Smart Greenhouse Project

Computer Systems Engineering DT4012

January 2023

Hind Dakkeh & Khaled Mhjazi

Project Catalog

Abstract

The purpose of this project is to design and implement a simulation of a greenhouse using an Arduino Due microcontroller and various hardware components. The system is designed to track and record temperature data, maintain the temperature within a certain range, and provide a healthy light-to-darkness ratio for plants inside the greenhouse. The system also includes a calendar function to display the date and time, as well as an alarm feature to alert the user if the temperature exceeds certain limits. In addition, the system includes a fast mode option to allow for faster simulation of temperature recording and plant growth.

Introduction

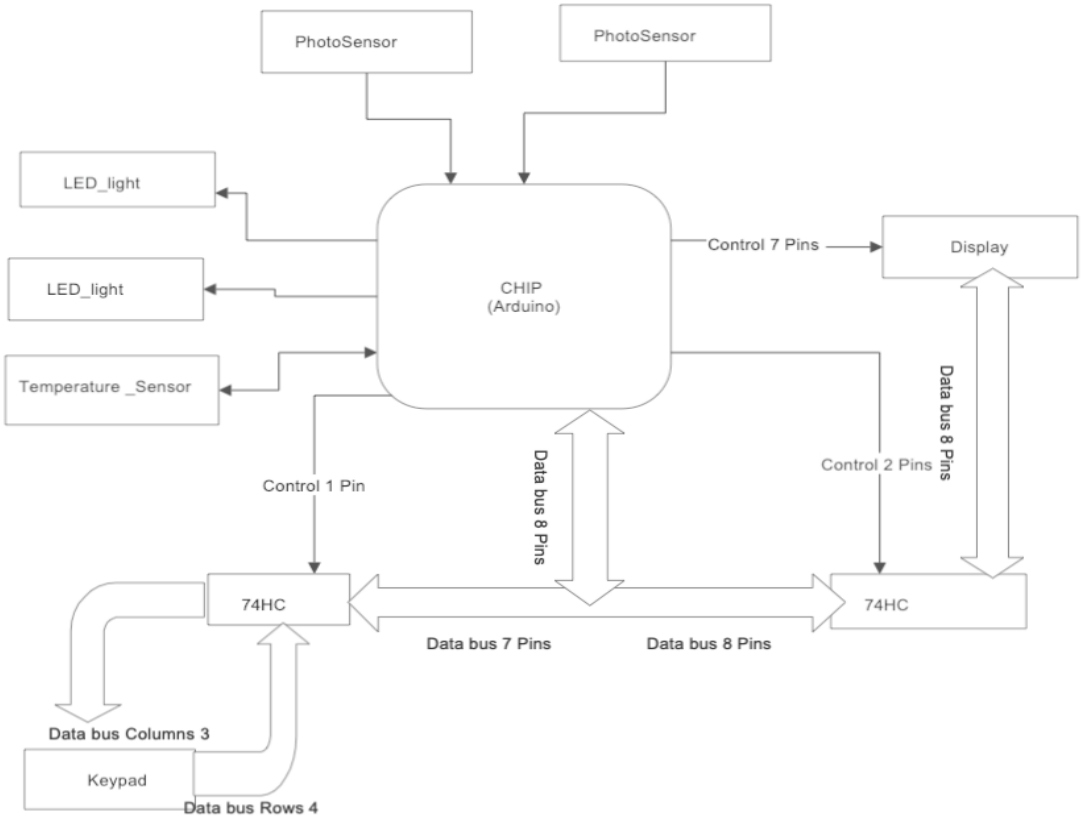
Purpose

The main goal of this project is to ensure that the greenhouse climate for the plants is within the desired range by monitoring and controlling the temperature and light intensity. Specifically, the aim is to maintain the temperature within the range of 20–25 degrees Celsius.

Components

Microchip (Atmel) SAM3X8E – ARM embedded computer platform (Arduino due)
Display (port I/O)
Keypad (Port I/O)
Temperature sensor (time)
Photosensor (A/D)
RC servo motor (PWM)
Data bus 74HC245
LEDs

Functional Block Diagram



The Embedded System

Microchip (Atmel) SAM3X8E (Arduino due)

The SAM3X8E is a microcontroller developed by Atmel, now a part of Microchip Technology. It is based on the ARM Cortex-M3 processor and is commonly used in the Arduino Due board, which is a microcontroller board based on the SAM3X8E. The ARM Cortex-M3 processor is a 32-bit processor that is designed for use in microcontroller applications. It has several features that make it well-suited for use in embedded systems, including low power consumption, a small size, and high performance. The Arduino Due board is a popular choice for projects that require a powerful microcontroller with many input/output (I/O) pins and a high-speed processor. It is the main microcontroller on the project which controls all Peripheral devices as (Display, keypad .... etc). Some of the key features of the SAM3X8E microcontroller and the Arduino Due board include:

32-bit ARM Cortex-M3 processor with a clock speed of 84 MHz.
512 KB of flash memory for storing programs and data.
96 KB of SRAM for fast data storage and processing.
54 digital I/O pins and 12 analog input pins [1].

Display

A liquid crystal display (LCD) is a type of electronic display that uses liquid crystals to display images and text. It is commonly used in devices such as TVs, computer monitors smartphones, and Screen is a black and white screen with a width of 30 and a height of 16. The LCD screen has a certain number of pins, which are 20, 8 pins of them used for transmitting and receiving data, and every character is coded in ASCII with offset 32, the rest used to control the screen. The screen has two modes read mode which is the display sends the data to Arduino and write mode which is the Arduino send the data to the display to do it [2][4].

Keypad

The keypad is connected to the data bus and configured with three signal input columns and four-row output lines. To determine which button has been pressed a low-level signal to one of the columns, and a high-level signal is sent to two of the three columns. When a button is pressed, the row output line corresponding to that button will have a low-level signal. For example, if a low-level signal is sent to column one and a low-level signal is read on row two, it indicates that button four has been pressed. If no button is depressed, all row output lines will be read as high [3].

Temperature

The temperature sensor is activated by sending a signal from the Arduino to initiate temperature readings. When this signal is received, the sensor's pin is configured as an input, allowing it to receive data from the sensor. The sensor sends back a series of waves, which are then used to determine the temperature. This is done by using a "Timer Clock1" to measure the time between the rising and falling edges of the waves.

The sensor operates by detecting rising and falling slope events, which are captured in two registers within the chip. One register is configured to store data when the sensor goes from a high to a low signal (falling slope), while the other register is configured to store data when the sensor goes from a low to a high signal (increasing slope). To begin the temperature measurement process, the sensor must first be reset by sending a high-to-low signal between 4.6 ms and 16 ms, in our case we use (15 ms), followed by a high signal. Then, a start pulse is generated using a delay function provided by the system's creators.

The difference between the rising and falling edges is used to calculate the number of cycles, and the time is determined by divided this value by the clock speed (42 MHz). The temperature in Celsius is then calculated using the equation: Temp[°C] = Time[µs]/5 - 273.15. This data is then used to regulate the greenhouse and ensure that it remains at the desired temperature [5].

Photosensor

The photosensor uses the ADC to measure the voltage that is produced when light passes through the sensor. The more light that is present, the higher the resistance and the lower the voltage will be, and vice versa. We have two photosensors, each of which has a channel that read through the ADC_CDR register, to get the value of voltage, first check if we have a value in the channel through

the ADCC_SR register, then we read 12 bits from the ADC_CDRx register then multiplier with (3.3/4095) to get the value of voltage [5].

Servo

The servo motor is controlled using pulse signals. To rotate the motor, a pulse is sent to it and the duty cycle is updated. The pulse width determines the angle at which the motor will point. The pre-scaler for the master clock is set to /32, which means that the duration of the pulse will vary depending on the desired angle of the motor. By adjusting the pulse width and the duty cycle, the servo motor can be made to rotate to any angle within its range of motion. In our case we use the highest pulse width modulation (2,3 ms), the servo represents the mirrors of a greenhouse that follow the sun light to get it into the plants and by using the servo we can determine the sun's position [5].

Data bus 74HC245

The 74HC245 a bus transceiver, which means it is used to transmit and receive digital data over a bus or communication link. The 74HC245 has eight input pins (A0-A7) and eight output pins (B0-B7). It also has a direction control pin (DIR) that determines the direction of data flow. When DIR is high, data flows from the A inputs to the B outputs. When DIR is low, data flows from the B inputs to the A outputs. The 74HC245 has a gate that is enable and can transfer the data when the gate pin is low and disable when the gate pin is high. In our case, we use two data bus, one of them connected to display and the another to keypad and when we want to transfer the data from the keypad, we enable the gate which connected to keypad and disable the gate which connected to display and vice versa.

LEDs

The LEDs are used to do two things:
The first(green) provides additional lighting which the plants need in the greenhouse when the level of natural sunlight is insufficient. The second (red) is used as an alarm that turns on when the temperature is outside a certain interval that configurable by the user. Both are connected to a pin on the Arduino that has been configured as an output, allowing it to be turned on and off as needed.
Project Requirements

Project Requirements

GUI

The project has a graphical user interface (GUI) where the user can interact with the system through the keypad and the display through the function Menu().
By pressing "1" set the Timestamp (Calendar).
pressing "2" Presentation of logged data
pressing "3" Light to darkness ratio, Sun position
pressing "4" Alarm at high or low temperature
when we press any number of above, we can go back to the menu through the press on the "*".
pressing "#" in fast mode.
pressing "0" in normal mode.

Timestamp (Calendar)

In the calendar.c file exists function for the day, month, year, hour, minute, and second. Every function appears to be implementing a process for setting the value on a calendar using a keypad. The function reads input from the keypad andupdates the value based on the input. The result is then returned as the output of the function. This SetDateTime() function allows the user to set the date and time on a calendar using a keypad. It reads input from the keypad and updates the values for the day, month, year, hour, minute, and second based on the input. The function begins a loop that will run repeatedly until the value of the "count" variable reaches 14. On each iteration of the loop, the function checks the value of "count" and calls one of several functions based on the value, the day and month, hour, minute, and second, need to be two places, but the year needs to four places, so these need to 14 places in the screen.

This function update () is updating the date and time on a calendar. The function increments the value of the "second" variable by 1 on each iteration to reach 60, and then checks the values of the other variables (minute, hour, day, month, and year) to ensure that they are all updated correctly based on the passage of time and checks if the values of "second", "minute", "hour", "day", or "month" are greater than their respective maximum values (60 for "second" and "minute", 24 for "hour", and 12 for "month"). If any of these variables have values that are too high, the function resets them to 0. By using Menu(), when pressing “1” in the keypad, then clears the display. It also displays a message on the screen indicating the date and time format (DD/MM/YYYY and hh:mm: ss) using plot (int x, int y, int size, char * value). Then set the date and time by calling SetDateTime(), and by pressing “5“ you can change the date and time.

The systick handler is configured to interrupt every 1 ms which updates the date and time in the fast mode and every 1 s in the normal mode, when we print these so we want to check if is the fast mode or the normal mode. On the screen, it will show you to enter the date and time that you want via the keypad. To print the numbers on the screen two functions are used two functions DatePlot() and printCalendar(int printvalue, int places, int x, int y).

Recording of temperature

The temperature is saved in the LinkedList for every minute by checking if the minuteFlag is 1 and by using the function insertfirst (LinkedListt **first, float t, int ho, int mi, int se). The LinkedList is a struct defined in the file "LinkedList.h " that has a variable for the temperature and variables for the time.

Every time we want to check if the LinkedList become full by using the function isFull() and if it is full, remove the last value from the list that was inserted by using RemoveLast (LinkedListt **first) and insert first a new node again. But we want to insert all values for every day, so we can check if the variable DaysCounter increment by 1 when the hour reaches 24, reaches 7 [6].

Presentation of logged data

After the temperature is read and saved in the linked list for every minute of the day, the function Min_Max_Average() compares the value for the temperature with the value saved in the variable "max", or "min", to save the maximum value and minimum value for the temperature in these variables and save the date and time in special variables for them, and the value for the average for every day.

Then put the value at the end of the day that is saved in the "min" in the mineral, this array saves the minimum temperature for every day. The same thing happens with maxArray but for the maximum temperature, and the Avg add the average array for every day. The dayArray is an array from type DaysList " this is the struct that has variables to save the date and the time for the maximum temperature and minimum temperature". Using these arrays, we could save all data that we need every day. When pressing 2 on the keypad, it shows on the screen the date, min, mintime, avg, max, and maxtime, for every day, by using the functions printDayView(), Print_Min_Max_Day().

Light to darkness ratio, Sun position

We regulate the length of the day and night and will set the day for 16 hours of light and the night for 8 hours of darkness that the plants needs. The start of the light is at 4 AM (4.00) and the end of the light is at 8 PM (20.00). During light time, the LED is active when the photosensor voltage for both sensors is above 0.22 '' this value for the sun'', but otherwise, the LED will be inactive because the sun exists. The darkness starts at 8 PM (20.00) and ends at 4 AM (4.00).

During the summer months, the sun light can be more than 16 hours during the day so we need to close the shades through the print at the screen "Turn on the shades" and open them again at 4 AM (4.00) through print "Turn off the shades" to get the light that the plants need using the functions sun_led() and printLightSystem().

By using the function rotateservo (int d), the servo can rotate in the highest to a certain degree that is determined by the function int differntvalue(), which calculates the difference between the light intensity between the two photosensors. In the main() the servo rotates every ten minutes for the fast mode and every second for the normal mode.

When pressing 3 on the keypad, it shows on the screen the number of hours of sun light led light, darkness, and degrees for the sun position by using the function printLightSystem().

Alarm at high or low temperature

The user can configurable the upper and lower limits of the temperature alarm inside the greenhouse through the keypad by pressing “4” and the LCD by using the function SetLowHigh().

The function SetAlarm() checks the temperature limits which was set previously and if the temperature is outside the interval the led will be turned on, and the user can reset the alarm

manually when pressing “7“ on the keypad. The screen show the low temperature and high temperature by using the function AlarmPlot(). By pressing “6“ you can change the Alarm.

High speed simulation to simplify testing

When pressing "#" on the keypad the fast mode will be active through "fastflag=1" which means the update() function will call every 1 ms to update the time and date to simulation for 7 days to get minimum and maximum and average for every day faster. By pressing "0" the program is back to normal mode which means the date is updated every second.

Conclusion section

We encountered some problems at work, including the memory space when saving temperatures in the LinkedList, where we modified the project settings and set the highest HEAP in the memory to "0xffff". In addition to that, we saved the temperatures with timestamp time, because we did not need to save the date, and thus we saved the largest possible number of data At the same time, before modifying, we were able to save 256 nodes before the memory was full, and after these modifications, we were able to save 2044 nodes before the memory was full, and thus we solved this problem that we faced.

We encountered another problem while printing on the screen, where the temperatures were incorrect or with a score of minus a large degree, and the reason was that the program was busy printing and because the printing on the screen was in a while loop without controlling the duration of printing or at any time the program updates the print. This makes degrees The temperature is read incorrectly, and we have fixed this problem by printing at certain times and as needed. For example, we need to update the printing in the interface that displays the min, max, and average every day once, because the program gives new values once every day, and for this, we need to print it every day only one time.

References

[1] [SAM3X8E Datasheet] https://ww1.microchip.com/downloads/en/devicedoc/atmel-11057-32-bit-cortex-m3-microcontroller-sam3x-sam3a_datasheet.pdf

[2] [Display NHD-240128WG-ATMI-VZ# Datasheet] https://pdf1.alldatasheet.com/datasheet-pdf/view/1027222/NEWHAVEN/NHD-24064WG-ATMI-VZ.html

[3] Hazem Ali, Computer Systems Engineering I, Hardware Lecture 3.

[4] Hazem Ali, Computer Systems Engineering I, Hardware Lecture 4.

[5] Hazem Ali, Computer Systems Engineering I, Hardware Lecture 5.

[6] Hazem Ali, Computer Systems Engineering I, Software Lecture 3.