
int get_angle()
{
if (A_minus_B < -80)
    return 180;
if (A_minus_B < -70)
    return 160;
if (A_minus_B < -60)
    return 155;
.
.
.
if (A_minus_B < 110)
    return 20;
if (A_minus_B < 130)
    return 10;
return 0;
}
```

3.5 Maintain the temperature between upper and lower limits (req 5)

The user can set limits, where the user wants to keep the temperature. By default, these limits are 20 and 25. When the temperature goes outside that limit, the alarm goes off. The alarm is a message printed on page 1 5. The alarm can turn off by pressing the button (the push button).

3.6 requirements 6, 12 and 16, Test modes

3.6.1 Fast Mode

Test modes are easy to implement. To simulate 30 minutes in a second. instead of increasing the seconds counter every 1000 ms by one, increase the seconds counter by 2 each 1 ms. In this way, we can simulate 30 minutes in one second.

```
void update_counters_speed_mode()
{
if (MILLIESECONDS_COUNTER >= 1)
{
    SECONDS_COUNTER += 2;
    MILLIESECONDS_COUNTER = 0;
    TICK |= 1; // set the seconds flag
}
...
}
```

3.6.2 Prerecorded data for the graph

We define a function that fills the min, max and avg arrays manually. Then plot these arrays.

```
void generate_test_data()
{min[0] = 10;
...
    max[0] = 20;
...
    avg[0] = 16;
...
    dates[0] = (struct date){20, 11, 2022};
...
    min_timestamp[0] = (struct time){00, 12, 20};
...
    max_timestamp[0] = (struct time){00, 35, 12};
 ...}
```

Plotting is done by this function: See figure **??**

```
void plot_days(double min[], double max[], double avg[], char day);
```

3.6.3 Unit testing: (req 16)

We have a separate project for this requirement. In that project, there is the file containing all the test cases for all the components.

3.7 Login system: (req 15)

3.7.1 MD5 hash

The MD5 hashing algorithmpap. You give a char array and it gives you back a hash with 16 byte. The steps for using the algorithm:

1. Add padding so the message is a multiple of 512 bits. M[] the padded message. In our case, the padding's shape is static. Because we are always hashing a 4-digit word.

   ```
   void padding(unsigned char *msg, unsigned int *m){
   unsigned char
       padded_msg[64] = {0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                   msg[3], msg[2], msg[1], msg[0]};

   for (char i = 0; i < 16; i++)
   {
       // copying date from a 64*1 bytes array to 16*4 bytes array
       m[i] |= (padded_msg[(i * 4) + 0] << 8 * 3);
       m[i] |= (padded_msg[(i * 4) + 1] << 8 * 2);
       m[i] |= (padded_msg[(i * 4) + 2] << 8 * 1);
       m[i] |= (padded_msg[(i * 4) + 3] << 8 * 0);}}
   ```

2. Initialize for unsigned integers (A, B, C, and D) with some initial value.

   - A = 0x01234567
   - B = 0x89abcdef
   - C = 0xfedcba98

- D = 0x76543210

3. Define auxiliary functions.

$$\begin{aligned} f(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) \\ g(x,y,z) &= (x \wedge y) \vee (y \wedge \neg z) \\ h(x,y,z) &= x \oplus y \oplus z \\ i(x,y,z) &= y \oplus (x \wedge \neg z) \end{aligned} \tag{1}$$

4. Define T[i]; i-th: $(int)4294967296 * |\sin(i)|$.

5. Copy the element of M[] to a new array X[].

6. Save A, B, C, and D in AA, BB, CC, and DD respectively.

7. Repeat these operations with different k, i, and s values. And the order of a, b, c, and d is different for each iteration.

   - $a = b + ((a + f(b,c,d) + X[k] + T[i]) <<< s)$
   - $a = b + ((a + g(b,c,d) + X[k] + T[i]) <<< s)$
   - $a = b + ((a + h(b,c,d) + X[k] + T[i]) <<< s)$
   - $a = b + ((a + i(b,c,d) + X[k] + T[i]) <<< s)$

8. The final hash value is:

$$\begin{aligned} A &= A + AA \\ B &= B + BB \\ C &= C + CC \\ D &= D + DD \end{aligned} \tag{2}$$

   The end results for a message of "0000": 0x2ae6'ac96, 0x1f48'8d69, 0x914c'f122, 0x1946'a80d.

3.7.2 Login System

The login system contains ONLY one user, the admin. The hash of the password has been saved. So when login in, the user is asked to input a 4-digit password. Then the program will generate the hash for the given password. If the hashes are the same, the login is successful. The user can change the default password. But first, the user needs to be logged in. This system knows if the user is logged in by checking a flag. This is not enough in a real-world application.

4 Conclusion

We were limited to a fixed size of memory, which was small. The linkedlists took the largest space in memory because the temperature was saved each minute. The accuracy of the temperature sensor is not that important. We do not need to use a double to store the temperature, but we use a float instead. That is because it is more than enough to get 2 places after the decimal point. By using floats instead of doubles we save 4 bytes for each linkedlist.

References

Md5 hash. URL http://practicalcryptography.com/hashes/md5-hash/.

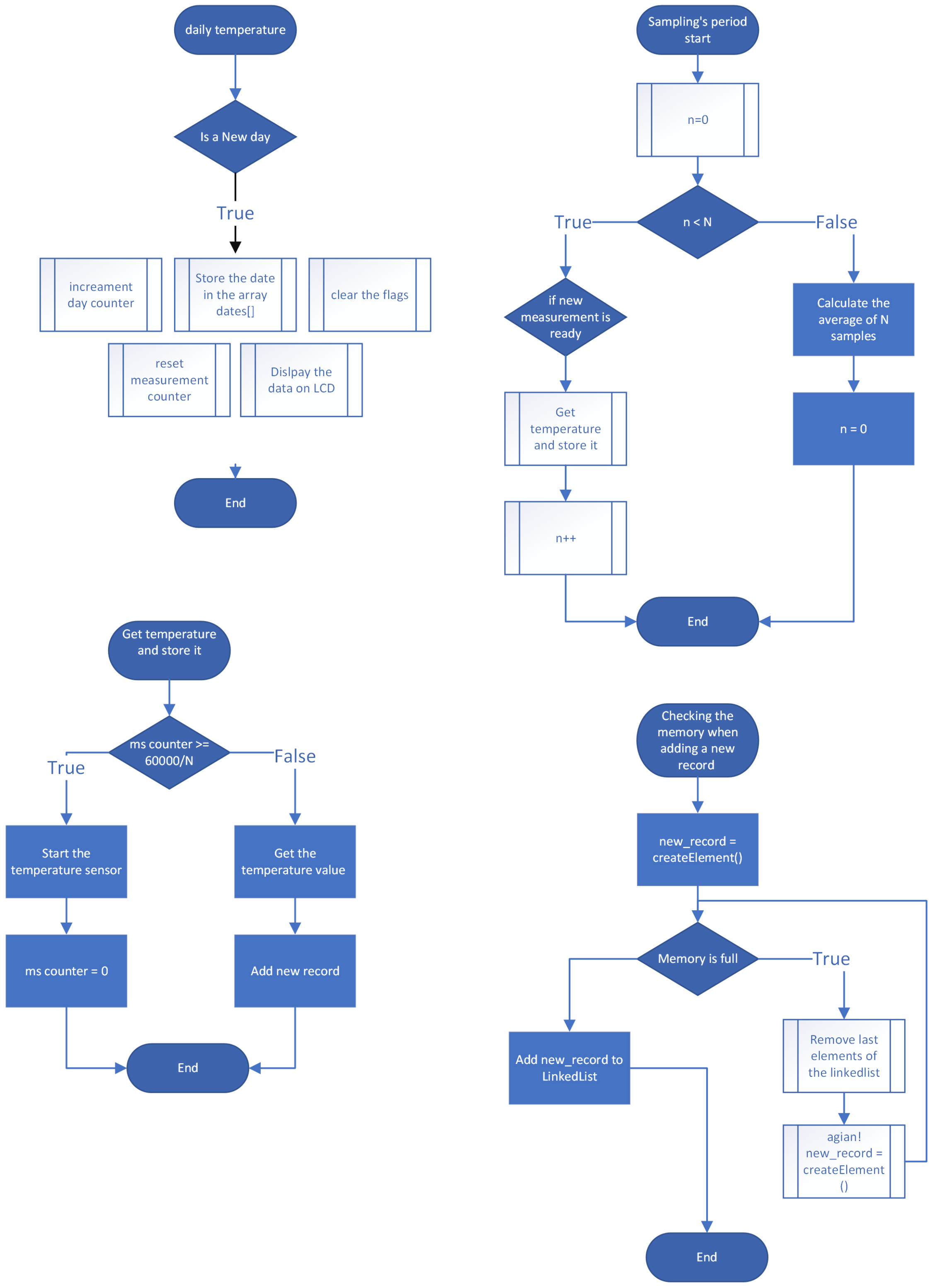

daily temperature
Is a New day
True
increament day counter
Store the date in the array dates[]
clear the flags
reset measurement counter
Dislpay the data on LCD
End
Sampling's period start
n=0
n < N
True
False
if new measurement is ready
Get temperature and store it
n++
Calculate the average of N samples
n = 0
End
Get temperature and store it
ms counter >= 60000/N
True
False
Start the temperature sensor
ms counter = 0
Get the temperature value
Add new record
End
Checking the memory when adding a new record
new_record = createElement()
Memory is full
True
Add new_record to LinkedList
Remove last elements of the linkedlist
agian! new_record = createElement ()
End

Figure 2: temperature functions

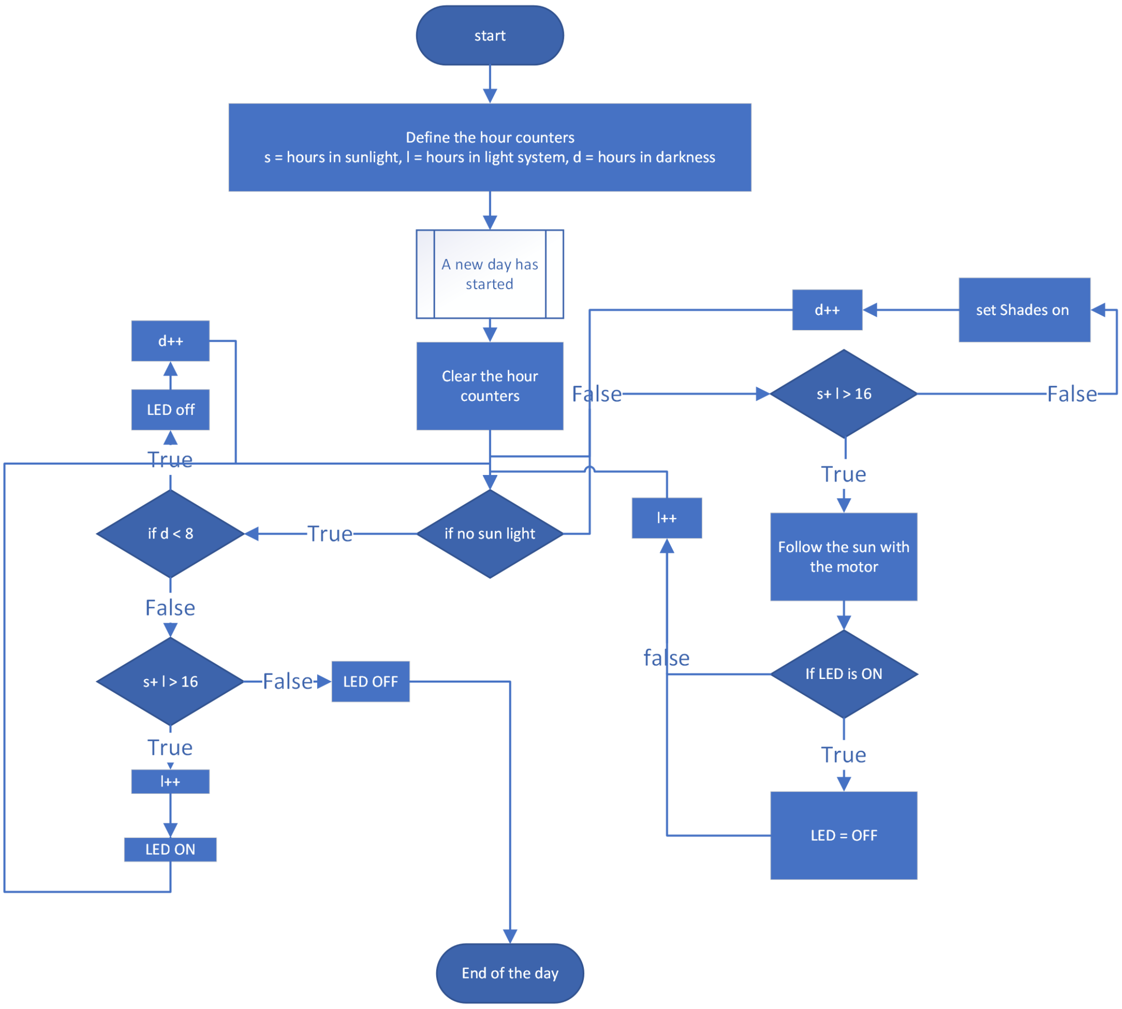
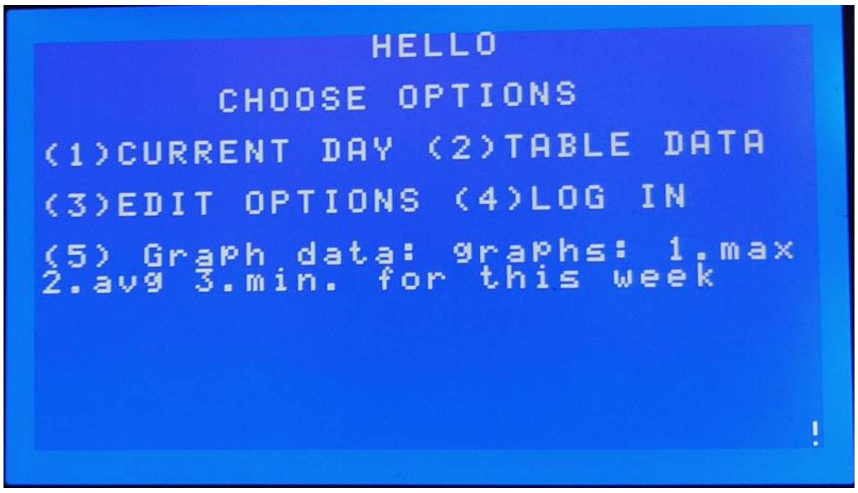
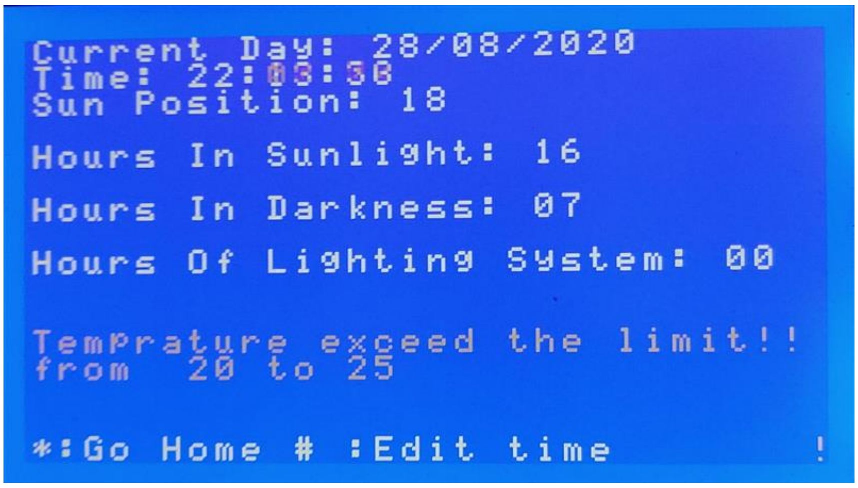


Figure 3: Greenhouse control of shades and light system



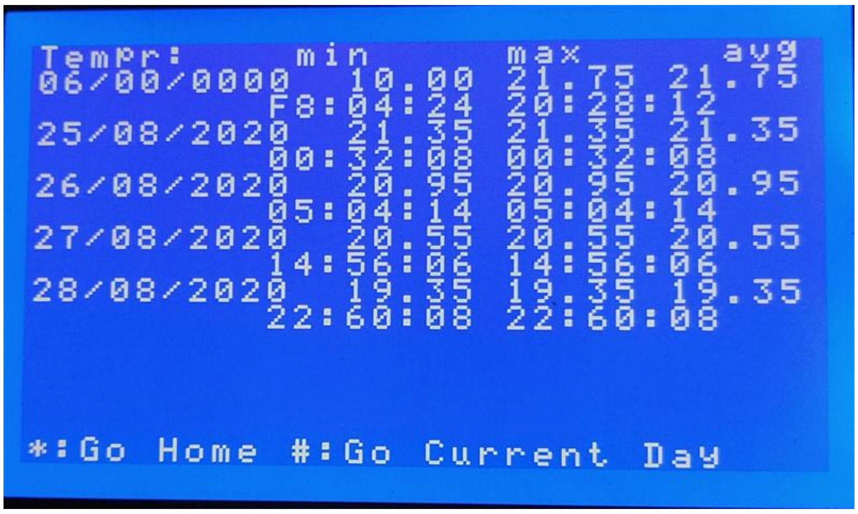
```
HELLO
CHOOSE OPTIONS
(1)CURRENT DAY (2)TABLE DATA
(3)EDIT OPTIONS (4)LOG IN
(5) Graph data: graphs: 1.max
2.avg 3.min. for this week
```

Figure 4: The GUI menu



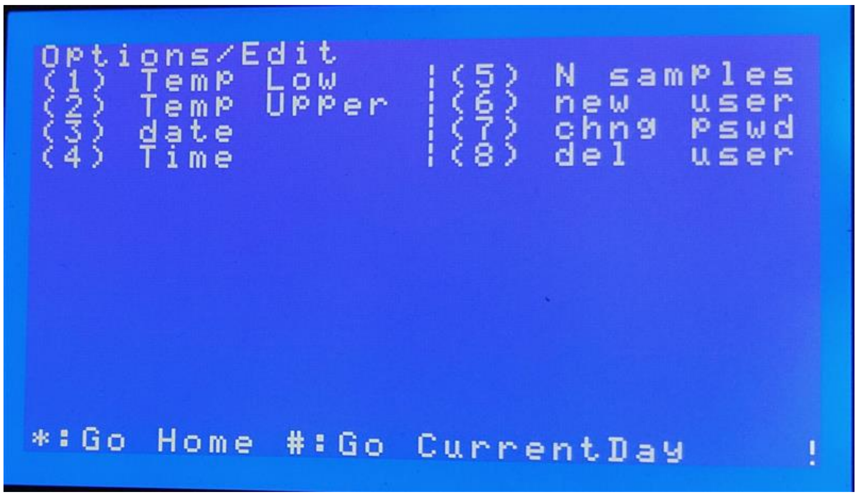
```
Current Day: 28/08/2020
Time: 22:03:30
Sun Position: 18

Hours In Sunlight: 16

Hours In Darkness: 07

Hours Of Lighting System: 00


Temprature exceed the limit!!
from  20 to 25


*:Go Home # :Edit time          !
```

Figure 5: The GUI page 1



```
Tempr:      min       max      avg
06/00/0000  10.00  21.75  21.75
          F8:04:24  20:28:12
25/08/2020  21.35  21.35  21.35
          00:32:08  00:32:08
26/08/2020  20.95  20.95  20.95
          05:04:14  05:04:14
27/08/2020  20.55  20.55  20.55
          14:56:06  14:56:06
28/08/2020  19.35  19.35  19.35
          22:60:08  22:60:08



*:Go Home #:Go Current Day
```

Figure 6: The GUI page 2



```
Options/Edit
(1) Temp Low      |(5) N samples
(2) Temp Upper    |(6) new  user
(3) date          |(7) chng pswd
(4) Time          |(8) del  user










*:Go Home #:Go CurrentDay        !
```

Figure 7: The GUI page 3