EE446 Term Project Report
Middle East Technical University

Zülal Uludoğan,

Ahmet Taha Çelik, 2515831

25 May 2025

Contents

1 Introduction **3**
1.1 Project Description 3

2 Datapath Module **4**

3 Controller Module **5**

4 UART Module **7**
4.1 Clock Domain Challenges and FIFO Synchronization 8
4.2 UART Transmitter and Receiver 8
4.2.1 Receiver State Machine 9
4.3 Peripheral Interface and Muxing 9

5 Cocotb Testbench **10**

Chapter 1

Introduction

1.1 Project Description

Chapter 2

Datapath Module

Chapter 3

Controller Module

The controller module is responsible for generating the control signals necessary to guide the datapath during the execution of RISC-V instructions. As shown in Figure 3.1, the controller design follows a modular approach and includes three functional units: the `MainDecoder`, the `ALUDecoder`, and the `PCLogic` unit.

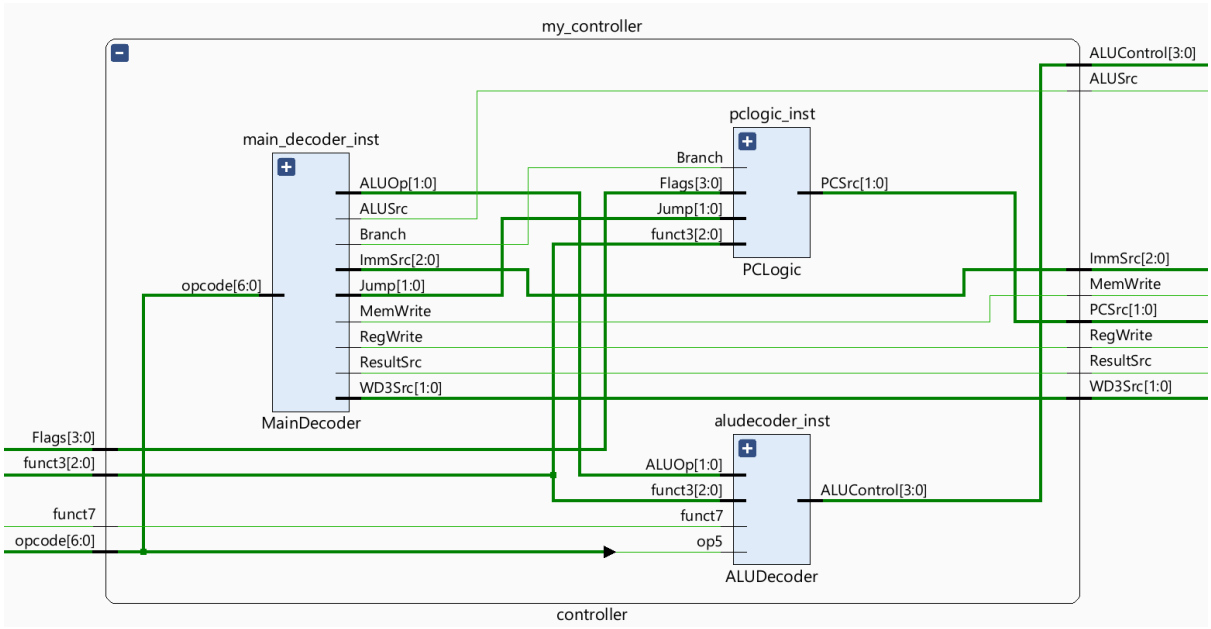

Figure 3.1: Synthesized RTL view of the controller module

The `MainDecoder` interprets the 7-bit opcode from the instruction and produces the primary control signals required by the datapath, such as register file write enable, memory write enable, ALU source selection, and immediate format selection. These signals are then distributed to the corresponding components to orchestrate the processor's behavior.

The `ALUDecoder` further refines control signal generation by examining the `funct3`, `funct7`, and other opcode-specific bits to determine the specific ALU operation to perform. It outputs a 4-bit `ALUControl` signal, ensuring the correct arithmetic or logical operation is selected based on the instruction type.

The `PCLogic` unit handles program counter selection. It considers both the branching conditions and the jump instructions to compute the correct value of the `PCSrc` signal. This logic is critical for implementing both conditional and unconditional control-flow changes in the processor.

The modularity of this controller design simplifies debugging and testing. Each submodule is implemented independently and tested in isolation before integration into the main controller unit.

Chapter 4

UART Module

The UART (Universal Asynchronous Receiver-Transmitter) module is responsible for facilitating serial communication between the RISC-V processor and external systems. Its implementation involved significantly more complexity than the controller module due to the nature of asynchronous communication, timing constraints, and the need to bridge two separate clock domains. Figure 4.1 shows the synthesized RTL view of the UART subsystem.

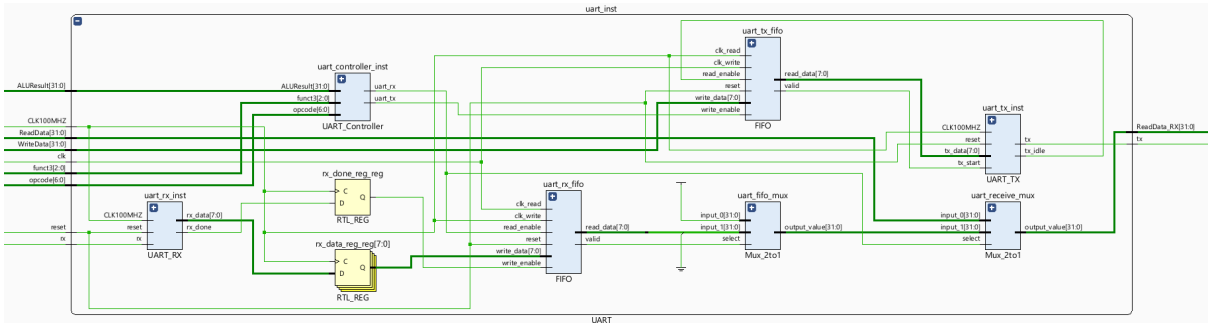

Figure 4.1: Synthesized RTL view of the UART module

Design Overview

The UART module integrates both transmitting and receiving functionalities while adhering to the 9600 baud 8-N-1 protocol. Memory-mapped I/O is used to send and receive bytes via store and load instructions to addresses `0x400` and `0x404`, respectively. When a store is made to `0x400`, a byte is written to the transmitter FIFO and scheduled for transmission. When a load is performed from `0x404`, the system reads the most recent byte from the receive FIFO; if no byte is available, the value `0xFFFFFFFF` is returned.

The UART controller examines the current instruction's opcode, `funct3`, and ALU result to determine whether a UART send or receive should be triggered. This controller outputs `uart_tx` or `uart_rx` flags accordingly.

4.1 Clock Domain Challenges and FIFO Synchronization

One of the key challenges in designing the UART module was handling the presence of two independent clocks: a 100 MHz clock (`CLK100MHZ`) required for UART transmission timing, and a manually generated system clock (`clk`) used by the rest of the processor, including the register file and instruction control. Since UART must transmit data at precise intervals defined by the baud rate, it must operate continuously using `CLK100MHZ`. However, if both the UART and the register file accessed the same FIFO at their own clock rates without isolation, it would introduce race conditions and data corruption.

To address this, the design includes dual-clock FIFOs that are explicitly controlled with separate clock signals: `clk_write` and `clk_read`. The write side of the FIFO operates on `CLK100MHZ`, synchronized with `UART_RX` and `UART_TX` modules, while the read side operates on the processor's `clk`, ensuring that data transfer from UART into the register file occurs safely. This separation of clock domains is essential to maintain system stability and correctness across asynchronous boundaries.

4.2 UART Transmitter and Receiver

The transmitter module initiates data transmission when a byte is enqueued into the FIFO and the line is idle. It serializes the data byte according to UART protocol and asserts the `tx_idle` signal once transmission is complete. If the FIFO holds multiple bytes, they are transmitted sequentially without processor intervention.

The receiver module continuously samples the input `rx` line using `CLK100MHZ`. When a valid start bit is detected, it begins capturing bits until the full byte is received. A `done` flag is raised once reception is complete. The received byte is then pushed into the 16-byte FIFO buffer, ensuring that no bytes are lost if the processor is not immediately ready to read.

UART Transmitter and Receiver State Machines

Both the UART transmitter and receiver modules are implemented as finite state machines (FSMs) that operate on the 100 MHz clock and comply with the 9600 baud 8-N-1 UART protocol. Their behaviors are structured as sequential state transitions, and the FSM logic ensures the correct serialization and deserialization of data.

Transmitter State Machine

The transmitter FSM begins in the `IDLE` state, where it waits for the `start` signal to be asserted, indicating that a new byte is ready for transmission. Upon activation, it transitions to the `START` state and drives the transmission line low to indicate the start bit. After holding the line low for one baud period, the FSM enters the `DATA` state, during which each of the 8 data bits is shifted out sequentially, least significant bit first.

Once all data bits are transmitted, the FSM transitions to the `STOP` state. It then drives the line high for one baud period to signal the stop bit. After this, the FSM returns to the `IDLE` state, sets the `idle` flag to indicate readiness, and awaits the next start signal.

This structure ensures that every byte is framed with a start and stop bit as per UART standards, and the `idle` output allows external logic to determine when the transmitter is available for the next byte.

4.2.1 Receiver State Machine

The receiver FSM also begins in an `IDLE` state, continuously monitoring the input `rx` line. Upon detecting a falling edge (start bit), it transitions to the `START` state and waits half a baud period to sample the start bit in the middle for noise immunity.

After verifying the start bit, it transitions to the `DATA` state, where it samples 8 data bits at one baud interval each. These bits are shifted into a register, with the least significant bit arriving first.

Following the data bits, the FSM enters the `STOP` state to sample the stop bit. If the stop bit is valid (logic high), the FSM sets the `done` flag and updates the `data_out` register with the fully received byte. It then returns to the `IDLE` state to prepare for the next frame.

This approach ensures that bytes are received reliably and with proper framing. The `done` signal is used to indicate reception completion, allowing external logic to push the received byte into a FIFO buffer or register.

4.3 Peripheral Interface and Muxing

The UART module includes additional logic for interfacing with the processor. When a load operation targets the UART receive address, a multiplexer determines whether valid data exists in the FIFO. If not, it returns `0xFFFFFFFF`, as required by the design specification. Similarly, when a store operation targets the transmit address, a different multiplexer ensures that the byte written by the processor is passed correctly to the transmit FIFO, provided the transmitter is not currently busy.

With careful clock domain handling, FIFO synchronization, and protocol adherence, the UART module operates as a robust peripheral enabling reliable serial communication within the single-cycle RISC-V system.

Chapter 5

Cocotb Testbench