Design and Implementation of a RISC-V-Based Smart Pot System

First A. Mohammad YahyaPour, Second B. Mohammad Moein Joneidi Jafari

Abstract—**This paper details the design, implementation, and verification of a comprehensive embedded system for a smart pot, aimed at optimizing plant irrigation based on environmental conditions and user-defined parameters. The hardware architecture is centered around an AFTAB RISC-V processor, which serves as the central processing unit. This core is connected via a system bus to a suite of memories and peripherals, including an SPI Flash-based instruction memory, an SRAM data memory, an I2C humidity and temperature sensor, a custom floating level sensor for the water reservoir, a pump controller, and a UART serial communication interface. The system's control software, implemented in C++, features an intelligent algorithm that dynamically calculates the optimal pump activation duration by considering real-time humidity and temperature data. A key feature of the system is its ability to update the target moisture threshold remotely via UART, a process managed by an interrupt service routine. The entire system's functionality was validated in a comprehensive Verilog simulation environment, confirming its operational correctness across various scenarios.**

Index Terms—**Embedded systems, hardware-software codesign, I2C, RISC-V, smart agriculture, System on Chip (SoC), UART, Verilog.**

I. INTRODUCTION

Manual plant care is a time-consuming and error-prone process, where improper watering is a primary cause of poor plant health. Smart systems can address this challenge by automating the irrigation process, ensuring optimal water delivery, which not only promotes plant vitality but also conserves water resources.

The primary objective of this project is the design and implementation of a complete embedded system for a "Smart Pot". The central challenge lies in integrating a processing core with a diverse set of sensors, actuators, and peripherals into a cohesive System on Chip (SoC), and co-designing the hardware and software to achieve the desired functionality.

The system developed herein is built around the AFTAB RISC-V processor. It continuously monitors soil moisture and ambient temperature using an I2C sensor and tracks the reservoir's water level via a custom floating level sensor (FLS). The C++ control software analyzes this sensor data and, based on a configurable moisture threshold, decides when to activate a water pump. The pump's runtime is intelligently calculated based on the degree of soil dryness and ambient temperature to ensure efficient watering. Furthermore, the system features a UART interface for external communication, allowing the user to remotely adjust the target moisture threshold. This dynamic reconfiguration is enabled by an Interrupt Service Routine (ISR) that processes incoming serial data without halting the main control loop.

This report first details the hardware architecture and the design of its constituent modules. It then examines the software algorithm and its implementation. Finally, it presents the simulation results that validate the system's integrated performance.

II. EMBEDDED SYSTEM DESIGN

The system design encompasses two concurrent development tracks: hardware design and software implementation, ensuring seamless integration and correct functionality of the final product.

A. System Hardware Architecture

The hardware is architected as an SoC where the AFTAB processor core communicates with memory and peripherals through a central bus module (

Bus.v). This modular structure facilitates design and verification.

1) Central Processing Unit (CPU): The AFTAB processor, based on the RISC-V instruction set architecture, serves as the system's computational core. It is responsible for executing the control software, managing interrupts, and directing all peripheral operations. As this core was a pre-existing IP, the project's focus was on its successful integration into the broader SoC framework.

2) Memory Subsystem: The system includes two primary types of memory:

- *Instruction Memory:* A virtual SPI Flash memory is used to store the program code. It is accessed via an SPI controller (

  `spi_flash_controller.vhd`). The `virtual_flash.v` module simulates the Flash memory, loading the compiled machine code from a `mem.mem` file at the start of the simulation. The system bus maps this memory to the address space below

  `0x00100000`.

- *Data Memory:* A 4KB SRAM provides storage for variables, the stack, and other runtime data. It is mapped to the base address

  `0x00100000`.

3) Peripherals: All peripherals are memory-mapped, with their address decoding managed by the `Bus.v` module.

- *Humidity and Temperature Sensor (HTS):* This unit uses the I2C protocol for communication. An

  `i2c_controller` module acts as the bus master. The sensor itself is simulated by an

  `i2c_slave_controller` that generates random values within a predefined range to mimic real-world readings. This peripheral is mapped to the

  `0xFFFFFF00` address region.

- *Floating Level Sensor (FLS):* This custom-designed peripheral (`floatingSwitch.v`, `FLSWrapper.v`) simulates the water level in the reservoir. The model is designed such that the water level decreases when the pump is active. The FLS is mapped to address

  `0xFFFF0000`.

- *Pump Controller:* A simple wrapper module (`pumpWrapper.v`) controls the water pump. The software activates the pump by writing a '1' to address

  `0xFFFF0F00` and deactivates it by writing a '0'.

- *Timer:* A 32-bit timer provides a mechanism for generating precise software delays. The software writes a target count value to the timer and polls an interrupt status bit to wait for the specified duration to elapse. The timer is mapped to the base address

  `0xFFFFF000`.

- *UART Communication Interface:* This interface consists of separate transmitter (`transmitter.v`, `TX_wrapper.v`) and receiver (`reciever.v`, `RX_wrapper.v`) modules. The receiver is configured to assert an interrupt request to the CPU upon the successful reception of a byte. The UART TX and RX modules are mapped to base addresses

  `0xFFF10000` and `0xFFF00000`, respectively.

B. Algorithmic and Software Implementation

The system's control logic is implemented in C++ and compiled for the RISC-V architecture.

1) Main Control Loop: The core logic operates within an infinite `while(1)` loop. In each iteration, the program first waits for a fixed interval (`CHECK_TIME`). It then reads the current temperature, humidity, and water level by accessing their respective memory-mapped addresses. If the water level is critically low, the system is designed to issue a warning. Otherwise, it proceeds to call the watering duration calculation function.

2) Watering Algorithm: The `calculate_pump_duration_ms_integer` function contains the system's core intelligence.

- If the current soil moisture is above the user-defined `MOISTURE_THRESHOLD`, the function returns zero, and the pump remains inactive.
- Otherwise, it computes two factors using fixed-point arithmetic to avoid the need for a floating-point unit:
  - *Moisture Factor:* This factor is proportional to the soil's dryness, calculated as `((THRESHOLD - current) * SCALE) / THRESHOLD`. Drier soil results in a larger factor.
  - *Temperature Factor:* This factor adjusts the watering duration to account for ambient temperature. If the temperature exceeds a baseline (`BASE_TEMP_C`), the watering time is increased to compensate for higher evaporation rates.
- The final pump duration is determined by multiplying a base duration (`BASE_PUMP_DURATION_MS`) by these two factors.

3) Interrupt Handling: The system's ability to be reconfigured dynamically is handled by an ISR.

- During initialization, the software configures the CPU's control registers (`mstatus`, `mie`, `mtvec`) to enable machine-level external interrupts and sets the vector table address to point to the ISR.
- When the UART receiver gets a new byte of data, it asserts an interrupt line to the CPU.
- The CPU suspends the main loop and executes the ISR, `m_mode_external_interrupt_handler`. This function reads the newly

received byte from the UART data register and updates the global `MOISTURE_THRESHOLD` variable.
- This mechanism allows the user to recalibrate the system's target moisture level on-the-fly, providing significant operational flexibility.

III. EXPERIMENTAL RESULTS

To validate the integrated system, a comprehensive simulation environment was developed using a top-level Verilog testbench (

`testbench.v`). This testbench instantiates the entire SoC, including the AFTAB core and all connected peripherals.

A. Basic Functionality Verification

A test scenario was created where initial sensor values were set to a state requiring irrigation (e.g., humidity at 50, below a threshold of 60, and temperature at 30°C, above the 25°C baseline).

- *Expected Behavior:* The software should read the sensor values, determine that the soil is dry, and calculate a non-zero pump duration. It should then activate the pump for this duration while loading the timer with the same value. Upon completion, the pump should be deactivated.
- *Simulation Results:* Analysis of the simulation waveforms confirmed the expected behavior. The

  `memWrite` signal was asserted for the pump controller's address (`0xFFFF0F00`), causing the `pump_activated` signal to go high for the calculated period. Simultaneously, data corresponding to the pump duration was written to the UART TX data register for transmission. These results validate the correctness of the main control loop and the watering algorithm.

B. Interrupt Mechanism Verification

A second scenario was designed to test the interrupt-driven reconfiguration feature.

- *Expected Behavior:* While the main loop is running, the testbench sends a new moisture threshold value (e.g., 75) to the system via the serial input pin. The UART receiver should trigger an interrupt. The CPU should then execute the ISR, read the new value, and update the `MOISTURE_THRESHOLD` variable. In the next control loop iteration, the watering algorithm should use this new threshold for its calculations.
- *Simulation Results:* The simulation waveforms showed that upon receiving serial data, the `uart_interrupt` line was asserted, causing the `machineExternalInterrupt` input to the core to go high. This action resulted in a context switch to the ISR, which was verified by observing a memory read from the UART's data register address (

  `0xFFF00008`). Subsequent pump activation cycles were correctly based on the new threshold of 75, confirming that the interrupt-driven reconfiguration mechanism functions as designed.

IV. CONCLUSION

This project successfully demonstrated the design, implementation, and verification of a complete embedded system for a smart pot. By integrating a RISC-V processor with a set of custom and standard peripherals, a capable and flexible hardware platform was created. The C++ control software, featuring an intelligent watering algorithm and an interrupt-driven interface for remote configuration, provides the system with robust functionality. Comprehensive simulation results have validated the correct, integrated operation of the hardware and software components, confirming that the project has successfully met its design objectives.

Future work could focus on extending the system's capabilities by incorporating IoT connectivity (e.g., Wi-Fi), allowing for full remote control and data logging. The software algorithm could also be enhanced with machine learning techniques to learn the specific watering needs of different plants over time.

REFERENCES

[1] "Core-Based Embedded System Design, Project - Part C: System Implementation," ECE 160, University of Tehran, Spring 1403-1404.

[2] mitya1337, *Simple_I2C*, GitHub repository, 2025. [Online]. Available: https://github.com/mitya1337/Simple_I2C.

[3] Russell, *UART Serial Port Module*, Nandland. [Online]. Available: https://nandland.com/uart-serial-port-module/

[4] Silicon Laboratories, *Si7021-A20 I²C Humidity and Temperature Sensor Data Sheet*, Rev. 1.3, June 2022. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf.

[5] ONNX, *Open Neural Network Exchange (ONNX) Library*, GitHub repository. [Online]. Available: https://onnx.ai.