FYDP PRESENTATION

Department:
Telecommunications Engineering

By:
Huzaifa Hassan TC-21060
M. Kashaf Khan TC-21065
Rao M. Umer TC-21073

Contents

Introduction

A ***System-on-a-Chip (SoC)*** is a single chip that contains all the components of a system, such as a central processing unit (CPU), memory, and I/O ports.
This project involves creating a SoC that contains the integration for a ***RISC V Processor*** as the main CPU that can communicate with peripheral devices through ***UART*** and ***I2C***.

Gantt Chart

Year	2024 - 2025									
Months	**Aug. 2024**	**Sep. 2024**	**Oct. 2024**	**Nov. 2024**	**Dec. 2024**	**Jan. 2025**	**Feb. 2025**	**Mar. 2025**	**Apr. 2025**	**May. 2025**
TASKS										
Literature review for project	■	■			■	■		■	■	
Complete RISC V Single Cycle Processor			■	■	■					
Understanding and Simulation of modules				■	■	■	■			
Implementation of Protocols						■	■	■	■	
Verification using ModelSim							■	■	■	■
Architecture development and Debugging									■	■
Report writing	■	■	■	■	■	■	■	■	■	■

SDGs

8 DECENT WORK AND ECONOMIC GROWTH

- The project supports economic growth by promoting advanced technology solutions in sectors like industrial automation, IoT, medical devices, and wearable technologies.
- By improving efficiency, productivity, and automation, it contributes to sustainable industrial development and helps create high-tech job opportunities in engineering, embedded systems, and digital design fields.

9 INDUSTRY, INNOVATION AND INFRASTRUCTURE

- The project directly aligns with SDG 9 by fostering innovation in embedded systems and digital hardware design.
- It promotes modern industrial automation, smart medical devices, and IoT solutions, thereby enhancing technological infrastructure and supporting sustainable industrialization through innovation.

12 RESPONSIBLE CONSUMPTION AND PRODUCTION

- By utilizing energy-efficient designs (e.g., low-power processors like RISC-V, optimized communication with I2C), the project encourages responsible resource use.
- It also contributes to sustainable production processes in embedded technology, focusing on reducing waste, optimizing power usage, and improving device efficiency.

Methodology

Literature review

Designing Modules

Software Implementation

Synthesis and Debugging

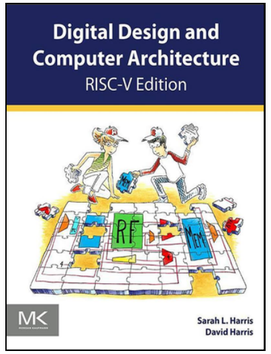
Literature Review

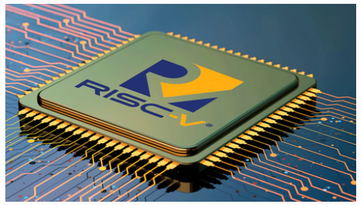
01 Utilized primary and secondary sources for data gathering.

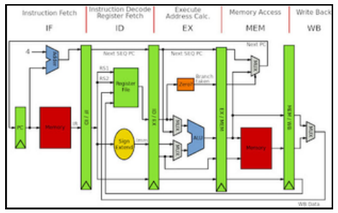
02 Comprehensive exploration of RISC-V processors

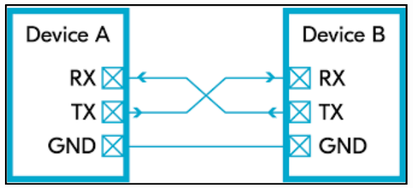
03 Pipelining in processors

04 UART and I2C Protocols, and interfacing components with processors

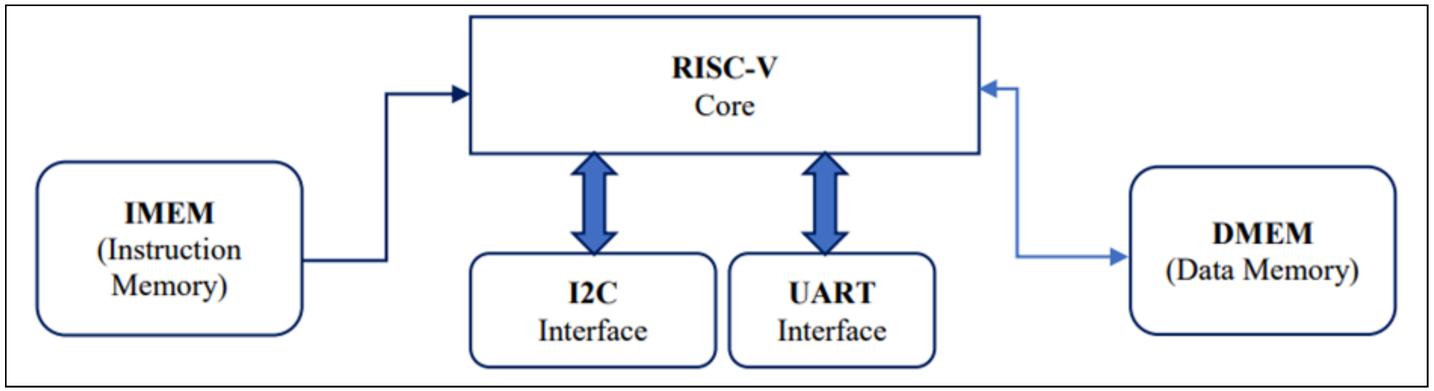








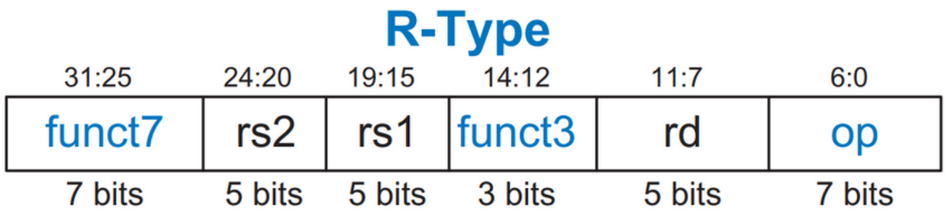
System Overview



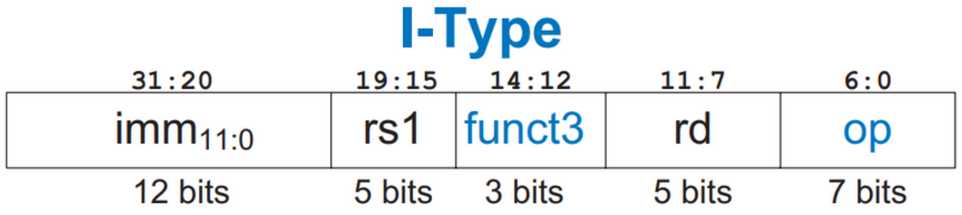
- Our main objective is to interface the **RISC V Processor** with **UART** and **I2C** by means of transmitting data to processor and then retrieving the results from it.
- Here, **UART** behaves as an **Input** interface that carries the data from user controller to the processor.
- The processor then evaluates the instructions and produce the results.
- The results will be driven to user by **I2C** as it behaves here like an **Output** interface.
- The **IMEM** is a memory block that contains the instructions to be executed inside the processor.
- The **DMEM** is also a memory block that behaves as a temporary storage block to hold the intermediate data for the instruction that is currently being processed.

RISC V Instructions Format

Register (R-Type) Instructions

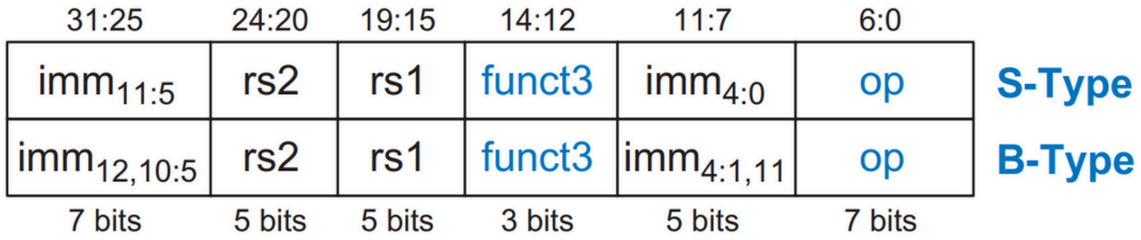


Immediate (I-Type) Instructions

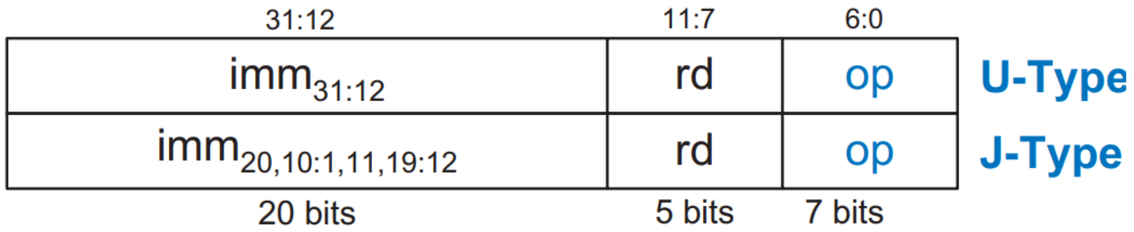


RISC V Instructions Format

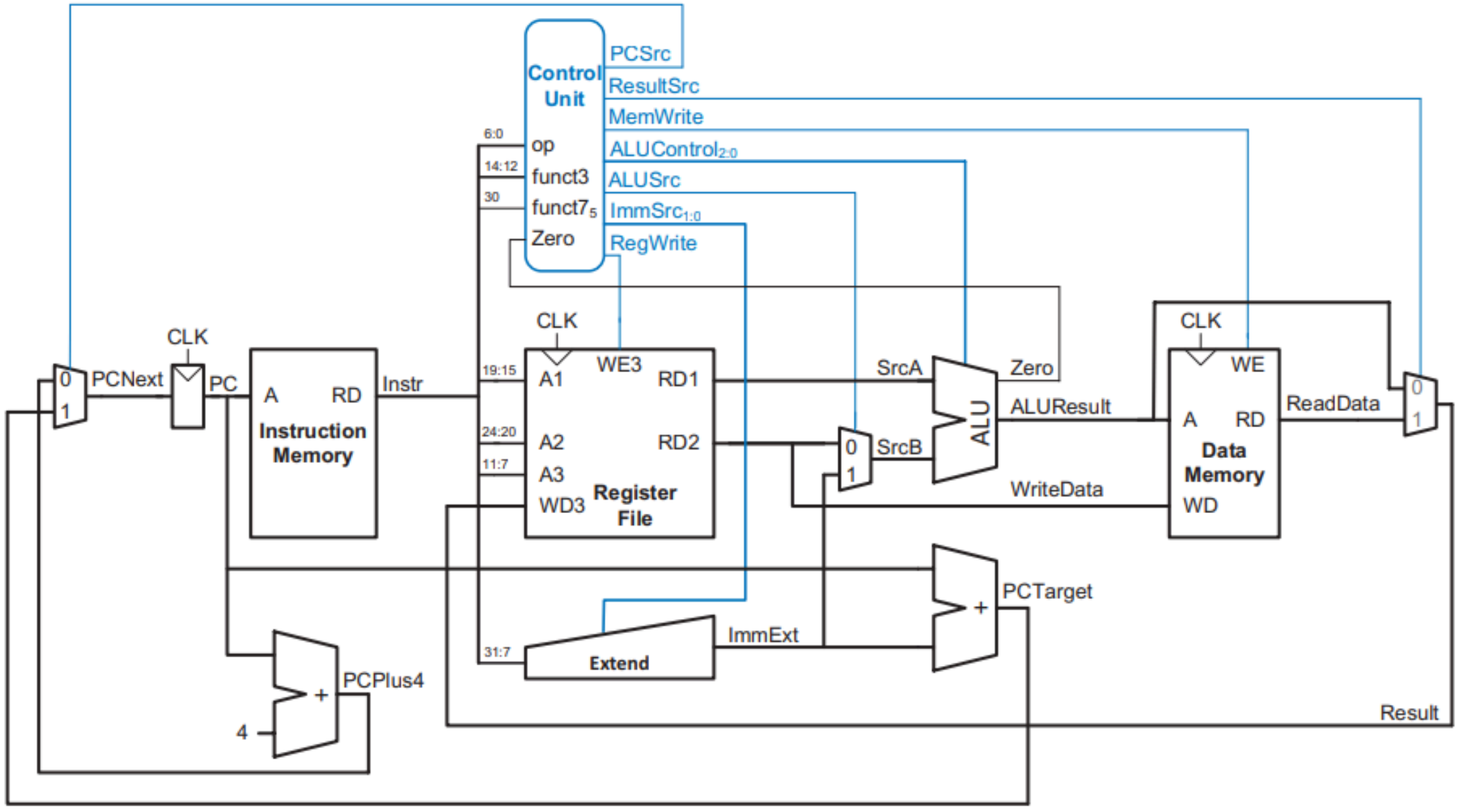
Store/Branch (S/B-Type) Instructions



Upper Immediate/Jump (U/J-Type) Instructions



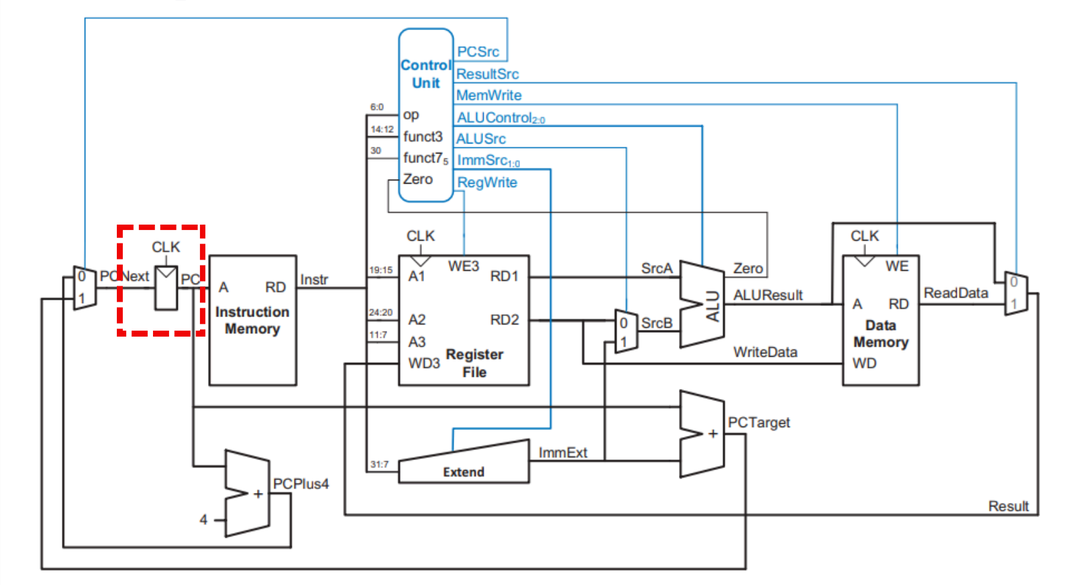
Single Cycle Processor



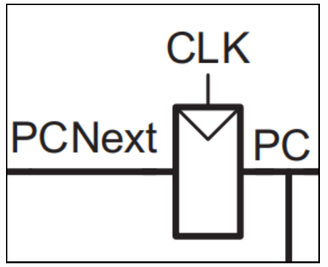
- This is a Complete Single Cycle RISC V Processor.
- We'll discuss all it's elements and datapath in the following slides.

Single Cycle Processor

Internal Components

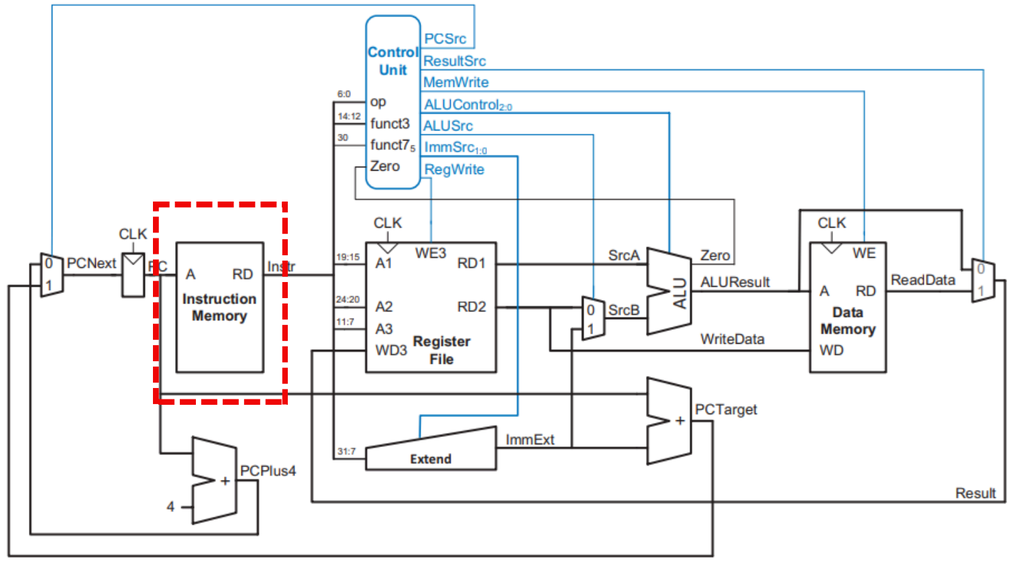


- **Program Counter**: The program counter (PC) points to the current instruction. Its input, PCNext, indicates the address of the next instruction.

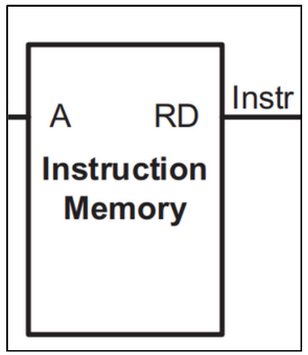


Single Cycle Processor

Internal Components

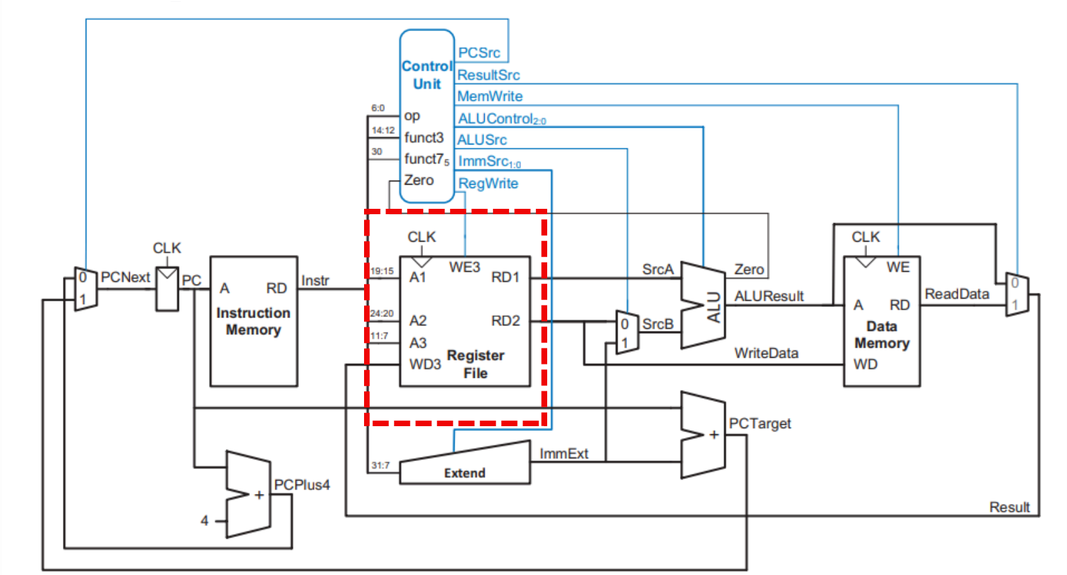


- **Instruction Memory**: It has a single read port. It takes a 32-bit instruction address input, A, and reads the 32-bit data (i.e., instruction) from that address onto the read data output, RD.

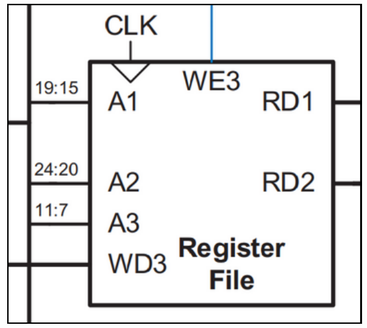


Single Cycle Processor

Internal Components

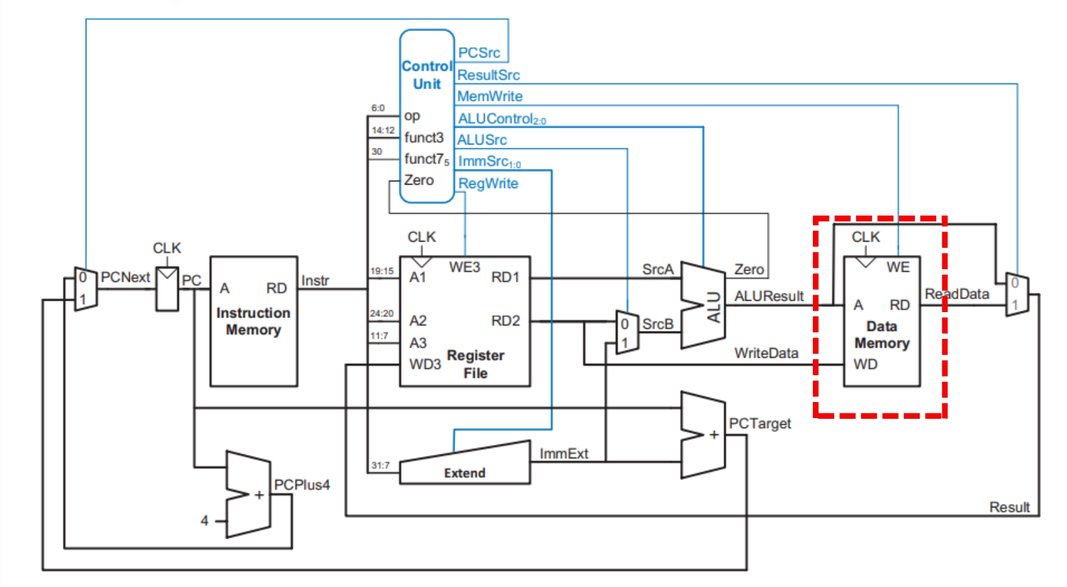


- **Register File**: The 32-element × 32-bit register file holds registers x0–x31. The register file has two read ports and one write port. The read ports take 5-bit address inputs, A1 and A2, each specifying one of the 25 = 32 registers as source operands. The register file places the 32-bit register values onto read data outputs RD1 and RD2. The write port, port 3, takes a 5-bit address input, A3; a 32-bit write data input, WD3; a write enable input, WE3; and a clock. If its write enable (WE3) is asserted, then the register file writes the data (WD3) into the specified register (A3) on the rising edge of the clock.

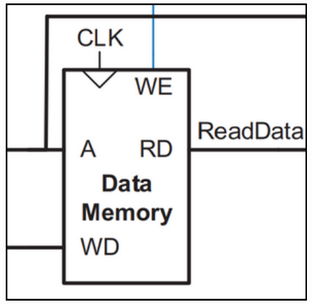


Single Cycle Processor

Internal Components

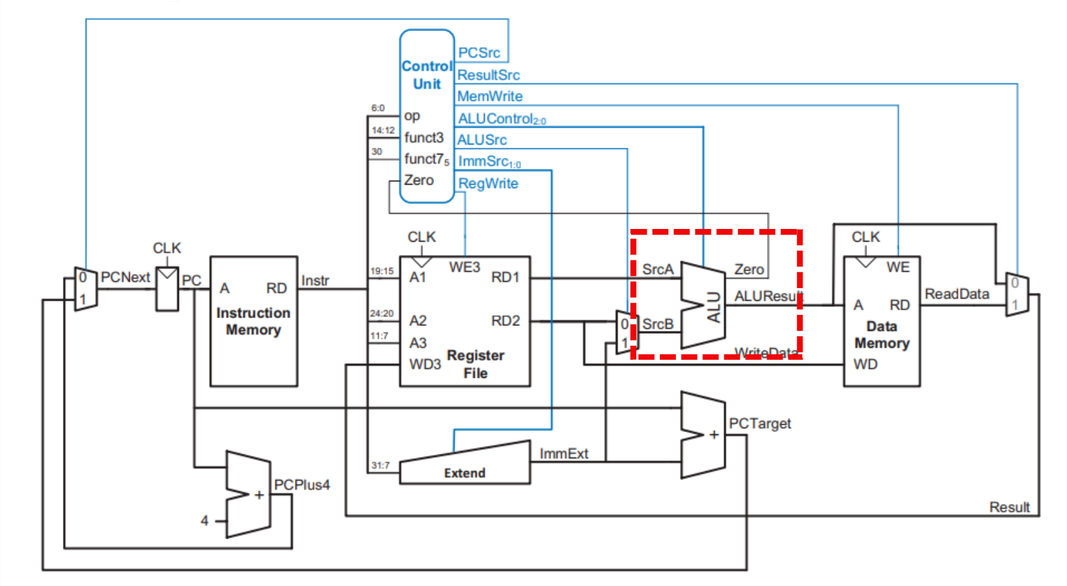


- **Data Memory**: The data memory has a single read/write port. If its write enable, WE, is asserted, then it writes data WD into address A on the rising edge of the clock. If its write enable is 0, then it reads from address A onto the read data bus, RD.

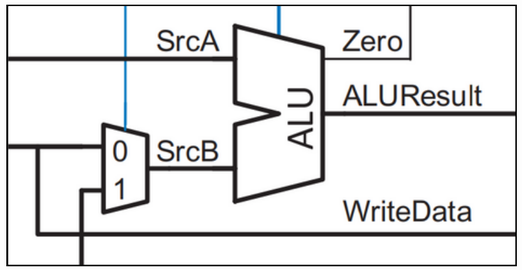


Single Cycle Processor

Internal Components

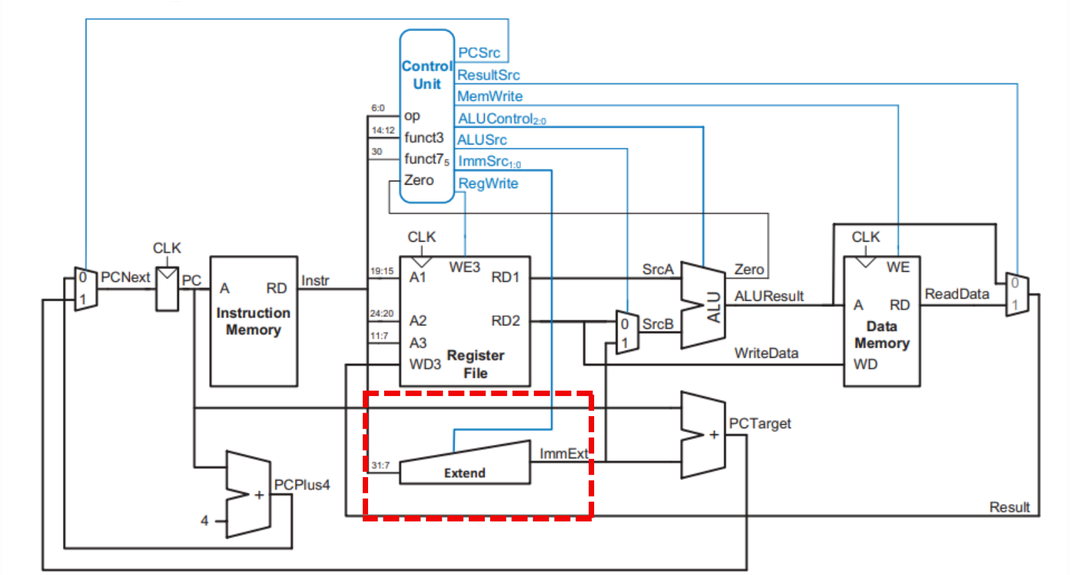


- **ALU**: ALU is responsible for the all arithmetic operations. It has two input ports SrcA and SrcB 32 bits long and output, ALUResult, is also 32 bits long. It performs operation like addition, subtraction, AND, OR, XOR, etc.

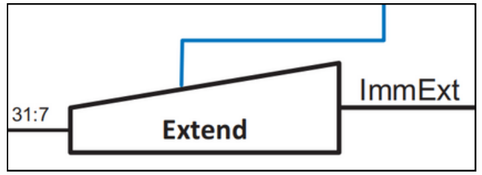


Single Cycle Processor

Internal Components



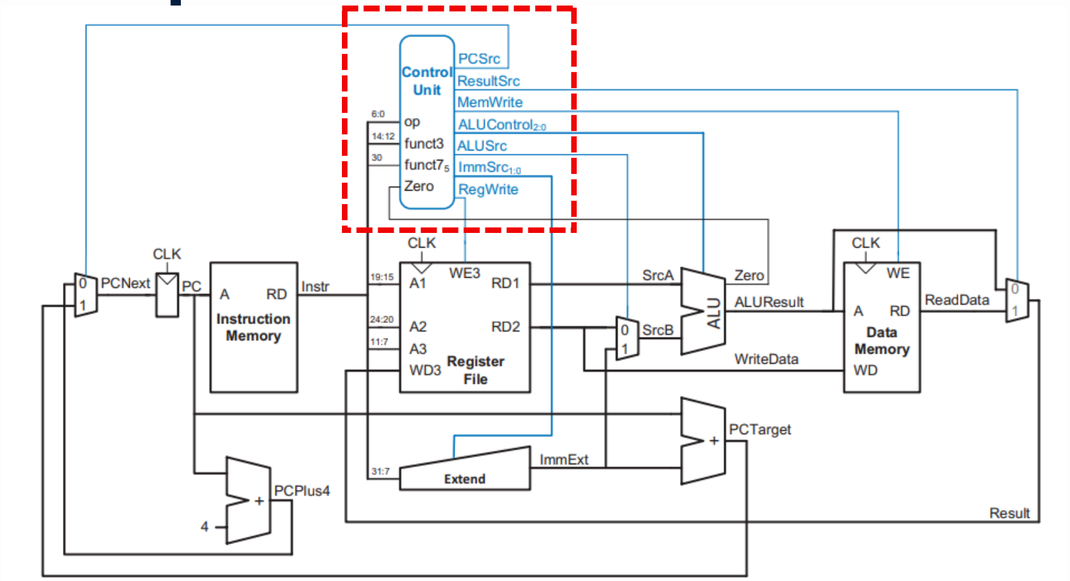
Extender Unit: Extender Unit have major use in the immediate type instruction it extend the input data by extending the Most significant bit It extend the instruction are as follows:



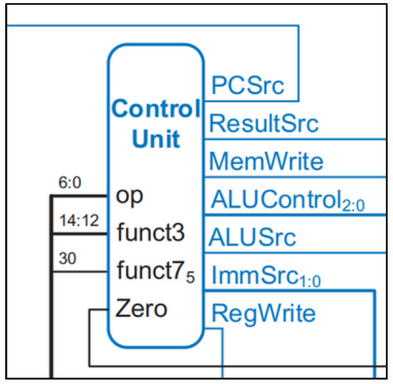
ImmSrc	ImmExt	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate
11	{{12{*Instr*[31]}}, *Instr*[19:12], *Instr*[20], *Instr*[30:21], 1'b0}	J	21-bit signed immediate

Single Cycle Processor

Internal Components

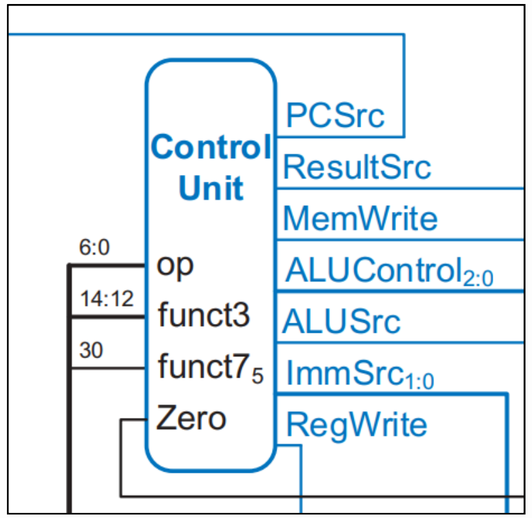


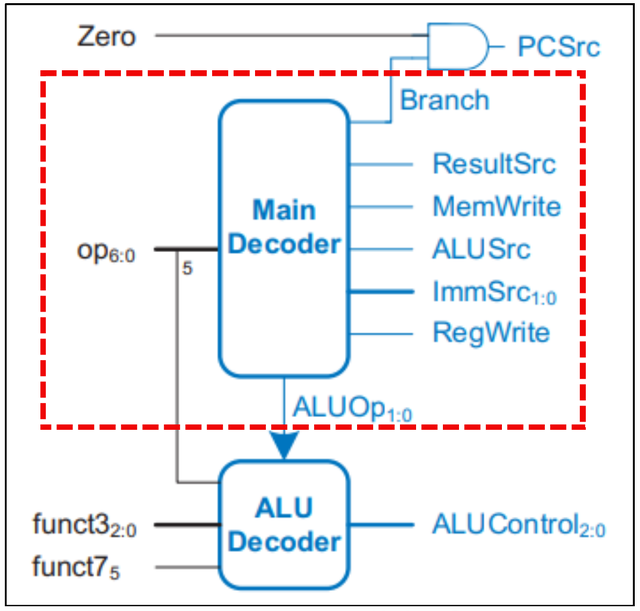
- **Control Unit**: Control unit consist of generating all the control signals within the processor, allows the components to behave according to instructions provided in the instruction memory.
- The Control Unit consists of 2 parts:
  - Main decoder
  - ALU Decoder



Single Cycle Processor

Internal Components



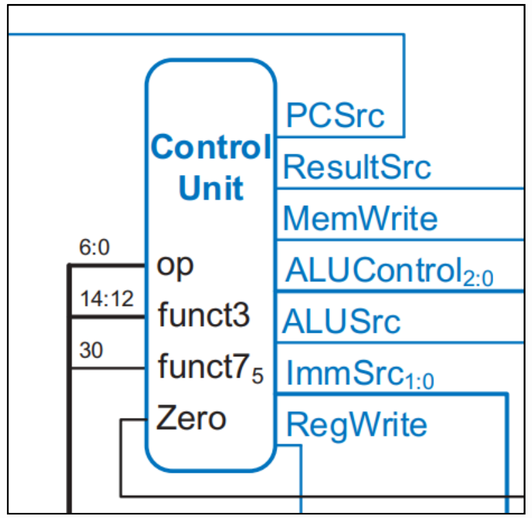


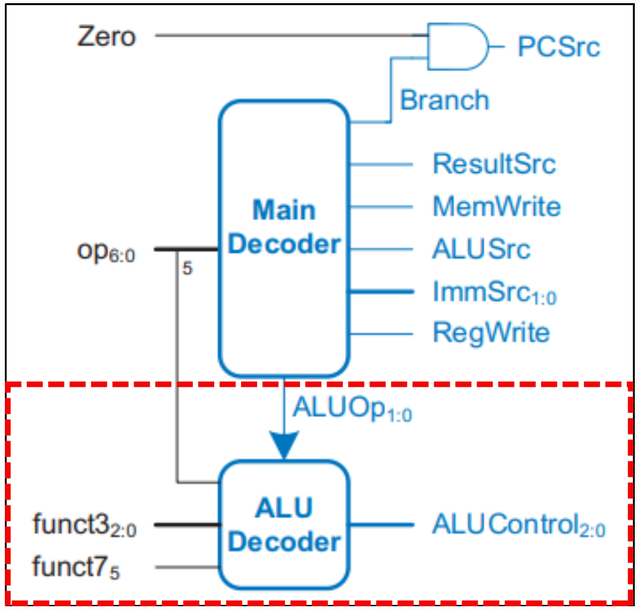
- **Main Decoder**: the main decoder in the control unit is responsible for generating control signals based on the instruction opcode. These control signals drive various components of the processor to execute the instruction correctly in one clock cycle.

Instruction	Op	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
lw	0000011	1	00	1	0	1	0	00
sw	0100011	0	01	1	1	x	0	00
R-type	0110011	1	xx	0	0	0	0	10
beq	1100011	0	10	0	0	x	1	01

Single Cycle Processor

Internal Components





- **ALU Decoder**: Generates the specific ALU control signals required for the ALU to perform the correct operation based on the instruction. The decoding table is as follows:

ALUOp	funct3	{op$_5$, funct7$_5$}	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

Single Cycle Processor

Simulation Results

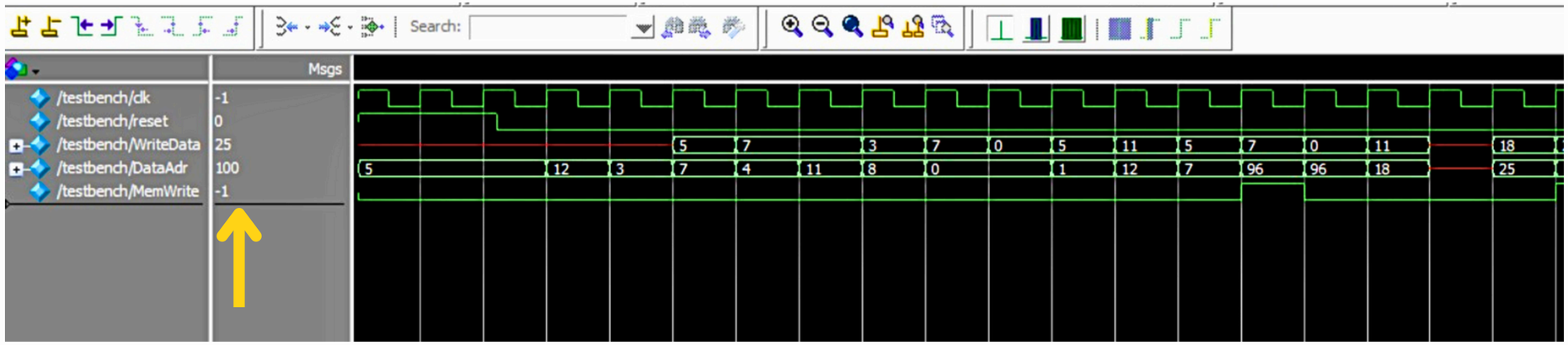
- Below is the RISC V Assembly Language Code used to simulate different commands on the processor.

```
#         RISC-V Assembly          Description                  Address   Machine Code
main:     addi x2, x0, 5           # x2 = 5                     0         00500113
          addi x3, x0, 12          # x3 = 12                    4         00C00193
          addi x7, x3, -9          # x7 = (12 - 9) = 3          8         FF718393
          or   x4, x7, x2          # x4 = (3 OR 5) = 7          C         0023E233
          and  x5, x3, x4          # x5 = (12 AND 7) = 4        10        0041F2B3
          add  x5, x5, x4          # x5 = 4 + 7 = 11            14        004282B3
          beq  x5, x7, end         # shouldn't be taken         18        02728863
          slt  x4, x3, x4          # x4 = (12 < 7) = 0          1C        0041A233
          beq  x4, x0, around      # should be taken            20        00020463
          addi x5, x0, 0           # shouldn't execute          24        00000293
around:   slt  x4, x7, x2          # x4 = (3 < 5) = 1           28        0023A233
          add  x7, x4, x5          # x7 = (1 + 11) = 12         2C        005203B3
          sub  x7, x7, x2          # x7 = (12 - 5) = 7          30        402383B3
          sw   x7, 84(x3)          # [96] = 7                   34        0471AA23
          lw   x2, 96(x0)          # x2 = [96] = 7              38        06002103
          add  x9, x2, x5          # x9 = (7 + 11) = 18         3C        005104B3
          jal  x3, end             # jump to end, x3 = 0x44     40        008001EF
          addi x2, x0, 1           # shouldn't execute          44        00100113
end:      add  x2, x2, x9          # x2 = (7 + 18) = 25         48        00910133
          sw   x2, 0x20(x3)        # [100] = 25                 4C        0221A023
done:     beq  x2, x2, done        # infinite loop              50        00210063
```

Single Cycle Processor

Simulation Results

- The simulation works perfectly as we wanted to write the data [25] at location [100] in memory. According to waveform, the port *WriteData* shows the data to be written and port *DataAdr* shows the location for the data to be written.
- Observing at the address [100], we can see the data [25] is written in the memory.



Pipelined Processor

What is Pipelining

- Pipelining is a technique used in modern processors to improve performance by executing **multiple instructions simultaneously**.
- It breaks down the execution of instructions into several **stages**, where each stage completes a part of the instruction.
- These stages can overlap, allowing the processor to work on different instructions at various stages of completion.

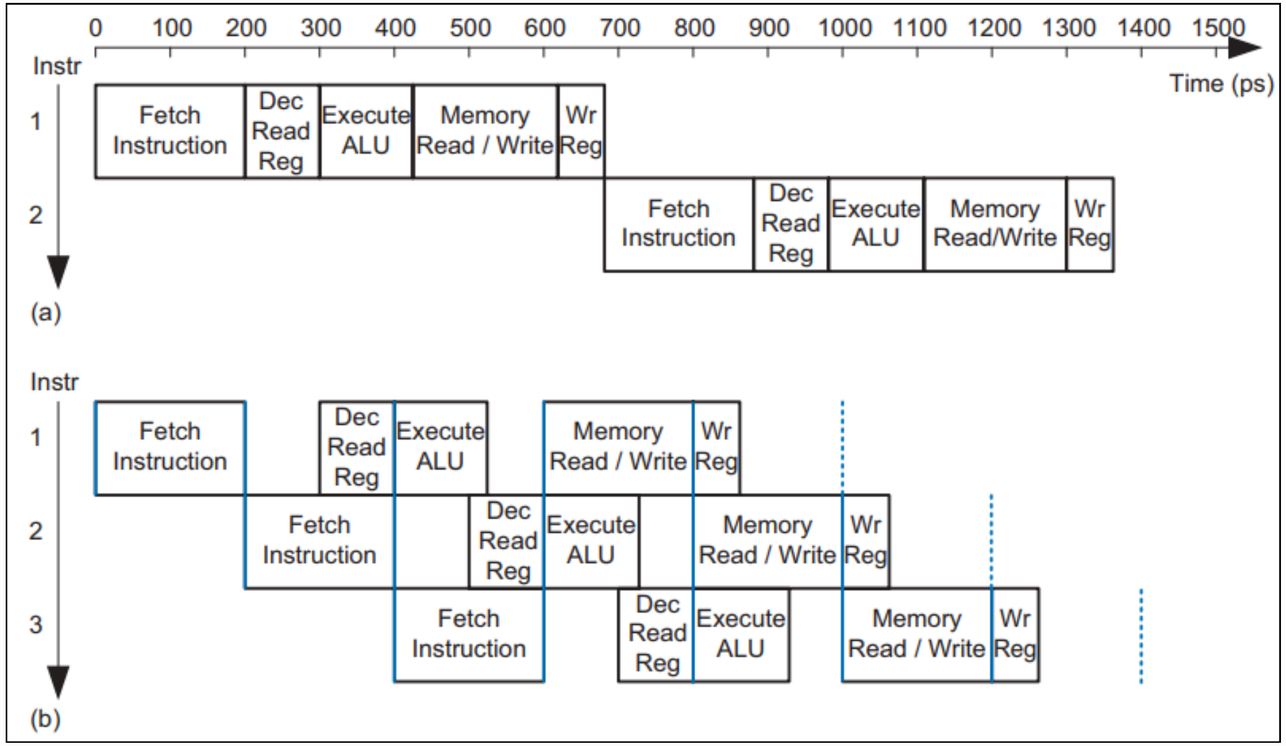
Pipelining the RISC V Single Cycle Processor

- To pipeline, we divide the processor into **5 stages**:
  - Instruction Fetch
  - Instruction Decode
  - Instruction Execute
  - Memory Access
  - Write Back

Pipelined Processor

Single Cycle v/s Pipelined RISC V Processor Instructions Execution Timing Diagram

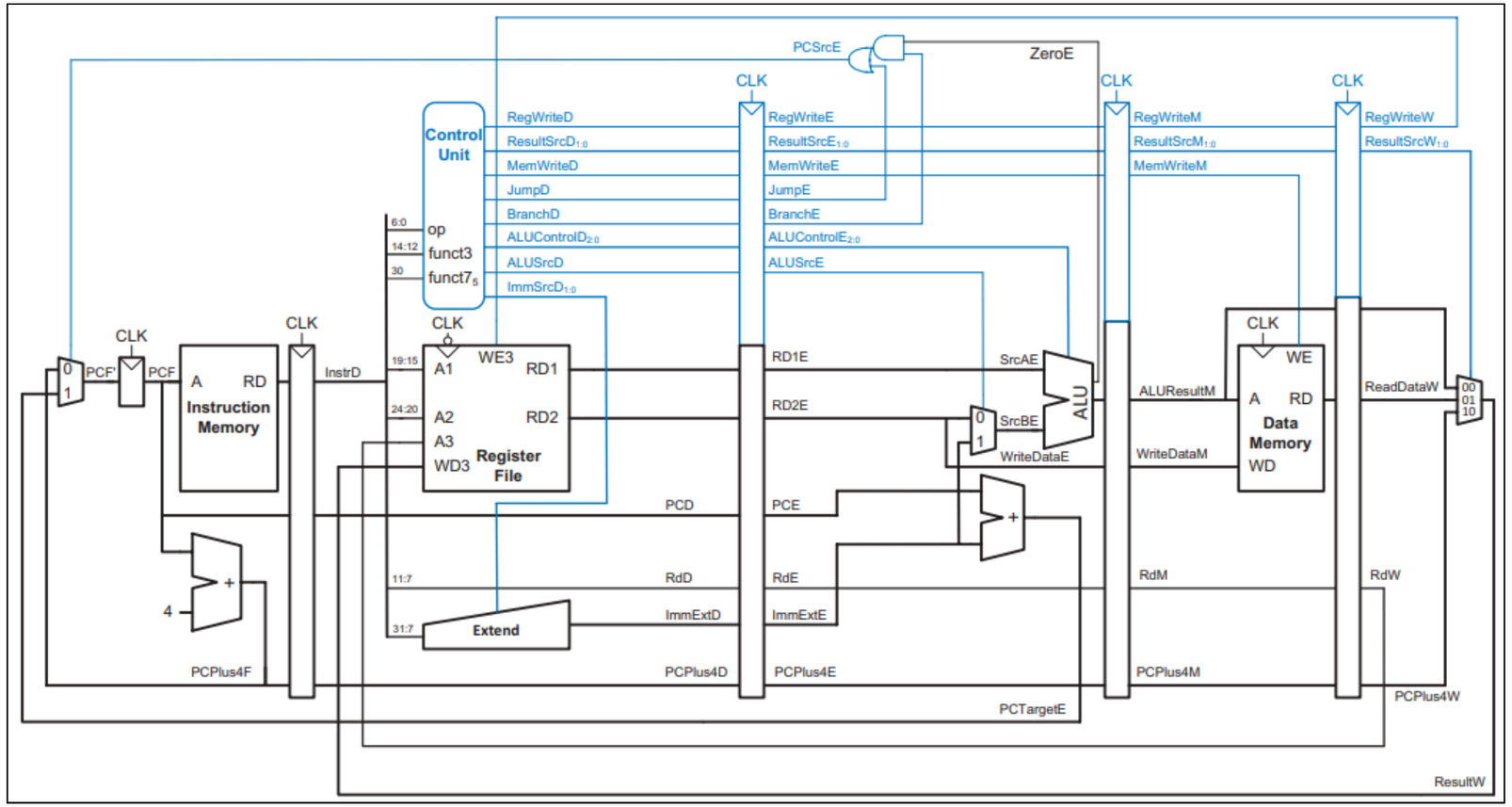
From the figures, we can clearly see that the pipelined system has executed 3 instructions **[figure b]** in the same time as the normal system executed 2 instructions **[figure a]**.



Pipelined Processor

Pipelined Processor Datapath

- Same control unit as single-cycle processor
- Control signals travel with the instruction (drop off when used)



Pipelined Processor

Pipeline Hazards

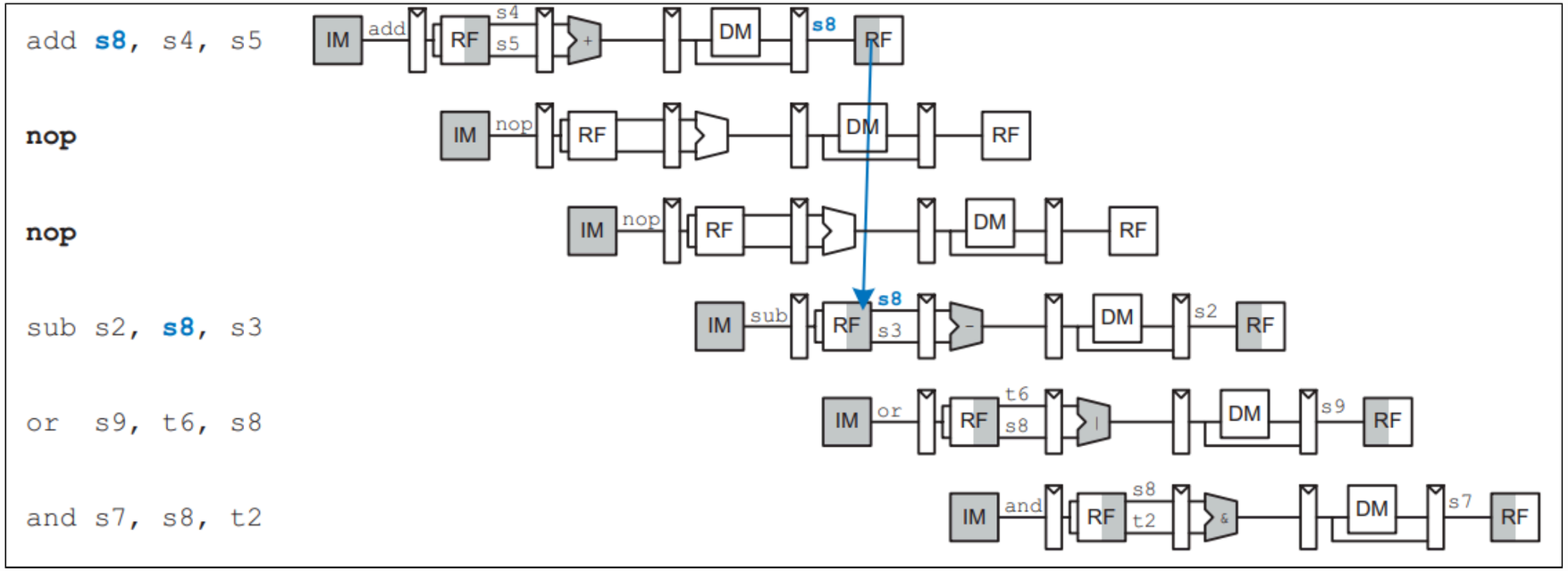
- **Hazards** are situations that prevent the next instruction in the pipeline from executing in its designated clock cycle.

Hazard Name	Cause	Example	Resolution
Structural Hazard	**Happens when two instructions need to use the same part of the processor at the same time, but there is only one of that part.**	*lw x1, 0(x2)* *lw x3, 4(x4)* ***(Both instructions want to use memory at the same time)***	**Use separate memories or stall (pause) one instruction and let the other finish first.**
Data Hazard	**An instruction needs a result that hasn't been calculated yet.**	*add x1, x2, x3* *sub x4, x1, x5* **(Needs x1 before it's ready)**	**Use forwarding to pass results early or stall the pipeline.**
Control Hazard	**The processor doesn't know which instruction to fetch next because of a branch.**	*beq x1, x2, label* **(Is the branch taken or not?)**	**Use branch prediction, delay slots, or flush wrong instructions.**

Pipelined Processor

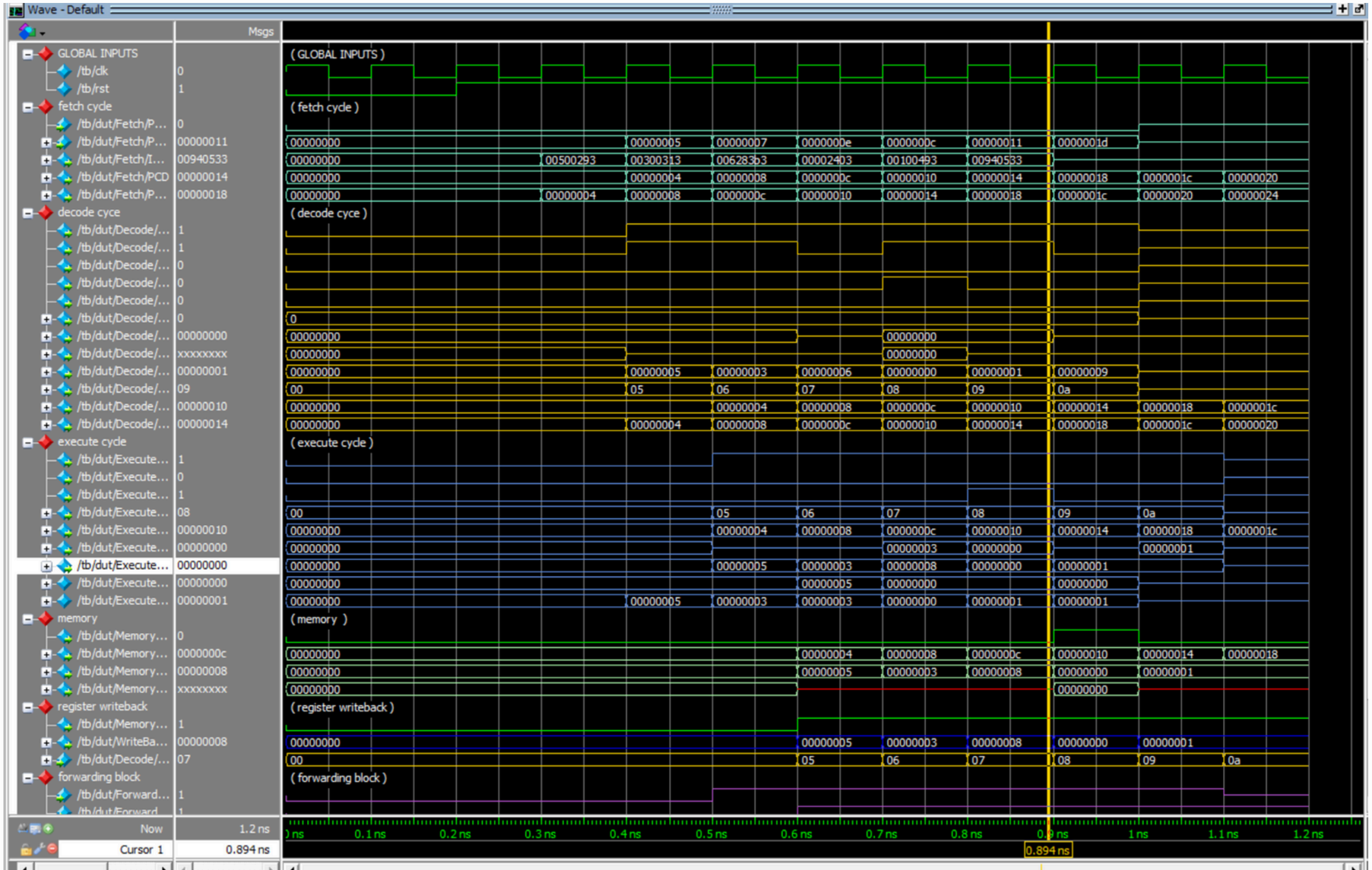
Compile Time Hazards Elimination

- Insert enough nops (no operations) for result to be ready.
- Or move independent useful instructions forward.



Pipelined Processor

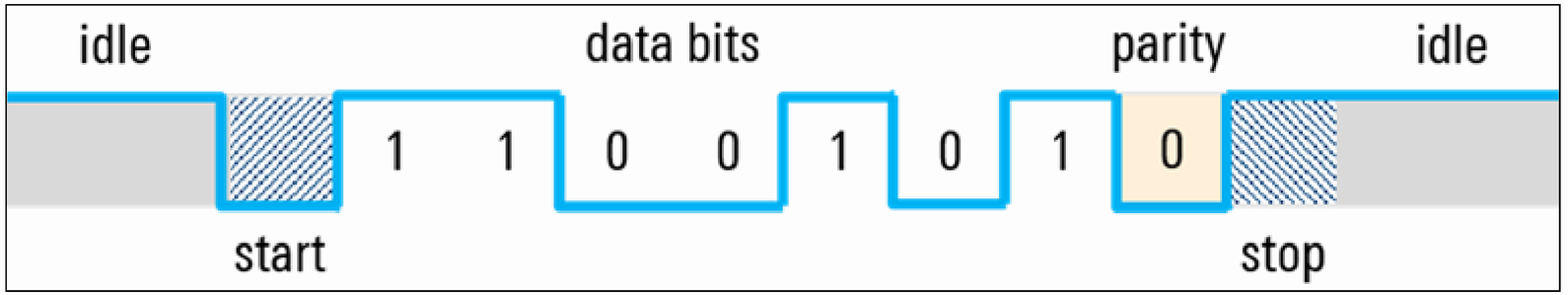
Simulation



Universal Asynchronous Receiver/Transmitter(UART)

What is UART

- It’s a **asynchronous** communication protocol used for data transmission with configurable speeds.
- It’s Asynchronous meaning it doesn’t need a ***clk*** signal to synchronize the transmitter and receiver.
- Shifts parallel data into serial data which can be transmitted and received and converted back to parallel data.
- Uses a special ***data frame format*** to transmit and receive data.



Universal Asynchronous Receiver/Transmitter(UART)

Components of UART

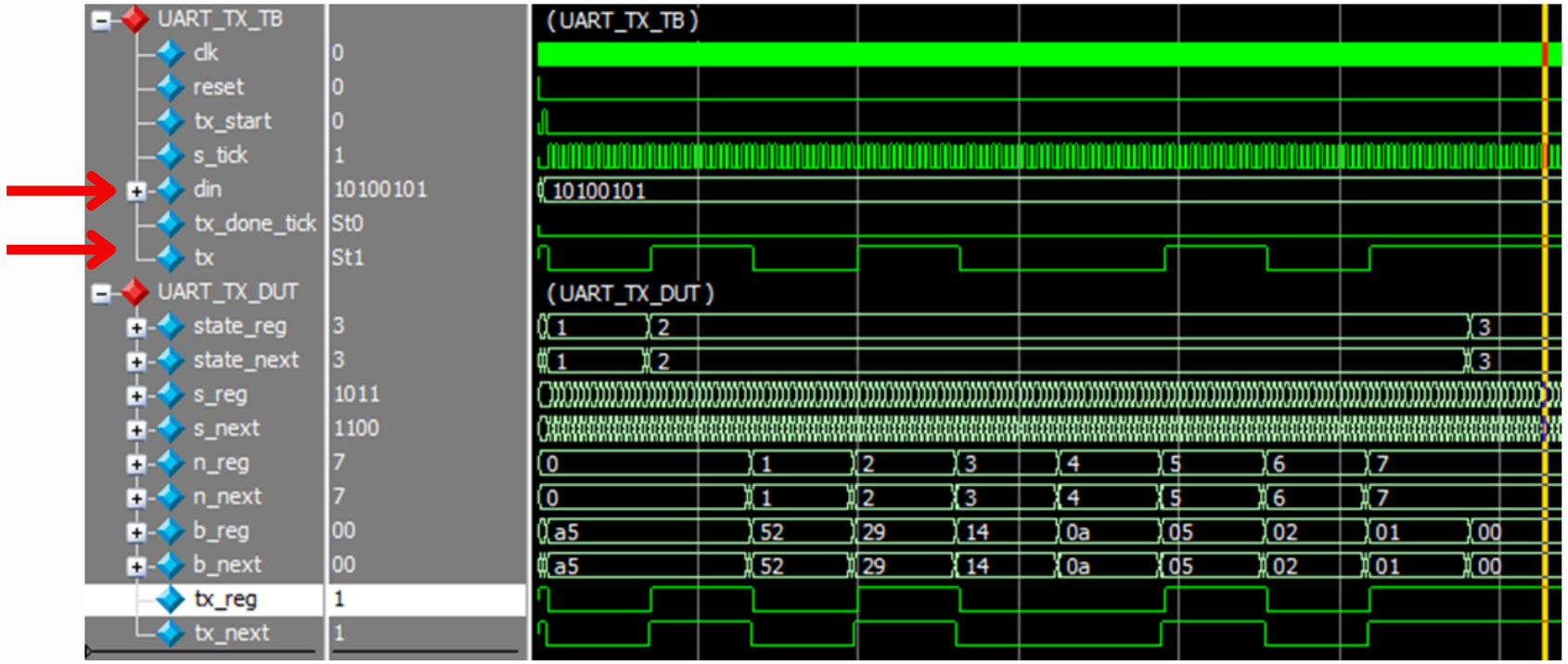
- **Transmit Shift Register:** Holds the data to be transmitted.
- **Receive Shift Register:** Holds the data to be received.
- **Tx and Rx Buffers:** Store the data bits that are currently being transmitted or received and are not valid for acceptation. Once they are valid, they are passed to their respective registers.
- **Baud Rate Generator:** Generates TICK signals as a reference to transmit and receive bits.
- **Parity Generator/Checker (Optional):** Used to generate and verify parity bits to check errors in the data.

Universal Asynchronous Receiver/Transmitter(UART)

Simulation Results

Transmitter Simulation

The data to be transmitted is present at port ***din*** and observing the output port ***tx***, we can observe that the data successfully transmitted on the channel.

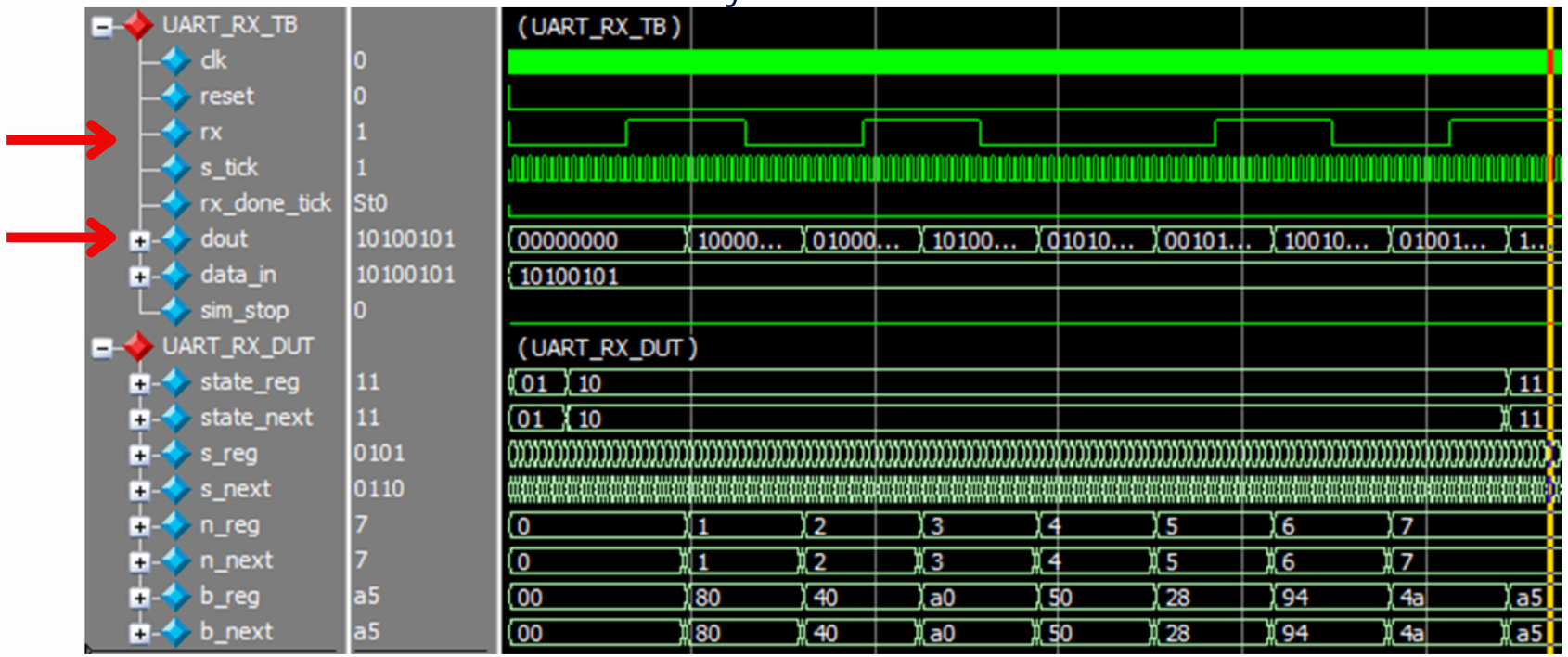


Universal Asynchronous Receiver/Transmitter(UART)

Simulation Results

Receiver Simulation

The data to be received is present at port ***rx*** and observing the output port ***dout***, we can observe that the data successfully received from the channel to the buffer.



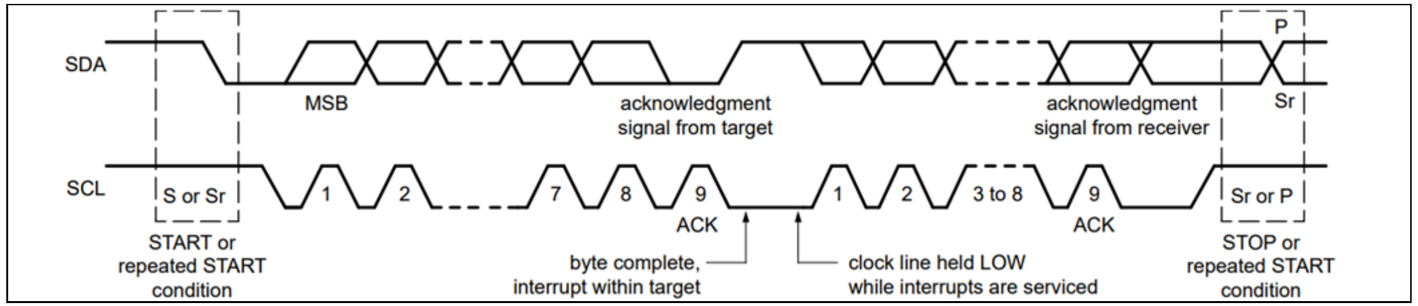
Inter Integrated Circuit (I2C)

What is I2C?

- It’s a **synchronous communication protocol** used for data transmission between multiple devices on the same bus.
- It’s synchronous, meaning it needs a clock (SCL) signal to synchronize the transmitter and receiver.
- Uses a specific frame format with **start condition**, **address**, **read/write bit**, **ACK/NACK**, and **stop condition** to manage data transfer.
- Supports communication between **multiple masters** and **multiple slaves**.
- Requires only two wires: **SDA** (Serial Data) and **SCL** (Serial Clock).
- Supports different speeds: **Standard Mode (100 kbps)**, **Fast Mode (400 kbps)**, and more.

Inter Integrated Circuit (I2C)

I2C Frame Format



- **SDA (Serial Data Line):**
  - Transmits data bits (addresses, commands, or sensor readings) between the master and slave devices.
  - **Working:** Data on SDA must remain stable when SCL is high; it changes only when SCL is low. Both master and slave can use this line to send or receive data depending on the communication phase.
- **SCL (Serial Clock Line):**
  - Sends the clock signal from the master to synchronize when each bit of data on SDA should be read or written.
  - **Working:** The master controls the clock; data on SDA is sampled on SCL's rising edge and should be set up when SCL is low.

Inter Integrated Circuit (I2C)

Components Of I2C

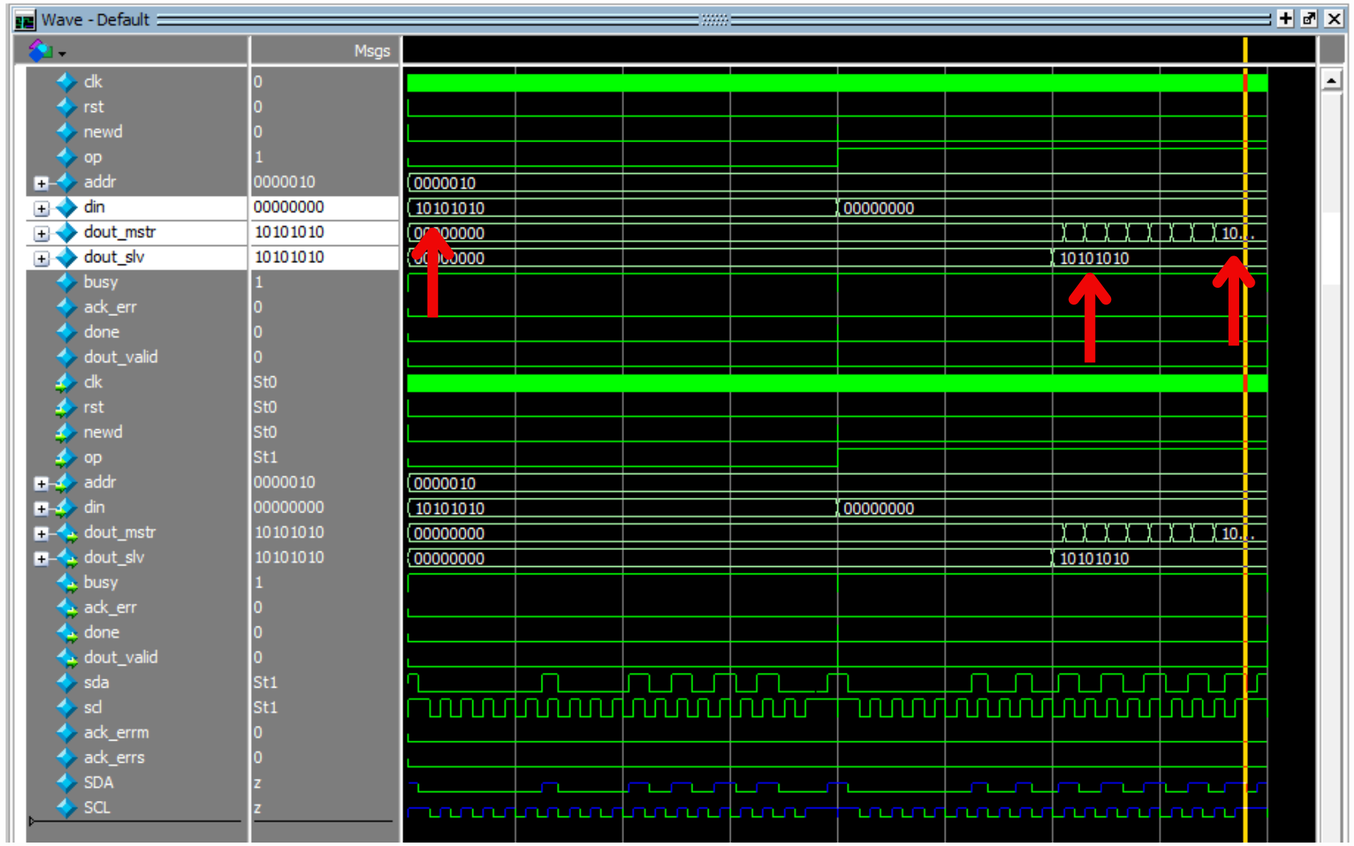
- **Transmit Shift Register:** Holds the parallel data to be transmitted. Converts it into serial form and shifts it out on the SDA line.
- **Receive Shift Register:** Collects incoming serial data from the SDA line and converts it into parallel form for output.
- **Tx and Rx Buffers:** Temporary storage areas that hold data before it enters the shift registers for transmission or after it’s received before processing.
- **Address Register:** Holds the slave address to identify the slave during communication. The master sends this address at the start of each transaction.
- **Acknowledge (ACK/NACK) Logic:** Handles the acknowledgment (ACK) signals after each byte to confirm that the slave has successfully received the data.
- **Finite State Machine (FSM):** Controls the sequence of operations during communication. It manages when to send start/stop conditions, send/receive data, and handle acknowledgments, ensuring proper protocol flow.

Inter Integrated Circuit (I2C)

Simulation Results

Top Module Simulation

The data to be transmitted is present at port ***din***, then the data is passed to slave seen at port ***dout_slv*** and then passed back to master as seen on port ***dout_mstr***.



Inter Integrated Circuit (I2C)

Simulation Results

Top Module Simulation

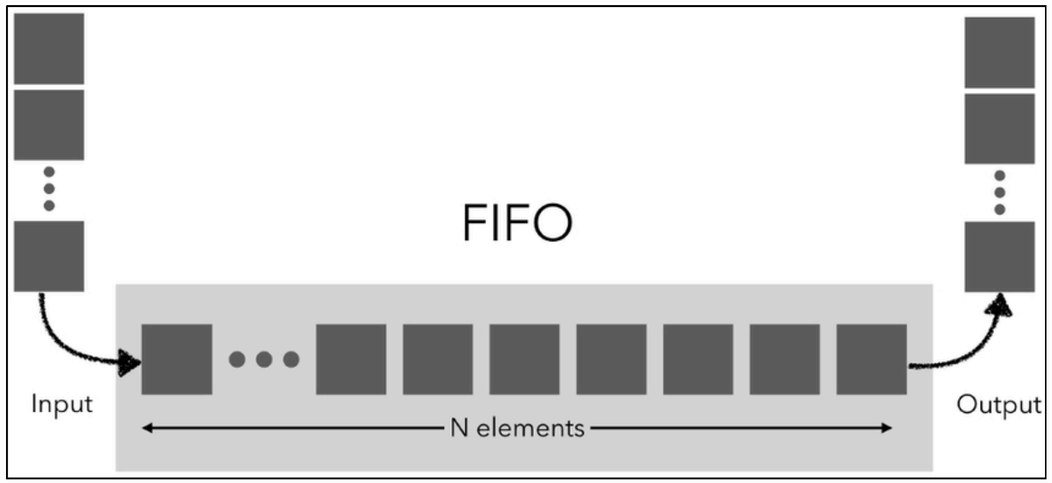
Transcript View of the I2C Top Module Simulation.

```
#    Time: 0 ps  Iteration: 0  Instance: /i2c_top_tb File: i2c_top_tb.sv
# [TB] Reset complete
# [TB] ---Write: Addr=10, Data=10101010---
# [TB] ---Read: Addr=10, Data=10101010---
# [TB] ---Verification PASSED: Addr=10, Data=10101010---
# [TB] ##################################################
# ** Note: $stop    : i2c_top_tb.sv(106)
#    Time: 160135 ns  Iteration: 2  Instance: /i2c_top_tb
# Break in Module i2c_top_tb at i2c_top_tb.sv line 106
# 0 ps
# 168141750 ps
```

First In First Out (FIFO)

What is FIFO

- A **FIFO** is a **queue-based memory structure** that stores data in the order it arrives and retrieves it in the same order.

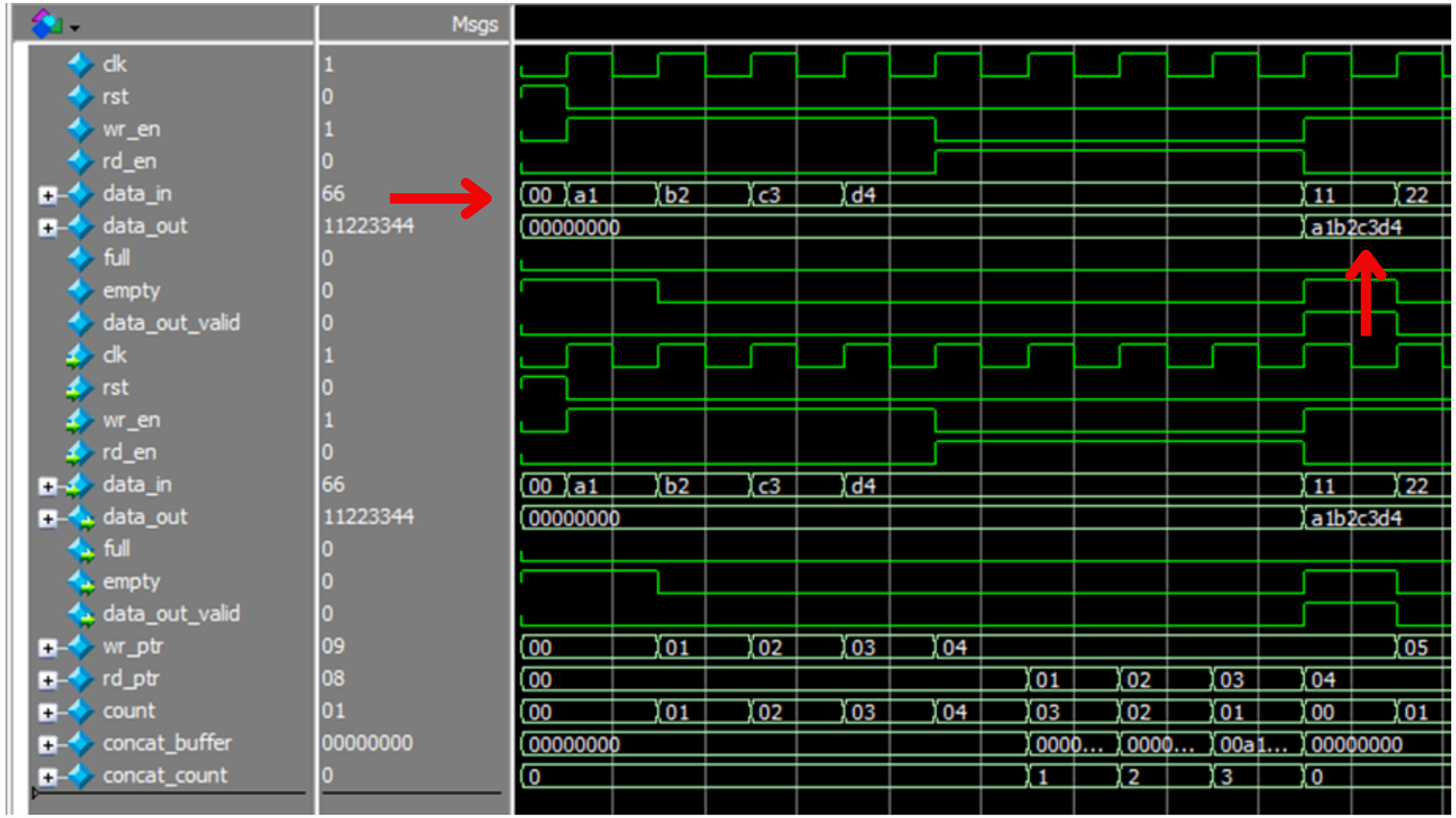


- At the input side, this FIFO has to take 4 inputs sequentially and have to concatenate the **4 words into one 32-bit chunks** to pass as a complete 32-bit instruction to the processor.
- Then at the output, this FIFO should break the **32-bit chunk into 4 words** and pass them to output.

First In First Out (FIFO)

Simulation Results

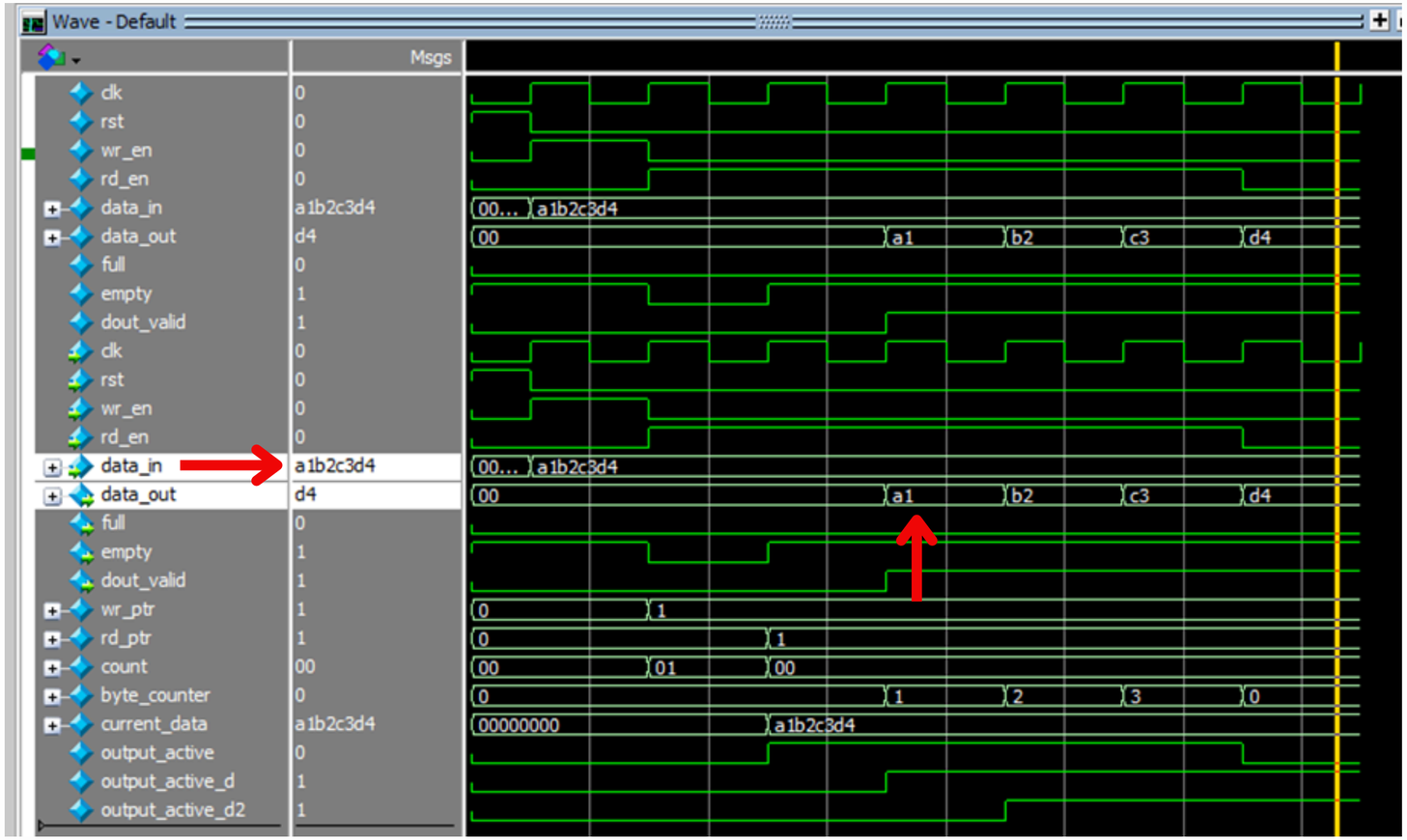
This FIFO takes 4 data points seen on port ***data_in*** and then concatenates them and sends them to it’s output as seen on port ***data_out***.



First In First Out (FIFO)

Simulation Results

This FIFO takes 32-bit chunk seen on port ***data_in*** and then splits into 4 words and sends them to it’s output one by one as seen on port ***data_out***.



First In First Out (FIFO)

Simulation Results

Transcript view of FIFO Simulation

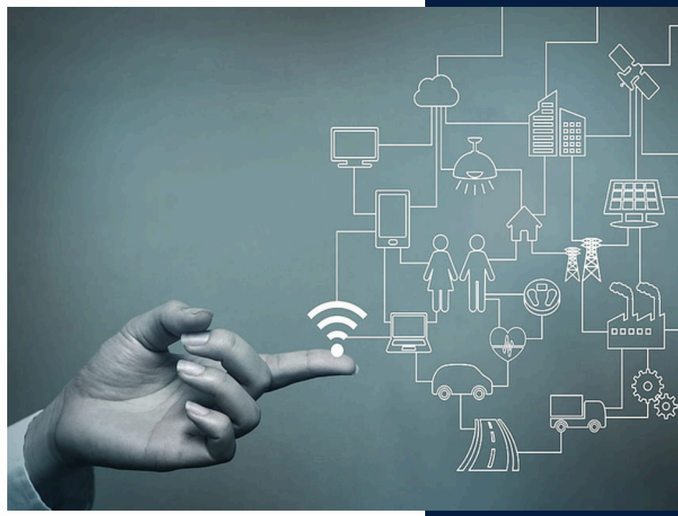
```
# Loading work.fifo_uart_tb
# Loading work.fifo
# [TBs] ---Test PASSED: data_out = a1b2c3d4, expected = a1b2c3d4
# [TBs] ---Test PASSED: data_out = 11223344, expected = 11223344
# [TBs] ---Test PASSED: data_out = 55667788, expected = 55667788
# ** Note: $stop    : fifo_uart_tb.sv(95)
#    Time: 255 ps  Iteration: 1  Instance: /fifo_uart_tb
```

APPLICATIONS

Embedded Systems in Industrial Automation

- **Purpose:** Enhances industrial machinery by processing real-time data from sensors to improve performance, ensure operational efficiency, and maintain safety.
- **Sensor Data Acquisition:**
  - UART interfaces receive sensor data (e.g., temperature, pressure) from external sources.
  - Chosen for its simplicity and reliable long-distance communication.
- **Data Processing:**
  - RISC-V processor analyzes incoming data to trigger appropriate safety or control responses.
  - Example: Activates safety mechanisms if critical parameters exceed set limits.
- **Output Control:**
  - I2C bus transmits processed data to actuators, displays, or motor controllers.
  - Suitable for managing multiple devices efficiently within the system.
- **System Benefits:**
  - Modular architecture supports efficient monitoring, real-time response, and robust safety measures, enhancing overall industrial productivity and reliability.

APPLICATIONS



IoT (Internet of Things) Devices:

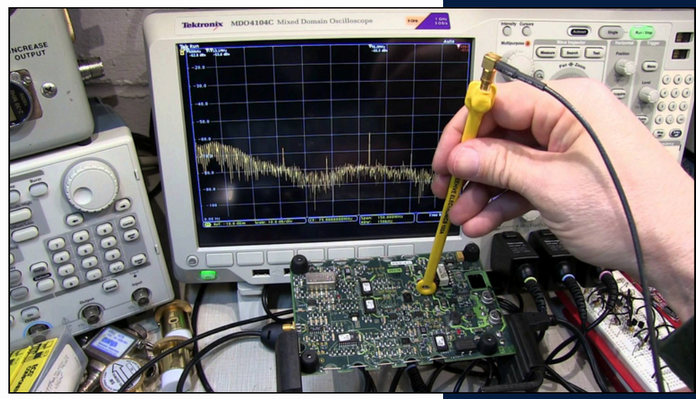
- **Purpose:** Designed to be compact, energy-efficient, and capable of wireless data exchange with other devices or cloud platforms. Can be used in smart homes, environmental monitoring, fitness trackers, and more.
- **Sensor Data Acquisition:**
  - UART interfaces receive sensor data (e.g., temperature, humidity, air quality) from external sources.
  - Provides reliable, simple, and low-power serial communication.
- **Data Processing:**
  - RISC-V processor performs data analysis, anomaly detection, and generates alerts or actions based on sensor input.
- **Output Control:**
  - I2C bus transmits processed data to displays or cloud gateways.
  - Supports efficient multi-device communication essential for IoT ecosystems.
- **System Benefits:**
  - Combines efficient sensor interfacing, real-time processing, and low-power operation, making it ideal for embedded IoT solutions requiring reliable connectivity and smart data management.

APPLICATIONS

Medical Devices:

- **Purpose:** Used in devices like ECG machines, glucose meters, and pulse oximeters to perform real-time health monitoring with high accuracy and speed.
- **Sensor Data Acquisition:**
  - UART interfaces collect serial data from medical sensors (e.g., ECG electrodes, glucose sensors).
  - Ensures reliable, low-latency communication for critical health readings.
- **Data Processing:**
  - RISC-V processor analyzes sensor data by performing tasks such as ECG filtering, heart rate calculation, or glucose level estimation.
- **Output Control:**
  - I2C bus sends processed data to displays or external medical systems for real-time visualization or remote monitoring.
- **System Benefits:**
  - Provides fast, accurate, and reliable processing of critical health data, ensuring safety, efficiency, and error-free operation in medical environments.

APPLICATIONS



Automated Test Equipment (ATE):

- **Purpose:** Used across industries to test and verify the functionality of components and systems through precise data collection, analysis, and result reporting.
- **Sensor Data Acquisition:**
  - UART interfaces collect raw data or test results from Devices Under Test (DUTs).
  - Provides a simple, reliable communication link for transmitting measurement data.
- **Data Processing:**
  - RISC-V processor processes the test data, including measurements of voltage, current, and resistance, ensuring accurate validation of DUTs under various conditions.
- **Output Control:**
  - I2C bus transmits the processed results to displays, recorders, or external storage for further analysis or documentation.
  - Suitable for handling multiple output devices within ATE setups.
- **System Benefits:**
  - Enables fast, accurate, and reliable testing of components and systems, ensuring high efficiency and data integrity in industrial test environments.

APPLICATIONS

Wearable Devices:

- **Purpose:** Used in fitness trackers, smartwatches, and health monitors to collect, process, and display personal health and fitness data in real time.
- **Sensor Data Acquisition:**
  - UART interfaces gather input from various sensors (e.g., accelerometers, heart rate monitors).
  - Facilitates reliable, low-power communication with sensor modules.
- **Data Processing:**
  - RISC-V processor performs real-time analysis of sensor data, such as step counting or heart rate monitoring.
- **Output Control:**
  - I2C bus transmits processed data to compact displays (e.g., OLED or LCD) or shares it with other connected devices.
  - Efficiently supports multiple peripherals in small form-factor designs.
- **System Benefits:**
  - Provides compact, energy-efficient, and real-time performance, making it ideal for wearable applications requiring instant feedback and low power consumption.

Conclusion

In conclusion, this project focuses on developing a System-on-a-Chip (SoC) that integrates a RISC-V processor as the central CPU, enabling communication with peripheral devices through UART and I2C interfaces. This design combines computational capabilities with efficient peripheral communication within a single chip, showcasing a complete and compact system integration.

THANK YOU!