CYBER-PHYSICAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

THERMOSMART

GROUP A6

ABDUL FIKIH KURNIA	**2106731200**
BISMA ALIF ALGHIFARI	**2106731402**
IBRAHIM RIJAL	**2106633323**
M. VARREL BRAMASTA	**2106733811**

PREFACE

We are excited to present our final project, "ThermoSmart: Temperature Control and Temporary Disposal of Room Cooling Water in the Cyber-Physical Systems Course." With the rapid advancement of technology, cyber-physical systems have become a significant focus in various aspects of everyday life. Our project addresses the challenge of effectively and efficiently controlling temperature and disposing of room cooling water. Through the development of the innovative "ThermoSmart" system, we aim to provide practical and efficient solutions for temporary temperature control and water disposal in room cooling. By combining our knowledge from the Cyber-Physical Systems course, we utilize sensors, micro controllers, and etc to control temperature and water disposal, while also creating a user-friendly interface.

We extend our gratitude to Assistant Laboratory Miranty Anjani Putri for her invaluable guidance and support throughout the project. Her expertise and advice have been essential in overcoming technical obstacles. We also want to express our appreciation to Supervisor F. Astha Ekadiyanto, S.T., M.Sc., for sharing their guidance and knowledge with us. Their dedication and commitment have been instrumental in guiding us through every step of the project.

This project provides us with a valuable opportunity to apply our acquired knowledge and witness the direct impact of our efforts. We aspire that the solutions we develop make a positive contribution to the field of cyber-physical systems. Finally, we would like to thank all those who have supported and encouraged us during the project's development process.

Depok, Mei 16, 2023

Group A6

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

In a room equipped with an air conditioner, individuals may experience discomfort when they feel that the temperature produced by the air conditioner is sometimes too cold and does not match their preferences. While air conditioning can be an effective solution for cooling the surrounding air, the temperature settings may not always meet the expectations of the individuals in the room. Apart from the issue of inconsistent temperatures, the environment surrounding the room can exacerbate the situation. Factors such as inadequate ventilation or leaks in the walls can affect the efficiency of the air conditioning system. As a result, the airflow produced by the air conditioner becomes uneven, with certain areas in the room feeling colder than they should be.

This condition leads to the need for temporary storage of the air conditioner's condensate water in containers. The purpose of this storage is to collect the water condensed by the air conditioner, which usually consists of the humidity removed from the indoor air. However, a problem arises when these storage containers are not promptly emptied when they become full. Negligence in managing this condensate water can lead to the risk of flooding inside the room. When the storage containers are full, there is no more space to accommodate the condensate water. The water can overflow from the containers, causing puddles around the air conditioner or even seeping into the floor. Apart from creating physical messiness, prolonged wet conditions can also damage furniture or floor surfaces.

To avoid the risk of flooding due to full storage containers, proper handling of the air conditioner's condensate water is necessary. Preventive measures include regularly emptying the containers or installing more efficient drainage systems.

1.2 PROPOSED SOLUTION

To address the aforementioned issues regarding the air conditioner being too cold and inadequate management of condensate water storage, the use of a smart control system can be

a viable solution. By integrating temperature and water level sensors into the air conditioning system, the system can be designed to operate more efficiently and effectively.

The temperature sensor plays a crucial role in ensuring the room remains at a comfortable temperature. It periodically reads the room temperature, and based on predefined thresholds, the system determines whether to activate or deactivate the air conditioner. If the temperature exceeds a certain threshold, indicating that the room is too warm, the system activates the relay to turn on the air conditioner. Conversely, if the temperature falls below a certain threshold, indicating that the room is too cold, the system deactivates the relay to turn off the air conditioner. This smart control mechanism ensures the air conditioner operates only when necessary, preventing excessive cooling and maintaining optimal comfort for the occupants.

In addition to temperature control, the water level sensor addresses the issue of inadequate management of condensate water storage. It periodically reads the height of the air conditioner's discharge water in the drainage tank. When the water level reaches the maximum level, indicating that the storage container is nearing full capacity, the system sends a notification to the user to manually empty the container. By providing this notification, the user becomes aware that the storage container needs to be emptied to prevent water overflow and the potential for flooding in the room. If the water level reaches the maximum limit, the water level sensor detects it and sends a signal to the control system. The control system then deactivates the relay that controls the air conditioner.

The implementation of Thermosmart with temperature and water level sensors provides a smarter and more efficient solution for managing room temperature and condensate water. Users can control the room temperature according to their needs, while the management of condensate water can be done in a timely and targeted manner. Thus, the comfort and operational efficiency of the air conditioner can be enhanced, avoiding temperature-related issues and the risk of flooding due to full storage containers.

1.3 ACCEPTANCE CRITERIA

The acceptance criteria of this project are as follows:

1. The system must be able to read temperature and water level input from the sensors and display it via LCD display
2. The system must be able to alert the user using buzzer when the water level exceeds the threshold
3. The system must be able to turn off the AC when the room temperature reaches the predetermined value.
4. The system must be turn OFF if the button is pressed

1.4 ROLES AND RESPONSIBILITIES

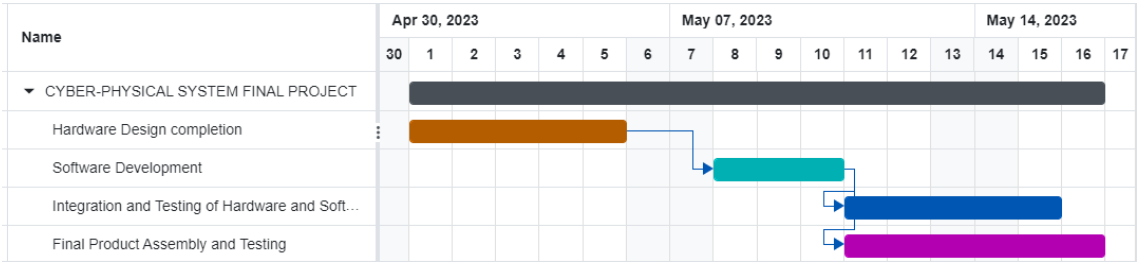
The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Codes and Proteus	LCD Display, Proteus, Master Slave.	Abdul Fikih Kurnia
Codes and Reports	Water level sensor, Reports, Slides PPT.	Bisma Alif Alghifari
Codes and Circuits	Buzzer driver, relay driver, Master Slave, Circuits Create.	Ibrahim Rijal
Codes and Reports	DHT 11, Reports.	M. Varrel Bramasta

Table 1. Roles and Responsibilities

1.5 TIMELINE AND MILESTONES

Insert Gantt Chart here. The Gantt Chart should consist of date interval for:



a) Hardware Design completion: A milestone indicating the date when the hardware design for the embedded system is finalized, including schematic.

b) Software Development: The date when the development of the user-created assembly code (software) begins, focusing on specific tasks and functionalities.

c) Integration and Testing of Hardware and Software: A milestone indicating when the hardware and software components are integrated and tested together to ensure proper functionality.

d) Final Product Assembly and Testing: A milestone marking when the final system product is assembled, tested, and verified to meet the acceptance criteria.

CHAPTER 2

IMPLEMENTATION

2.1 HARDWARE DESIGN AND SCHEMATIC

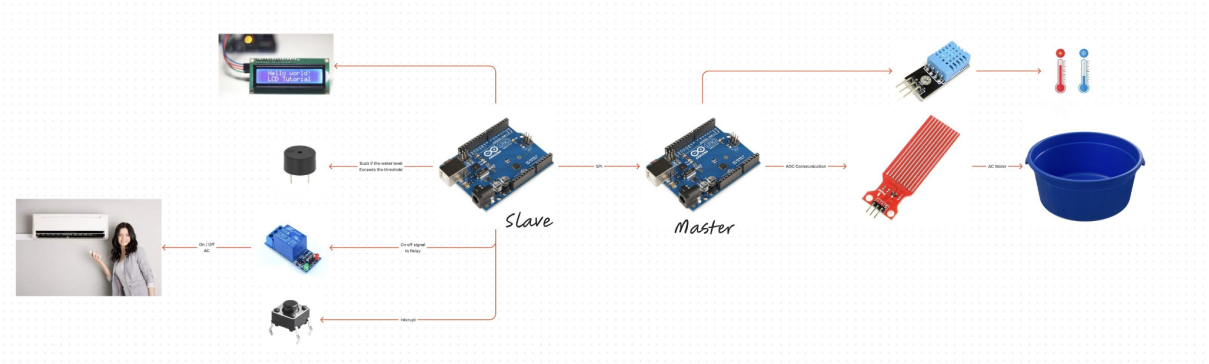


Figure 1 Schematic hardware

Our hardware schematics consists of two arduinos, water level sensor, temperature sensor, buzzer, relay, push buttons, and an LCD Display. The two arduinos will be used separately. The first one will be configured as a master that interacts directly with the user and the device its controlling, while the second one will be configured as a slave that receives input from the sensors and sends it to the master using I2C, where the data will be processed and displayed.

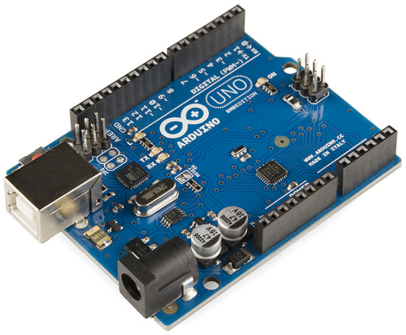


Figure 2.1 Arduino uno

Figure 2.2 LCD display

The water level sensor interfaces via analog input that goes into ADC, where the analog signal will be processed and converted into digital signal, ranges from 0 to 533, whereas the temperature sensor interfaces via digital input where the checksum from the sensor will be processed. Due to the sensor's properties that also outputs humidity data in its checksum, a processing should be done to ensure the data received is only the temperature data, this processing is done in the slave device. Then, the temperature, and water level data is sent to the master device using I2C interface

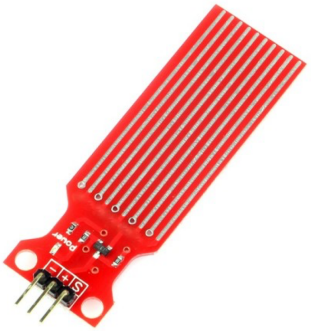


Figure 2.3 Water Level

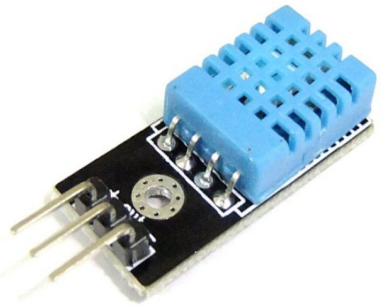


Figure 2.4 DHT11

In the master device, the temperature and water level data will be displayed in the LCD display. The master device is also responsible for driving the buzzer and relay. The master will play a 1kHz tone in the buzzer if the water level exceeds the predetermined

threshold. The master device is also controlling the relay, so that it will be turn off everytime the room temperature reaches a predetermined minimum threshold.

Figure 2.5 Buzzer

Figure 2.6 Relay

Figure 2.7 Button

2.2 SOFTWARE DEVELOPMENT

Our project is implemented in Arduino Unos based on Atmel ATMega328p using assembly as the main language.

THERMOSMART FLOWCHART :

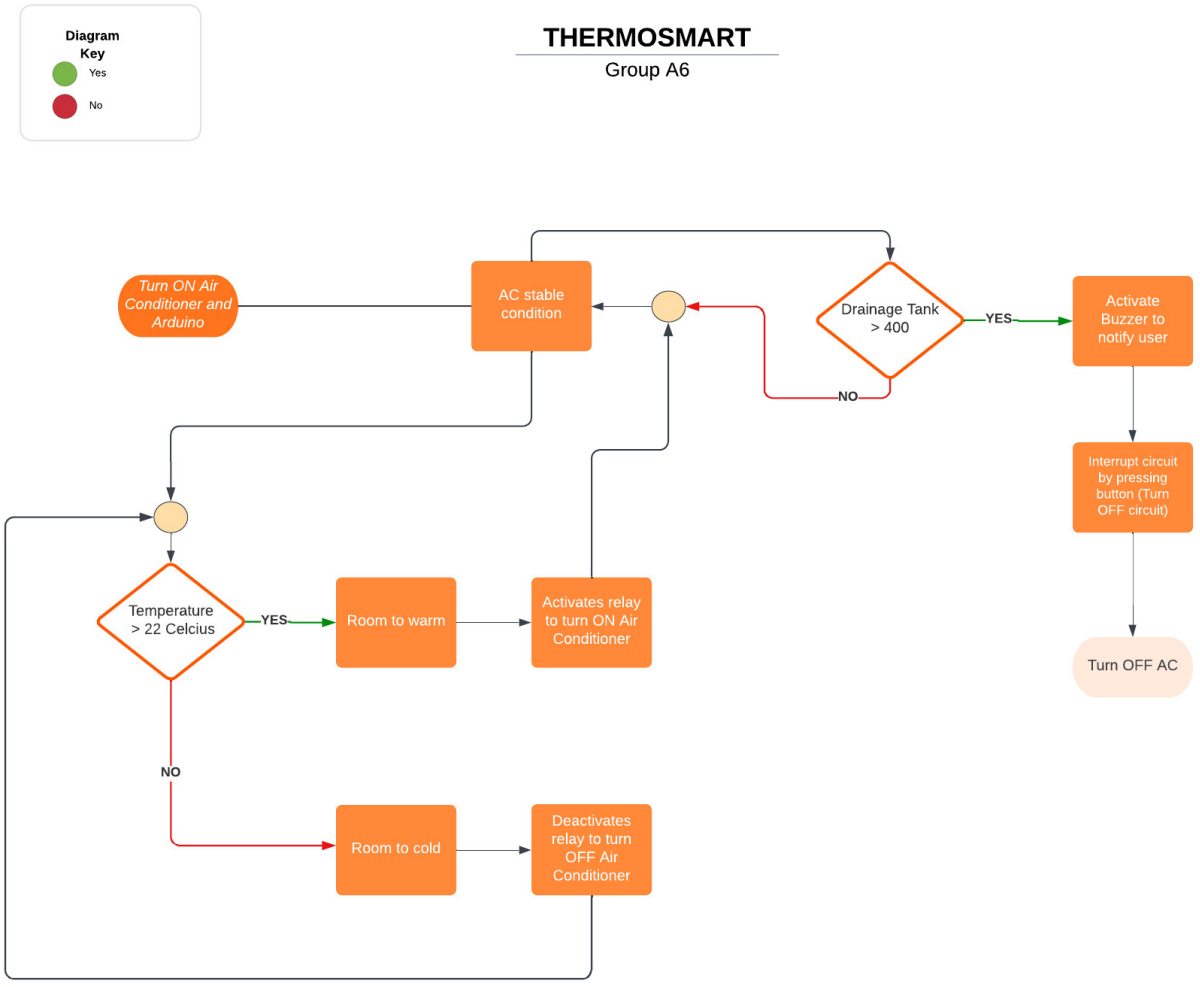


figure 2.8 Flowchart

1. DHT11

The code begins with some preprocessor directives and the declaration of the "DHT11_sensor" function as a global symbol."DHT11_sensor" function is the main entry point. It starts by calling the "delay_2s" subroutine to wait for 2 seconds for the DHT11 sensor to get ready.Next, it initializes the necessary pins and sends a start signal to the DHT11 sensor. It sets pin PD7 as an output and sends a low pulse for 20 milliseconds followed by a high pulse.After sending the start signal, the code waits for the DHT11 sensor to send a response signal. It sets pin PD7 as an input and waits for a low pulse from the sensor. Then it waits for a high pulse and another low pulse.Once the response signal is received, the code calls the "DHT11_reading"

subroutine three times to read the humidity and temperature data from the sensor. The received data is stored in registers R24 and R25.After reading the data, the code jumps back to the "agn" label to repeat the process for another sensor reading.The "DHT11_reading" subroutine is responsible for receiving 8 bits of data from the DHT11 sensor. It uses a loop to detect the data bit and waits for a high pulse. After a delay of 50 microseconds, it checks the received bit value and shifts it into the data register (R24) accordingly. The subroutine repeats this process for all 8 bits.The code also includes two delay subroutines: "delay_20ms" and "delay_2s". These subroutines provide delays of 20 milliseconds and 2 seconds, respectively, using software loops.Additionally, there is a "delay_timer0" subroutine that provides a 50-microsecond delay using Timer 0 of the microcontroller.

2. Master

   The main function is the entry point of the program. It starts by defining some constants (SCK, MOSI, and SS) representing the pin numbers for the SPI communication. The SPI interface is then initialized by configuring the corresponding pins as outputs and setting up the SPI control register (SPCR) with the necessary settings for SPI master mode.After the initialization, the code enters a loop labeled again, where it performs the SPI communication with the DHT11 sensor and the water sensor. It starts by calling the dht_start subroutine, which triggers the DHT11 sensor to start sending data. It then waits for the response from the DHT11 and reads the humidity and temperature data.Next, the code calls the sensor_Water subroutine to read data from the water sensor using the ADC (Analog-to-Digital Converter). The result is stored in registers R16 and R25 (low and high bytes of the sensor data).After reading the sensor data, the code performs some delay using the delay_4sec subroutine, which introduces a 4-second delay. Finally, the code repeats the transmission by jumping back to the again label.There are additional subroutines included in the code. DHT11_reading subroutine is responsible for reading individual bits of data from the DHT11 sensor. It waits for specific high and low pulses and determines the value of each bit received.delay_20ms subroutine introduces a 20ms delay using nested loops and software counting.delay_timer0 subroutine implements a 50µs delay using Timer 0. It utilizes the timer's compare match functionality and a specific configuration of Timer 0 registers.delay_4sec subroutine creates a 4-second delay using Timer 1. It sets up Timer 1 with a specific counter value and prescaler

settings to achieve the desired delay.Lastly, the sensor_Water subroutine initializes the ADC to read data from an analog input pin (ADC0 or PC0 on the microcontroller). It then triggers an ADC conversion and waits for the conversion to complete before retrieving the result.

3. Slave

main function sets pin PC0 as an input for ADC0, sets up the ADC with a 2.56V reference voltage and a prescaler of CLK/128, and enables SPI as a slave. It then enters a loop where it reads data from SPI and displays it on the LCD display using the LCD_write subroutine.LCD_write subroutine initializes the LCD display, displays a message on the first line, and then displays the ADC value on the second line. It does this by converting the ADC value to an ASCII character and then sending the character to the LCD display using the data_wrt subroutine. command_wrt subroutine is used to send commands to the LCD display.first subroutine data_wrt seems to write data to the LCD display. It uses the OUT instruction to output the high nibble of a register to the PORTD data register. It then sets the RS bit to 1 to indicate that the data being written is actual data, not an instruction. It sets the EN bit to 1, followed by a short delay. Then, it sets the EN bit to 0, followed by another delay. It repeats this process for the low nibble of the register.delay_short subroutine is simply a delay of two NOP instructions.delay_us subroutine is a delay of approximately 100 microseconds, achieved by a loop that repeats 90 times and calls delay_short subroutine each time.delay_ms subroutine is a delay of approximately 20 milliseconds, achieved by a loop that repeats 40 times and calls delay_us subroutine each time.delay_sec subroutine is a nested loop that provides a delay of approximately 3.11 seconds.disp_msg1 subroutine displays a message "WaterLvl: " on the LCD display. It loads the address of the message string into the Z register, reads each character from the string using the LPM instruction, and calls the data_wrt subroutine to display each character.encrypt subroutine is used to encrypt data. It moves the high byte of a register to R16 and compares R16 with 228. If R16 is greater than or equal to 228, it jumps to adjust. Otherwise, it enters a loop where it compares R16 with 100. If R16 is greater than or equal to 100, it increments a counter and subtracts 100 from R16. It continues to loop until R16 is less than 100. It then compares R16 with 10 and increments another counter if R16 is greater than or equal to 10. Finally, it subtracts 10 from R16 until it is less than 10. It then pushes the

counters onto the stack.clr subroutine sets the ADSC bit in ADCSRA register to start an ADC conversion.

4. Buzzer

   Inside the "main" function, the registers R16 and R17 are loaded with binary values 00100000 and 00000000, respectively. R16 is used to toggle the state of pin PB5 (D13). Pin PB5 is set as an output by setting the corresponding bit in the DDRB register. The value of R17 is written to the PORTB register, setting PB5 to a low state.Pin PC0 (A0) is set as an input by clearing the corresponding bit in the DDRC register. The internal pull-up resistor for PC0 is enabled by setting the corresponding bit in the PORTC register.The program then enters a loop labeled "agn." Inside this loop, R17 is loaded with the value 00000000, and it is written to the PORTB register, setting PB5 to a low state. The code then checks if pin PC0 is high using the SBIS instruction. If PC0 is high, the program jumps to the "l1" subroutine. Otherwise, it continues looping And The "l1" subroutine is responsible for generating a 1000 Hz square wave. It calls The register R18 is loaded with the value 100.the "delayhalfms" subroutine to create a delay of 0.5 milliseconds using Timer0. After the delay, R17 is XORed with R16 using the EOR instruction, toggling the state of PB5. The new value of R17 is written to the PORTB register. The code then checks if PC0 is high using the SBIS instruction. If PC0 is high, indicating a button press, the program jumps back to "l1" and repeats the toggle. Otherwise, it jumps to the "agn" loop and repeats the process.The "delayhalfms" subroutine generates a 0.5 ms delay using Timer0 to create a 1 kHz tone. It initializes Timer0 with appropriate values and enters a loop where it checks the OCF0A flag in the TIFR0 register. When the flag is set, indicating the desired delay has passed, the subroutine stops Timer0 and clears the flag. It then returns to the calling subroutine.Additionally, the code contains two duplicate subroutines named "buzzer_toggle_on" that toggle the state of pin PB1.

5. Relay

   Inside the "main" function, the registers R16 and R17 are loaded with binary values 00010000 and 00000000, respectively. R16 is used to toggle the state of pin PD5, which is connected to a relay. Pin PD5 is set as an output by setting the corresponding bit in the DDRD register. The value of R17 is written to the PORTD register, setting PD5 to a low state.Pin PC0 is set as an input by clearing the corresponding bit in the

DDRC register. The internal pull-up resistor for PC0 is enabled by setting the corresponding bit in the PORTC register.The program then enters a loop labeled "loop." Inside this loop, the code checks if pin PC0 is high using the SBIC instruction. If PC0 is high, indicating a button press, the program continues looping. Otherwise, it jumps to the "relay_toggle" subroutine.The "relay_toggle" subroutine is responsible for toggling the state of pin PD5. It XORs the value of R17 with R16 using the EOR instruction, toggling the state of PD5. The new value of R17 is written to the PORTD register. After that, the program jumps back to the "loop" and repeats the process.

6. Display LCD

The code provided is written in assembly language for a microcontroller. The main function begins by setting pin PC0 as an input for ADC0 using the SBI instruction. Then, the code sets up the ADC by configuring the ADMUX and ADCSRA registers with appropriate values.Next, the code jumps to the LCD_write subroutine. The LCD_write subroutine initializes an LCD display by sending commands to its command register. It sets up the data and command pins as output and clears the display. It then calls the LCD_init subroutine to initialize the LCD. Following that, it calls the disp_msg subroutine to display a message on the LCD.Afterward, it sets up the R16 register with the value 0x20 and calls the data_wrt subroutine to write the value to the LCD. Then, it initializes R19 with the value 48 and jumps to the encrypt subroutine.The LCD_init subroutine initializes the LCD by sending a sequence of commands to the command register. It sets up the LCD for 4-bit data, specifies two lines and a 5x7 matrix, turns on the display, clears the display, and sets the cursor to shift right.The command_wrt subroutine is responsible for writing commands to the LCD. It takes the value from R16 and sends the high nibble to the LCD's data pins, sets the RS pin (Register Select) to 0 for a command, toggles the EN pin (Enable) to execute the command, and introduces delays.The data_wrt subroutine is similar to command_wrt but is used for writing data (such as characters) to the LCD.disp_msg1 subroutine is responsible for displaying a specific message on the LCD. It uses the LPM (Load Program Memory) instruction to load characters from the msg1 string into R16. It checks if the character is the end of the string (0x20) and exit if it is. Otherwise, it calls the data_wrt subroutine to display the character on the LCD. subroutine loops back to load and display the next character until the end of the string is reached.msg1 string contains the message "WaterLvl: ".clr subroutine sets the

ADSC (ADC Start Conversion) bit in the ADCSRA register to start an ADC conversion.encrypt subroutine performs some calculations based on the value in R16 (which is assumed to contain an ADC value). It compares R16 with certain thresholds and increments counters (R25 and R23) accordingly. These counters are used to generate ASCII characters representing the most significant digit (MSD), middle digit, and least significant digit (LSD) of the ADC value. The subroutine then calls the data_wrt subroutine to display these digits on the LCD.After displaying the ADC value, the code sets R16 to 0x20 (a space character) and calls data_wrt to display a space on the LCD. It then sets R16 to 0x43 (the character 'C') and calls data_wrt to display 'C'. The code moves R16 to 0xC0 and calls the command_wrt subroutine to position the cursor at the beginning of the second line of the LCD.Next, the code calls the disp_msg1 subroutine again to display the message "WaterLvl: " on the first line of the LCD.Following that, the code retrieves another ADC value and performs similar calculations and display operations to show the temperature reading on the LCD. The temperature-related characters displayed are a space (0x20), the percentage symbol (0x25), and the character 'C'.

7. Water Level Sensor

   Inside the "main" function, pin PC0 is set as an output by setting the corresponding bit in the DDRC register. Then, the value 0x40 is loaded into register R20, and this value is stored in the ADMUX register to configure the ADC. The ADC is set to read the ADC0 input pin, and the ADC is enabled by setting the prescaler to 128 in the ADCSRA register.The "init_serial" subroutine initializes the UART (serial communication interface) for communication with an external device. The registers UCSR0A, UBRR0H, and UBRR0L are set to configure the baud rate and enable the UART transmitter and receiver.The "print_ADC" subroutine is responsible for reading the ADC measurement results, converting them to ASCII characters, and sending them through the UART. It begins by setting the ADSC bit in the ADCSRA register to start the ADC conversion. Then, it waits for the conversion to complete by checking the ADIF flag in the ADCSRA register. Once the conversion is done, the ADC result is obtained from the ADCL and ADCH registers. The result is then converted to ASCII characters and sent through the UART. The subroutine handles the transmission of the most significant digit (MSD), middle digit, and least significant digit (LSD) of the ADC result.The code includes various loops that check the UART

data buffer's status (UDRE0 bit in UCSR0A register) to ensure that data can be sent before transmitting the ASCII characters.The "delay_sec" subroutine implements a 1-second delay using nested loops. It counts down three registers (R20, R21, and R22) to introduce a delay of approximately 1 second.After the necessary initialization and subroutines, the program jumps to the "print_ADC" subroutine to continuously read and send the ADC measurement results through the UART. It includes additional code for printing a newline character and a carriage return character after each measurement and introduces a delay of approximately 1 second between measurements using the "delay_sec" subroutine.

2.3 HARDWARE AND SOFTWARE INTEGRATION

To integrate the hardware and software components of the Smart Thermostat application with the Arduino IDE, you would need to follow these steps:

1. Connect the Temperature Sensor (DHT11) to the Arduino:
    - Connect the VCC pin of the DHT11 to the 5V pin on the Arduino.
    - Connect the GND pin of the DHT11 to the GND pin on the Arduino.
    - Connect the DATA pin of the DHT11 to a digital pin on the Arduino
    - DHT11 will send data in hexadecimal form to the LSD display with the intermediary of two Arduino SPI communications.
2. Connect the Water Level Sensor to the Arduino:
    - Connect the VCC pin of the water level sensor to the 5V pin on the Arduino.
    - Connect the GND pin of the water level sensor to the GND pin on the Arduino.
    - Connect the data pin of the water level sensor to a digital pin on the Arduino.
    - Water Level sensor will read water level send data to the LSD display with the intermediary of two Arduino SPI communications.
3. Buzzer:
    - buzzer function to give a warning or alarm if the water level exceeds a certain threshold. If the water level reaches or exceeds the threshold, the master device will send a signal to the buzzer, and the buzzer will play a 1kHz tone as a warning to the user.

- the buzzer receives data from the water level through the intermediary of two arduino with SPI communication

4. LCD Display :
    - The LCD Display will display data from the temperature and water level on the master slave
    - The LCD display will display data taken from DHT11 (in the form of temperature) and water level
5. Relay:
    - Connect the VCC pin of the relay module to a suitable power supply on the Arduino board.
    - Connect the GND pin of the relay module to the common ground (GND) connection on the Arduino.
    - Connect the signal pin of the relay module to a digital output pin on the Arduino.
6. Turn On/Off Button:
    - If the button is pressed then button will send a command to turn off the circuit as a whole

CHAPTER 3

TESTING AND EVALUATION

3.1 TESTING

Testing Thermosmart involves simulating different conditions to evaluate its functionality. The first condition to test is a scenario with a low water level and a high temperature. In this case, the water level in the condensate container is below the predefined threshold (under 400 value), indicating that it does not requires attention. The temperature in the room is above the desired range (above 22 Celcius), signifying the need for cooling.

Moving on to the second condition, we simulate a scenario with a high water level and a high temperature. In this case, the water level in the condensate container exceeds the maximum capacity, indicating that it is full and requires immediate attention (above 400

value). The temperature in the room remains above the desired range (above 22 Celcius), further emphasizing the need for cooling.

The third condition to test is a scenario with a low water level and a low temperature. In this case, the water level in the condensate container is below the predefined threshold, indicating that it does not requires attention (under 400 value). However, the temperature in the room is below the desired range (under 22 Celcius), suggesting that we don't need cooling.

Going on to last condition, we simulate a scenario with a high water level and a low temperature. In this case, the water level in the condensate container exceeds the maximum capacity, indicating that it is full and requires immediate attention (above 400 value). Additionally, the temperature in the room is below the desired range(under 22 Celcius), indicating that cooling is not needed.

3.2 RESULT

During this first condition test, the expected output from Thermosmart is as follows. The LED indicator turns ON, indicating that the water level is low and doesn't need to be addressed but the temperature is above the desired range. Additionally, the AC is turned ON to cool the room and maintain a comfortable temperature for the occupants. This condition is verified by the testing that is being done on the physical circuit, Referring to image below :

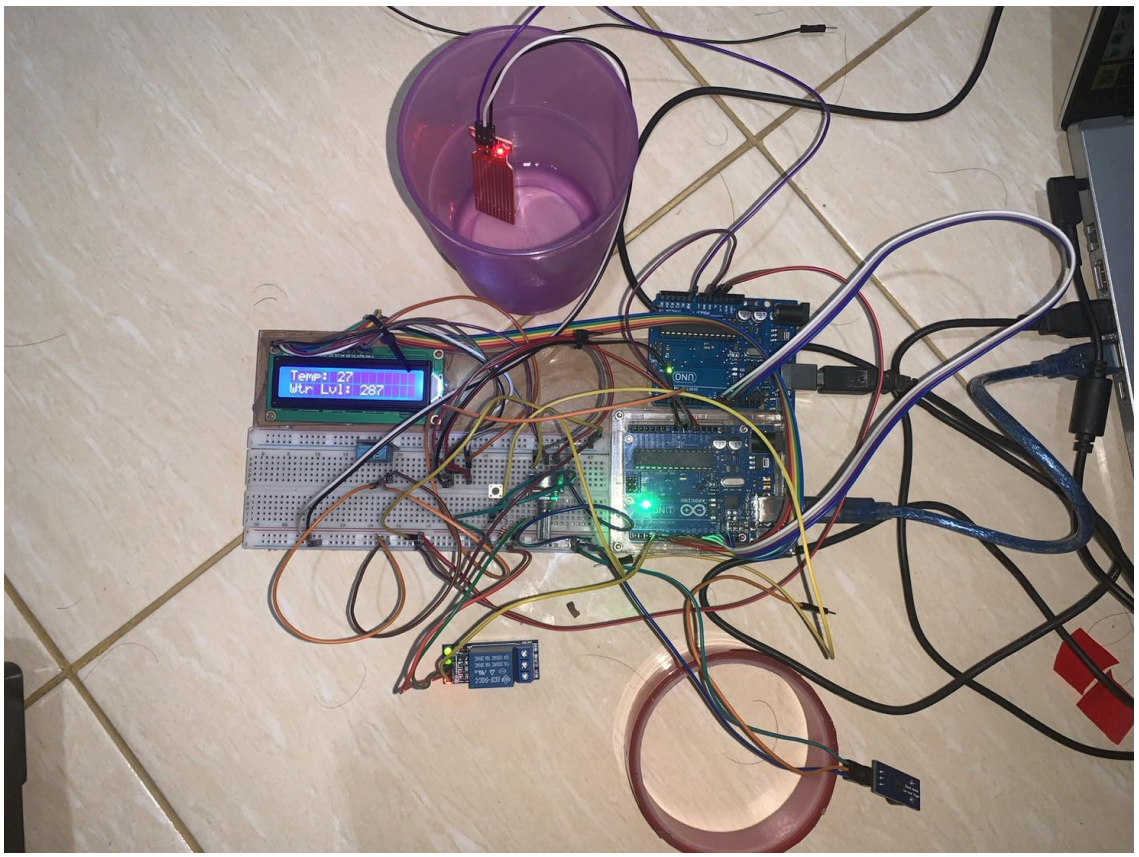


Fig 3.1 Testing Result

During this second condition test, the expected output from Thermosmart is as follows. The LED indicator turns OFF, indicating that the water level is high and has reached its maximum capacity. Also the temperature is above the desired range. This serves as a visual alert to the user to promptly empty the container to prevent overflow or potential damage. Additionally, the AC is turned OFF to prevent further cooling and stopping the water leakage from the container. This condition will also trigger a buzzer to notify the user that the drainage tank is Full. This condition is verified by the testing that is being done on the physical circuit, Referring to image below :

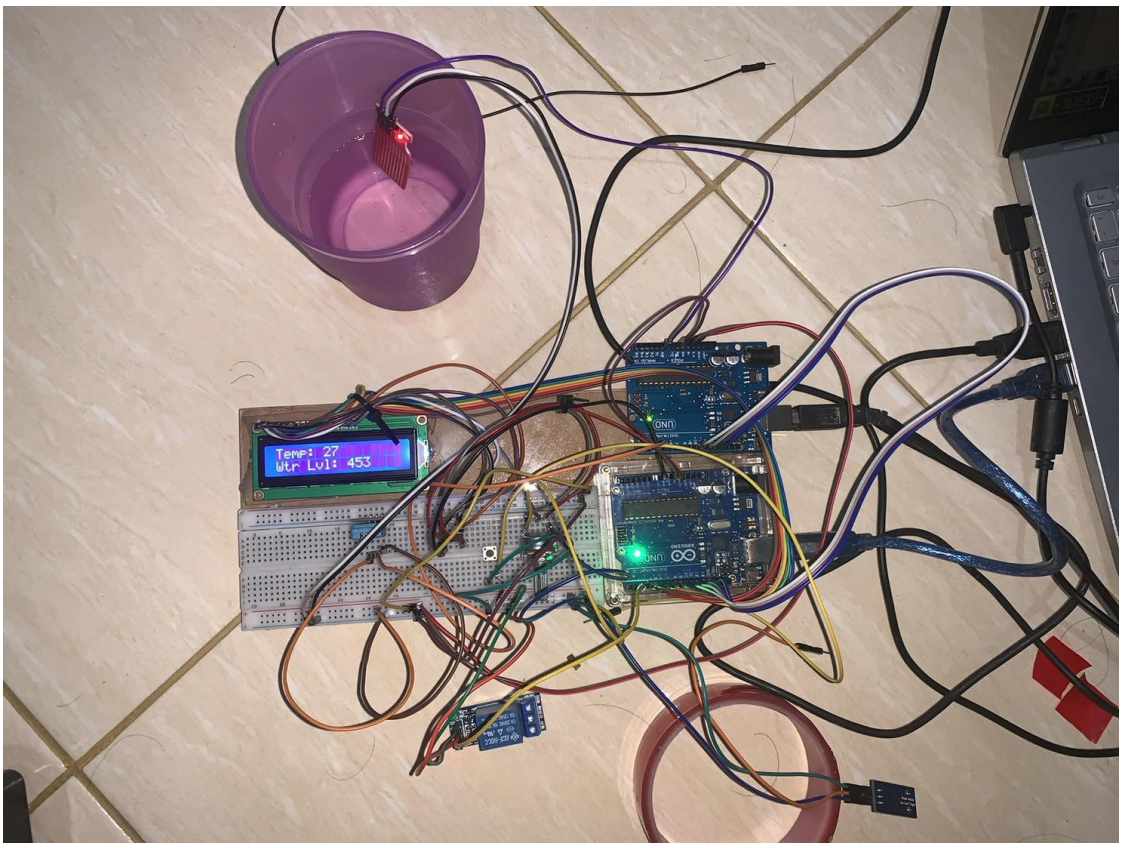


Fig 3.2 Testing Result

During this third condition test, the expected output from Thermosmart is as follows. The LED indicator turns OFF, indicating the low water level that does not requires attention. However, since the temperature is already within the desired range, the AC remains OFF. This ensures energy efficiency by not activating unnecessary cooling when the temperature is already comfortable.

During this fourth condition test, the expected output from Thermosmart is as follows. The LED indicator turns OFF, indicating the high water level that requires prompt action. Since the temperature is already within the desired range, the AC remains OFF as cooling is not necessary and stopping the water leakage from the container. This condition will also trigger a buzzer to notify the user that the drainage tank is Full.

3.3 EVALUATION

An evaluation of the Serial Peripheral Interface (SPI) communication in both Proteus and physical circuit reveals a discrepancy in their behavior. In Proteus, the SPI communication can run smoothly, allowing multiple transmissions within a loop. However, in

the physical circuit implementation, the SPI communication only sends data once, and subsequent transmissions do not occur as expected.

This inconsistency can be attributed to various factors. Firstly, there might be a discrepancy in the hardware configuration between the Proteus simulation and the physical circuit. The components used in the physical circuit, such as the microcontroller or SPI modules, may have different specifications or limitations compared to the simulated environment. This discrepancy could result in the inability to perform continuous SPI transmissions in the physical circuit.

To further enhance the efficiency and effectiveness of the smart control system, it is crucial to evaluate and improve the system's logic. The current logic may have limitations or room for optimization. By conducting a thorough analysis, it is possible to identify areas for improvement and incorporate advanced algorithms or machine learning techniques. These advancements can enable the system to adapt more accurately to occupants' preferences and create a more comfortable environment. By refining the logic, the smart control system can maximize energy efficiency and provide optimal temperature control, resulting in an enhanced user experience.

Implementing a mechanism to display the water level in percentage using assembly language further enhances the user experience. By utilizing assembly instructions, the system can accurately calculate the percentage based on the decimal representation of the water level. The display interface can then provide a clear visual representation of the current water level as a percentage. This improvement simplifies the understanding of the water level, allowing users to quickly assess whether the container requires emptying and take appropriate actions in a timely manner. Displaying the water level in a familiar decimal percentage format allows users to quickly grasp the amount of condensate water present in the storage container. This enhancement improves usability and ensures that users can easily monitor the water level without the need for additional interpretation or conversions.

Introducing a better interrupt button mechanism that enables users to forcefully turn on the air conditioner, regardless of the current temperature or system logic, enhances user control. The existing system logic may prevent the AC from operating when the temperature does not exceed predefined thresholds. However, in certain situations, users may desire immediate cooling, such as during unexpected heatwaves or emergencies. By implementing an enhanced interrupt button, users can bypass the system logic and activate the AC on

demand. It is essential to incorporate safety measures to prevent misuse or potential damage to the system. Currently, the system may deactivate the AC when the storage container approaches full capacity to prevent water overflow. However, this interruption in cooling might not always be desirable. By adjusting the system settings, the AC can continue operating, while proper management ensures efficient water discharge. This modification reduces the risk of unnecessary interruptions in cooling and maintains optimal comfort levels for occupants.

CHAPTER 4

CONCLUSION

Thermosmart offers a comprehensive solution for air conditioner temperature and condensate water management. By evaluating and refining the system's logic, incorporating advanced algorithms, Thermosmart could achieves enhanced efficiency and effectiveness in maintaining optimal temperature control. This ensures a comfortable environment for occupants while maximizing energy efficiency. Additionally, the implementation of a mechanism to display the water level in percentage using assembly language enhances usability and facilitates timely management of condensate water. With Thermosmart, users can easily monitor the water level and take appropriate actions to prevent overflow or potential damage.

Moreover, the integration of an enhanced interrupt button empowers users to have more control over the air conditioner. This allows for immediate cooling when needed, regardless of the system's predefined thresholds. Adjusting the system settings to allow the AC to continue operating even when the storage container reaches maximum capacity ensures uninterrupted cooling and mitigates the risk of unnecessary interruptions or potential flooding. With Thermosmart, users can enjoy a comfortable and controlled environment, customized to their preferences, while ensuring efficient management of condensate water. In conclusion, Thermosmart provides a user-centric and intelligent solution that enhances comfort, energy efficiency, and safety within the environment.

REFERENCES

[1] "Assembly via Arduino - DHT11 Data on MAX7219 Display," Blogspot.com, 2021. [Online]. Available: https://akuzechie.blogspot.com/2021/12/assembly-via-arduino-dht11-data-on.html. [Accessed: May 16, 2023]

[2] "Arduino® UNO R3." Accessed: May 16, 2023. [Online]. Available: https://docs.arduino.cc/static/9413481fd4a05b48e99ed972b93d2771/A000066-datasheet.pdf

[3] "EMAS2: Log in to the site," emas2.ui.ac.id. [Online]. Available: https://emas2.ui.ac.id/pluginfile.php/3797142/mod_resource/content/1/Modul%208%20SSF_%20I2C%20_%20SPI.pdf

[4] "EMAS2: Log in to the site," emas2.ui.ac.id. [Online]. Available: https://emas2.ui.ac.id/pluginfile.php/3744299/mod_resource/content/3/Modul%205%20SSF%20Aritmatika.pdf

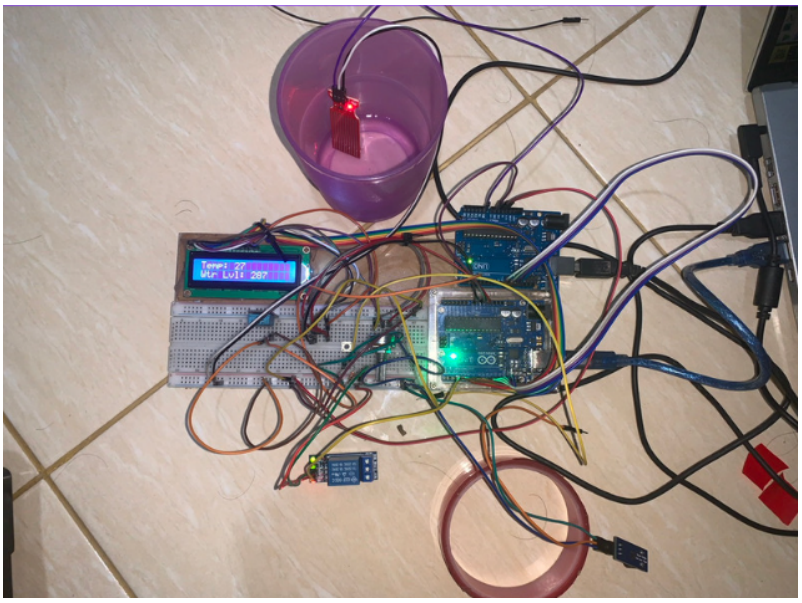
[5] "EMAS2: Log in to the site," emas2.ui.ac.id. [Online]. Available: https://emas2.ui.ac.id/pluginfile.php/3754317/mod_resource/content/1/Modul%206%20SSF_%20Timer.pdf

[6] "EMAS2: Log in to the site," emas2.ui.ac.id. [Online]. Available: https://emas2.ui.ac.id/pluginfile.php/3767711/mod_resource/content/3/Modul%207%20SSF_%20Interrupt.pdf

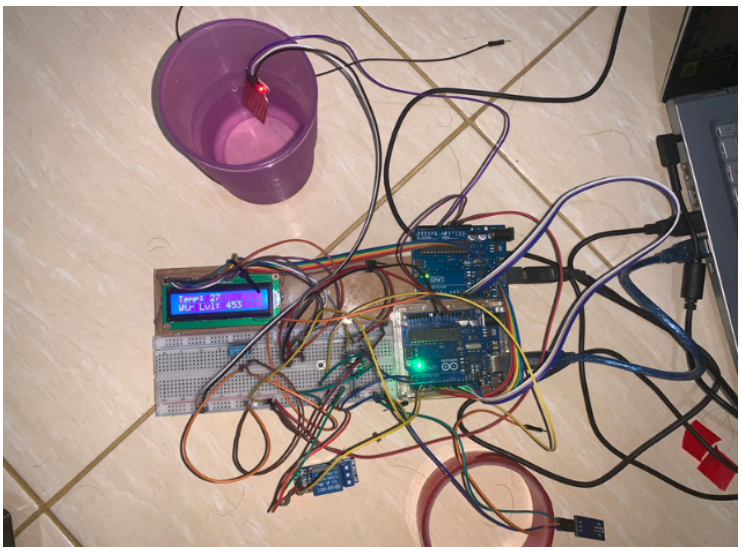
APPENDICES

Appendix A: Project Schematic

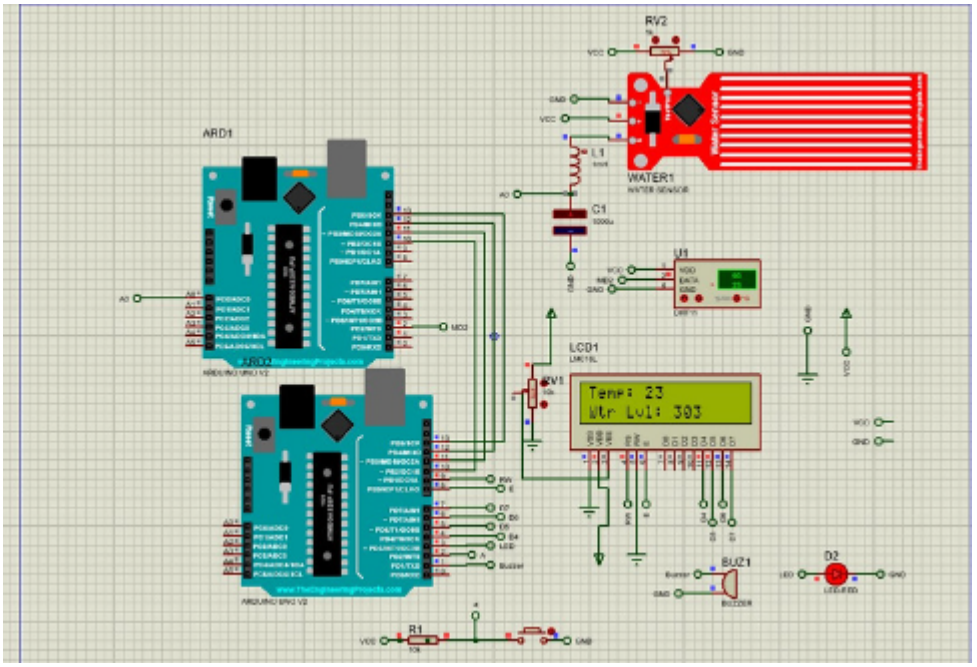
Results Low Water Level & High Temperature



Results High Water Level & High Temperature



Proteus



Appendix B: Documentation

Put the documentation (photos) during the making of the project