Smart Greenhouse

Ivan Shuba

Mahmood Mohammadi

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Program in Electrical and Automation Engineering

Project report 16-12-2025

1. Outline of the Project

This project implements a Cyber-Physical System (CPS) for a smart greenhouse environment. The system integrates sensing, computation, communication, and actuation using two bare-metal microcontroller nodes communicating via SPI and UART data flow. One node act as a **Master / Smart Edge Device**, while the other functions as a **Slave Sensor-Actuator Node.** CPS monitors environmental parameters (soil moisture, temperature, humidity, light level, and door position) and performs control actions such as fan and pump control, lamp switching, servo-based vent control and data visualization on an LCD display.

Project Components:

- Master Node: Arduino Uno with DHT11, soil moisture sensor, 12VDC fan, 12VDC water pump and LCD display
- Slave Node: ATmega328PB with light sensor, servo motor, dry contact door sensor and LED lamp
- Communication: SPI protocol with custom command structure, I2C between Master and LCD display, UART data output.
- Power Management: Sleep modes and interrupt-driven architecture

2. Introduction

2.1 Project Description

The aim of this project is to design a Smart Greenhouse CPS. The goal is to control and monitor environmental conditions in a greenhouse setting. The system employs a distributed architecture where sensors data collection, processing, and actuation are coordinated between two microcontroller nodes to optimize plant growth conditions. The project is done mostly with the bare metal programming, basically manipulating the hardware internal peripherals for optimizing the functionality and speed.

2.2 Implemented Features

Master Node (Arduino Uno) Implementation:

The system is developed using bare-metal programming and employs timer-driven periodic sampling at two-second intervals to ensure consistent data acquisition. Temperature and humidity measurements are obtained using a DHT11 sensor, while soil moisture levels are monitored through an analog sensor with ADC conversion and calibration for improved accuracy. Based on predefined soil moisture thresholds, the system automatically controls a fan to support appropriate environmental conditions. User interaction is provided through a multi-page I2C LCD display, with navigation handled via external interrupt push button. Communication with external devices is managed through an SPI master interface using a structured command-response protocol, enabling reliable and organized data exchange.

Slave Node (ATmega328PB) Implementation

The system is implemented using ultra-low-power bare-metal programming, with deep sleep operation enabled through SLEEP_MODE_DOWN to minimize energy consumption. An external interrupt (INT1) is used to wake the system by master request when activity is required. Environmental light levels are continuously monitored using the ADC, allowing the system to react to changing conditions. A PWM-controlled servo motor manages the ventilation door in 2 positions: open and closed. The door’s open and closed states are detected using dual limit switches for reliable binary sensing. In addition, the system controls an LED lamp and transmits multi-byte state packets over SPI, ensuring that system status information is communicated efficiently and accurately.

Communication Protocol:

The system uses a command-based SPI communication protocol built around eight distinct commands, allowing clear and efficient control of data exchange. To handle complex state information, the protocol supports multi-byte data transfers, ensuring that detailed system data can be transmitted reliably. Data integrity is maintained through checksum-based verification, which helps detect transmission errors. In addition, information is organized into structured data packets that include timestamps and status flags, making communication more robust, traceable, and easier to interpret.

2.3 Project Results

The implemented system successfully demonstrates autonomous environmental monitoring with a two-second refresh cycle, providing up-to-date sensor data with minimal latency. Reliable SPI communication between master and slave nodes ensures consistent and accurate data exchange throughout the system. Energy-efficient operation is achieved through the use of sleep modes, reducing overall power consumption by more than 99%. The system also delivers real-time control feedback with sub-second response timers, enabling prompt reactions to environmental changes. In addition, the modular system architecture allows individual nodes to operate and be tested independently, simplifying development and debugging. Overall, the system integrates 4 sensors and three actuators into a coordinated and fully functional embedded platform.

3. System Architecture

3.1 System Components

The system is built around a distributed architecture consisting of a master node and a slave node, each responsible for specific tasks. An Arduino Uno is used as the master or edge node, handling data processing, decision-making, user interaction, and coordination of the system. The ATmega328PB Xplained was selected as a power efficient slave node. Slave node functions, dedicated to sensor acquisition and actuator control. This separation improves modularity and simplifies testing and expansion.

Environmental data is collected using multiple sensors. Temperature and humidity are measured with a DHT11 sensor, while soil moisture and ambient light levels are monitored using analog sensors connected through the ADC. Door position is detected using limit switches, enabling reliable state detection for the ventilation mechanism.

The system controls several actuators to regulate environmental conditions. Fan and pump are controlled through digital outputs that drive optocouplers and MOSFETs, providing safe isolation and power switching for air- and water-flow management, an LED lamp is operated via SPI commands to provide supplemental lighting, and a servo motor, driven by PWM using Timer1, controls the ventilation door position.

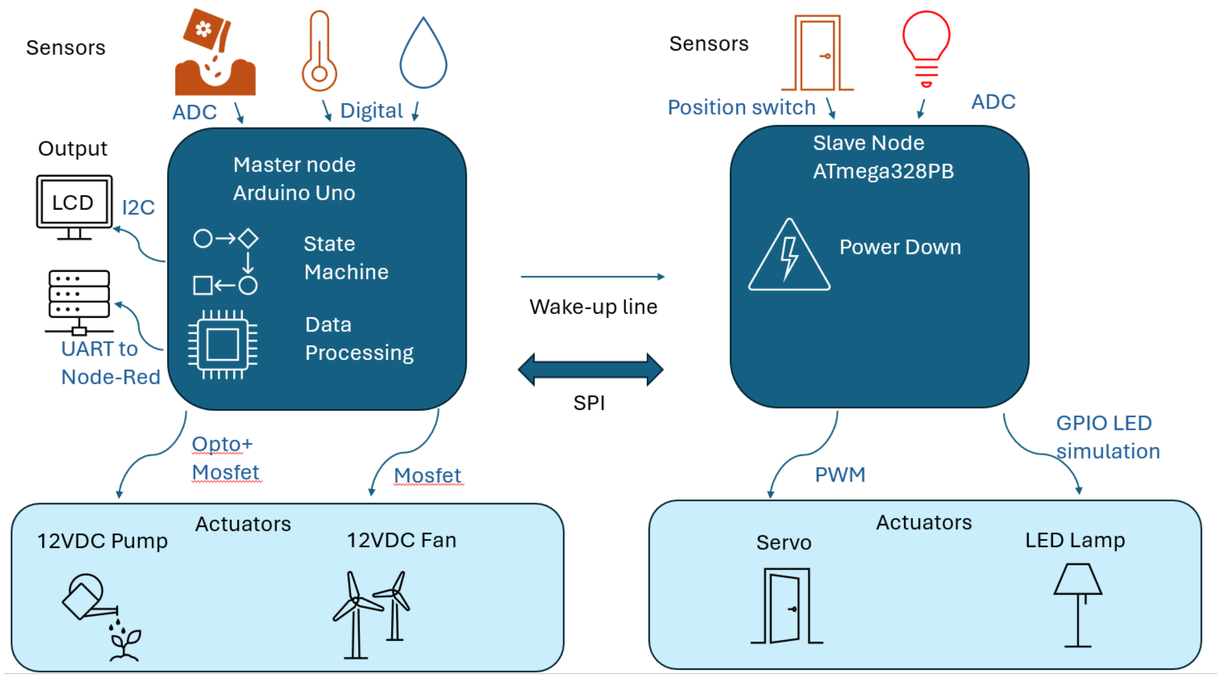
Communication between nodes is achieved using the Serial Peripheral Interface (SPI) protocol. SPI was selected due to its high speed, full-duplex capability, and reliability, making it well suited for fast command and data exchange between embedded devices. For visualization a Grove LCD connected via I2C provides real-time system information, while a UART serial monitor is used for logging and development diagnostics.

3.2 Interconnection

The master and slave nodes are interconnected through an SPI bus using the MOSI, MISO, SCK, and SS lines. Analog sensors are connected directly to the ADC channels of the microcontroller, allowing precise measurement of environmental parameters. The servo motor is driven using a PWM signal generated by Timer1, ensuring accurate position control. Door position switches are connected via GPIO pins with internal pull-up resistors, providing stable and noise-resistant digital inputs.

3.3 System Architecture Diagram

The overall system architecture follows clear data and control flow. Environmental conditions are first sensed by the sensors and processed by the slave node. The collected data is then transmitted to the master node via SPI, where processing and decision logic are applied. Based on these decisions, control commands are executes locally or sent back to the slave node using SPI, which then actuates the appropriate devices. These actions modify the physical environment, completing the control loop.



4. System Implementation

4.1 Bare-Metal Programming Approach

Both nodes in the system are developed using bare-metal C, without relying on any operating system. This approach ensures that most of the code interacts directly with hardware, providing precise control over timing and system resources. All peripheral operations are managed through direct register manipulation, which allows the system to operate efficiently and predictably. For instance, the ADC is configured using the ADMUX and ADCSRA registers, timers are set up via TCCRx and OCRx registers, GPIO pins are controlled through DDRx, PORTx, and PINx registers, and SPI communication is managed with SPCR, SPSR, and SPDR registers.

Interrupt service routines (ISRs) play a crucial role in maintaining accurate timing and responsiveness. On the master node, Timer1 operates in CTC mode to trigger periodic sampling every two seconds. On the slave node, SPI interrupts handle command reception and response processing, and an external interrupt (INT1) enables wake-up from deep sleep, supporting energy-efficient operation.

Low-level peripheral drivers enhance flexibility and modularity. A custom ADC driver allows dynamic channel selection for sensors, PWM generation via Timer1 on a slave node in Fast PWM mode controls actuators such as servo motors, a UART driver provides debugging and data output to Node-Red and CSV, and SPI master and slave drivers enable reliable node-to-node communication. This bare-metal approach ensures the system operates with high efficiency, low latency, and full control over hardware resources.

4.2 Complete System Cycle

The system operates through a structured cycle consisting of several key phases:

Initialization Phase (Power-on): Upon powering up, the master node initializes all essential peripherals, including the ADC, timers, I2C, and SPI interfaces. Meanwhile, the slave node enters deep sleep mode after completing its initialization. The LCD display shows a default "Smart Greenhouse" message, signalling that the system is ready.

Sensing Phase (Every 2 Seconds on Master): Periodic sampling is triggered by the Timer1 interrupt, which sets a sample Flag. The main loop detects this flag and initiates sensor readings. The DHT11 sensor provides temperature and humidity measurements via a single-wire protocol, while soil moisture is captured through a 10-bit ADC conversion. The raw soil moisture data undergo calibration and scaling to ensure accuracy.

Processing Phase: After data acquisition, the system calculates the soil moisture percentage using linear mapping and compares it against predefined thresholds to determine fan and pump operation. Boolean logic is applied to decide whether to activate or deactivate the fan and pump, with corresponding GPIO pins manipulated to control the actuator.

Communication Phase: Data exchange between the master and slave nodes is handled via SPI. The master pulls the SS line low and transmits a command byte (e.g., CMD_LED_ON). The slave receives the command in its SPI ISR, processes it, and prepares a response. The master then sends a dummy byte to clock out the response, and the SS line is released. A 50 µs delay ensures that the slave ISR completes its execution before the next operation.

Visualization Phase: The LCD display is cleared and updated with the latest sensor data. Users can navigate through multiple pages using button, with multi-line text rendering for all sensor values. The backlight remains constant for clear visibility.

Sleep Phase (Slave Only): After processing commands, the slave node re-enters SLEEP_MODE_PWR_DOWN, disabling all peripherals except INT1. It remains in this low-power state until a falling edge on the INT1 pin is detected, which occurs when the master asserts the SS line. This strategy significantly reduces power consumption while maintaining responsiveness.

4.3 Energy Efficiency

Through the combination of deep sleep modes, selective peripheral activation, and interrupt-driven wake-up, the system achieves substantial energy savings. By entering

SLEEP_MODE_PWR_DOWN and BOD disable when idle and waking up only, when necessary, power consumption is reduced by over 99,5% without compromising real-time performance or responsiveness. Power consumption on active sensor sampling mode 240mW and during power down about 720uW-1.5mW. Shunt resistor voltages on images below.

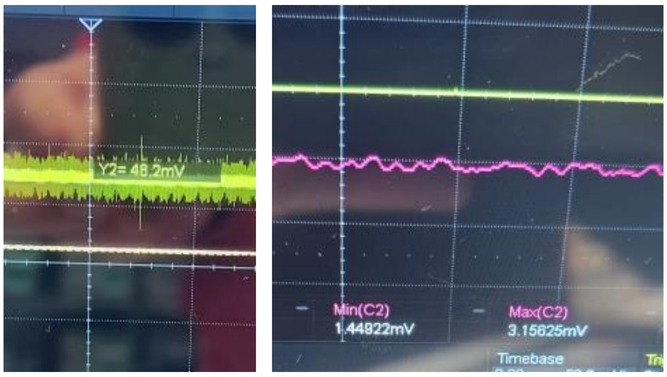


Image 1a. Shunt voltages on sampling mode. 1b. Shunt voltages on sleep mode.

4.4 Data Filtering and Data Structure

For filtering purposes, we use an exponential moving average (EMA) with a weight of 20%. This EMA is applied to temperature, humidity, and soil moisture values. Threshold detection is also implemented for the soil sensor: when the ADC value is less than 5, the system generates a warning message and deactivates the pump.

All sensor data and Boolean actuator states are combined into a single structured datatype called SensorData. This struct is shared between the master and slave nodes, with unused fields commented out in the slave implementation. When the master requests the current state, the slave generates a data package directly from this struct.

4.5 SPI Communication and Command List

Additional reason we selected SPI as the master–slave communication protocol is its independence. The master already uses I2C for the LCD and UART for data output, so SPI provides a dedicated and reliable channel for inter-node communication. During the design process, we defined eight common commands, which represent the minimum set required to cover the basic workflow. The command list is shown in Table 4.5..

Table 4.5. Command list

Command	Code — Description
CMD_LED_ON	0x10 — Turn Lamp ON
CMD_LED_OFF	0x11 — Turn Lamp OFF
CMD_DOOR_OPEN	0x20 — Activate servo to OPEN vent
CMD_DOOR_CLOSE	0x21 — Activate servo to CLOSE vent
CMD_GET_DOOR_STATE	0x30 — Checking vent State: open/closed?
CMD_GET_STATE	0x40 — Ask slave for current state. In reply it sends data package including .2-byte timestamp; Sensor data: light, lamp status, vent door position; checksum.
CMD_GET_SENSOR	0x41 — Get light sensor level
CMD_DUMMY	0xFF — Empty byte to read response

The most challenging part of this protocol was configuring the response to **CMD_GET_STATE**. It required taking into account several steps, including timing delays, sampling sensor data, preparing the data packet, and ensuring correct synchronization between master and slave.

5. Results

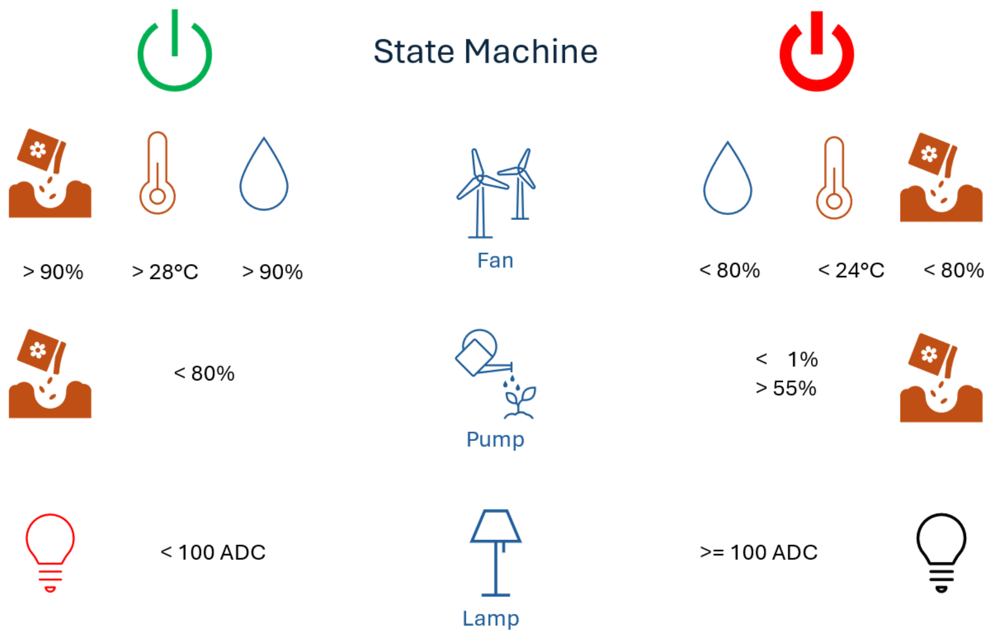
5.1 Functional Testing Results

Functional testing allows to improve weak points and confirm that the system operates reliably across all intended use cases. Sensor readings, actuator responses, and communication between nodes behaved as expected under normal and edge conditions. The system maintained autonomous operation while responding correctly to environmental changes, user interactions, and communication events. All core subsystems—including sensing, control, communication, and power management—were validated through repeated testing cycles. However, there is space for upgrade and improve.

5.2 Demonstration

State machine

Right now, we implement system with following state machine:



Energy-Efficient Operation

This scenario highlights the system's low-power design. The slave node remains in deep sleep mode, consuming approximately 150 μA, until actuation is required. Upon detecting a need for action, the master node triggers slave by external interrupt, waking the slave. Multiple SPI commands are exchanged in sequence, after which the slave executes the required actions and sends responses. Once communication is complete, the master releases the interrupt signal and the slave re-enters sleep mode. The total active time remains under two seconds every two minutes, demonstrating highly efficient power usage.

Fault Handling

Fault tolerance was evaluated by introducing sensor and communication errors.

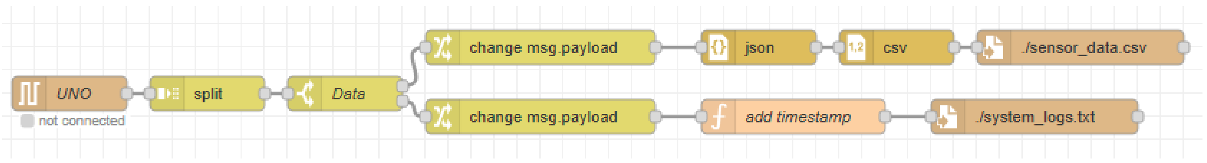
- DHT11. In the event of a DHT11 read failure producing invalid (NaN) values, the LCD displays “DHT Error” while the system continues operating using the remaining sensors.
- Soil. If soil sensor out of soil ADC shows 0% it activate pump. To avoid it pump off if soil moisture less than 1% and It send warning “Senor out of SOIL”
- Checksum. If checksum mismatch occurs during SPI communication, the master automatically sends error log messages. When master sends command to ensure reliable data transfer in reply slave sends confirmation.
- Additionally, when both door limit switches are inactive, the slave reports the door state as *DOOR_STATE_MOVING*, allowing the master to correctly interpret transitional states without halting operation.

Debug and log flag

For debug processing we have LOG_ENABLE and DEBUG_ENABLE flags which activate equal functions with UART logs flow. For example we can get message like:
“LOG:INTO:Smart Greenhouse CPS Node Loading” – In the end of master init.
“LOG:ERROR:Protocol Error: End Marker missing.” – if SPI confirmation not received.

Json type output and data flow to Node-Red

Master prepares and send data in JSON format by function sendJsonData(). Because debug and log messages also send by UART, we made split flow for different data types to be able save both. Data flow seva in CSV file sensor_data.csv, logs and debug messages with timestamp in system_logs.txt



6. Discussion

6.1 Advantages of the Implemented System

Modularity and Scalability

The system is designed in a modular way, where each node has a clear and separate role. This makes testing and debugging easier, since each node can be checked on it own. The SPI communication bus allows more slave nodes to be added without changing the overall system design. The command-based protocol is also easy to extend; although only eight commands are currently used, more can be added in the future. By separating sensing tasks from actuation tasks, the system remains organized and easier to maintain.

Energy Efficiency

Low power consumption is one of the main strengths of the system. The slave node uses a deep sleep mode that reduces power usage by more than 99% when the system is idle. The interrupt-driven design removes the need for constant polling, which further saves energy. Components are only activated when required, ensuring that actuators do not consume power unnecessarily.

Predictable Real-Time Performance

Because the system is implemented using bare-metal programming, timing behaviour is stable and predictable. Time-critical operations execute very quickly. Periodic sensor sampling is also very accurate with minimal timing variation.

Reliability

The system includes several features that improve reliability. Data integrity is protected using checksum verification during SPI communication. Sensor failures are detected and handled without stopping system operation. A watchdog timer has been implemented and can be enabled in future versions to improve fault recovery. In addition, the use of multiple door states helps prevent mechanical damage by correctly detecting movement and end positions.

Educational Value

All aside, the project provides strong educational value by combining sensing, actuation, communication, and power management in a single embedded system. It demonstrates practical use of bare-metal programming, interrupts, timers, and communication protocols, making it a useful learning platform for CPS.

6.2 Disadvantages and Limitations

Although the system performs well, some limitations were identified during development and testing. These limitations also highlight possible areas for future improvement.

One limitation is the use of basic sensors, such as DHT11, which has limited accuracy and a relatively slow update rate. In future versions, more accurate sensors could be used to improve measurement reliability. Similarly, the soil moisture sensor is sensitive to calibration drift over time, which may require periodic recalibration or replacement with a capacitive sensor. The current system uses a wired SPI connection, which limits the physical distance between the master and slave nodes. For larger deployments, wireless communication technologies such as LoRa, BLE, or Wi-Fi could be considered increase flexibility and range.

We have only one slave node is currently implemented. Future work could include adding multiple slave nodes to fully show the scalability of the SPI-based architecture. For this purpose, requires recombine nodes setups and divide them to two group “sensor node” and “actuator node”. This helps to gain more efficiency and scalability.

Power efficiency is already good, but it could be improved further. The watchdog timer can be improved. Slave sensor node can be battery powered. However, power monitoring is also not included and could be added to report battery level and trigger low-power warnings.

From a software perspective, the user interface is functional but simple. Future improvements could include extra control buttons, smoother page transitions, warning icons, or trend displays on the LCD. Logging data to external memory or sending it to a remote system could also improve long-term monitoring and analysis.

Overall, these improvements would enhance accuracy, scalability, reliability, and usability, while keeping the system aligned with its original low-power and modular design goals.

The main limitation is the variation in greenhouse types, control circuits, and power lines. Based on our understanding, each greenhouse automation project is more of a **case-specific design** rather than an out-of-the-box solution. As mentioned above, dividing the system into **“sensor nodes”** and **“actuator nodes”** can help manage this complexity.

6.2 Key updates for version 2.0

For the next version, we plan to focus on a **local indoor solution**. Instead of full greenhouse automation, the goal is to create a compact application to solve the **plant watering problem**.

- The **master node** will remain, controlling a pump and monitoring a water tank with a water-level sensor.
- The **slave nodes** will switch to **wireless communication** and be **battery powered**.
- Each slave will handle a **soil moisture sensor** and control **electromagnetic valves** for watering.
- The target system size: up to **5 pots in a row** with pipelines up to **2 meters** in length.
- Update code to modular structure by functions(sensor.c, spi-master.c and so on).

7. Conclusion

This Smart Greenhouse CPS project successfully demonstrates a complete distributed cyber-physical system, from bare-metal sensor interfacing to real-time control and inter-node communication. All main project requirements were met, including extensive bare-metal programming, energy-efficient operation using deep sleep modes, reliable SPI communication, and integrated control loops for environmental regulation.

The system combines multiple sensors and actuators across two coordinated nodes resulting in a practical and functional greenhouse automation protype. Implementing deep sleep modes and interrupt-based wake-up, makes the system energy efficient. The Master-Slave implementation is effective for separating high-power actuator control from sensitive sensor monitoring.

The project highlights the importance of reliable communication protocols in distributed systems and shows how simple microcontrollers can perform complex automation tasks. The modular design provides foundation for future enhancements, making this system a practical solution for small-scall automated agriculture.

8. References

- Course Materials
- AI & google tools

Microcontroller Datasheets:

- Microchip ATmega328P/PB
- ARM Cortex-M Programming Manual

Sensors Datasheets:

- DHT11 Humidity & Temperature Sensor Datasheet
- Soil Moisture Sensor Datasheet
- LDR Datasheet

Online Resources:

- Embedded System Course Materials – UC Berkeley
- Youtube tutorials for Serial Communication protocols

9. What have I learned?

This project taught me far more than just how to connect sensors and write code. It gave me a clear picture of how CPS actually works in the real world.

I learned how to work directly with AVR registers instead of using high-level Arduino functions, I gained a much deeper understanding of how microcontrollers really operate. Configuring timers, ADCs, GPIOs, and communication peripherals manually made the hardware feel much less like a black box. I also learned how important interrupts are in embedded systems. Designing an interrupt-driven system showed me why ISRs must be short, efficient and how timing and priority directly affect system behavior. Using hardware timers instead of software delays taught me how precise and reliable timing should be handled.

Implementing communication between two nodes was another major learning experience. Working with SPI helped me understand synchronous communication, master-slave roles, and how clocking and timing really work in practice. Designing a state handling, and documentation are. I also learned that real systems rarely work perfectly on the first try, small delays and timing adjustments were needed to make communication reliable, even when the protocol looked correct on paper.

Power management was an eye-opening part of the project. Implementing sleep modes and wake-up mechanisms showed me how much power can actually be saved when a system is designed carefully.

I learned how to interface sensors and actuators. With analog sensors, I learned that raw ADC values are meaningless without proper calibration and scaling. Working with the DHT11 highlighted how timing-sensitive digital protocols can be fragile and unreliable. On the actuator side, generating PWM signals for servo control helped me understand how timer configuration directly affects resolution and movement accuracy. Even simple GPIO control taught important lessons about current limits and safe hardware operation.

From a system-level point of view, the project improved my understanding of distributed architectures. Coordinating multiple nodes made me aware of communication delays, failure cases, and the limits of master–slave designs. I also learned that real-time systems are not about being fast, but about being predictable. Identifying critical code paths and ensuring consistent timing became more important than raw speed.

Error handling turned out to be more important than expected. I learned that embedded systems must continue operating even when things go wrong, such as sensor failures or communication errors. Detecting errors is only useful if the system knows how to respond safely and continue running.

The development process itself taught many practical lessons. Debugging embedded systems required different tools and approaches than software-only projects. UART logging became essential, and using measurement tools like oscilloscopes helped reveal timing issues that were invisible in code. I also learned the importance of building systems step by step — testing each module individually before integrating everything saved time and frustration in the long run. Good documentation proved to be critical, not just for others, but for understanding my own code weeks later.

Beyond technical skills, the project helped me grow personally. I developed a more structured problem-solving approach by breaking complex problems into smaller parts and testing ideas systematically. I became better at managing time and learned to leave room for unexpected issues, especially hardware-related ones. Attention to detail became crucial, as even a single wrong bit or timing miscalculation could break the entire system.

Finally, the project helped bridge the gap between theory and practice. Concepts like interrupts, ADCs, PWM, and communication protocols became real and concrete through hands-on work. I also learned that engineering is always about trade-offs — between power and performance, simplicity and flexibility, and cost and reliability. While bare-metal programming is extremely valuable for learning, it also gave me a new appreciation for abstraction layers and operating systems in real commercial products.

Overall, this project was challenging, sometimes frustrating, but extremely rewarding. It strengthened my technical skills, improved my problem-solving ability, and gave me confidence in working with complex embedded systems from the ground up

10. Recommendations and Feedback

For future students, I would recommend starting with a simple working prototype and gradually add more features instead of building everything at once. All pin connections should be documented immediately to avoid confusion later. Adding basic debugging outputs, such as UART messages or status LEDs, from the beginning can save a lot of time during troubleshooting. Communication should be tested separately before full system integration. Correct filtering and calibration could improve sensor accuracy.