RISC-V Processor Design using LLM

“To enable users to design processors through natural language interactions, translating user input into hardware description code like verilog”

Primary Objectives :

- To shorten the time window in processor design by leveraging LLM
- Moving towards an autonomous process with no human in the loop (HITL)

Guide: Dr.Jaynarayan T Tudu

Shanmukh (CS21B018) & Pranay (CS21B024)

Problem Statement

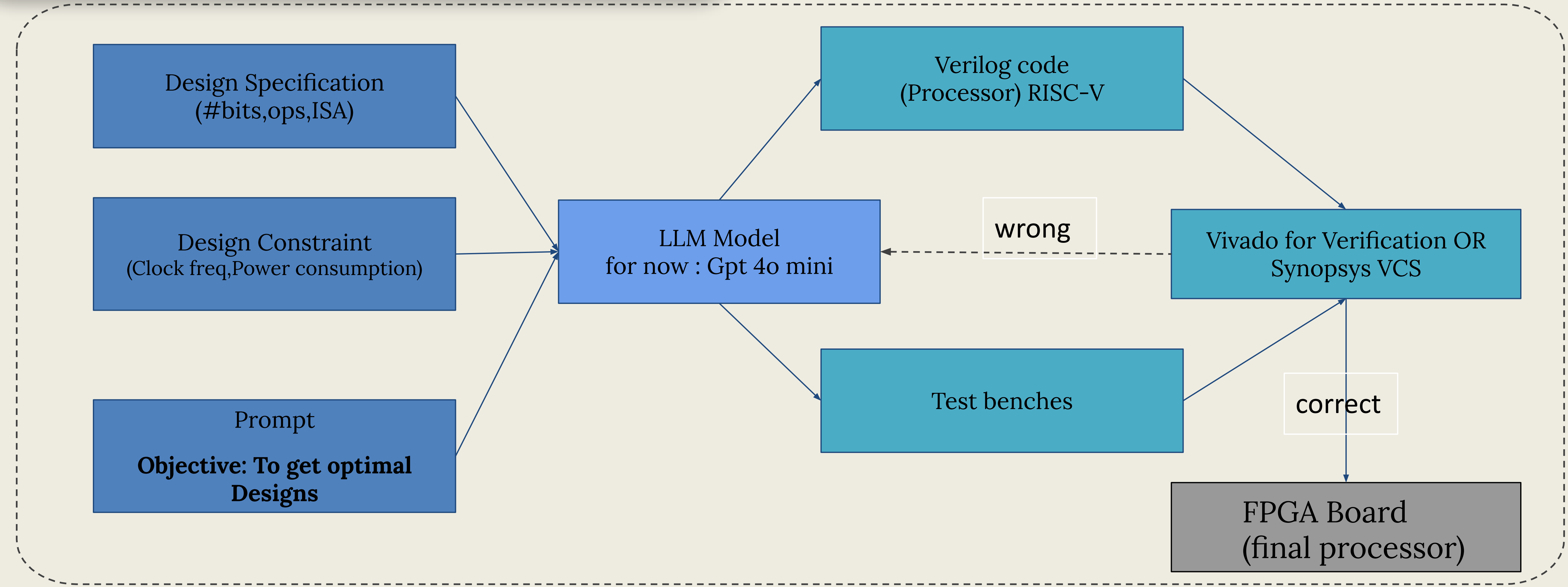
Statement:

- **RISC-V Processor using LLM** : To design a RISC processor completely by utilizing LLM with none or minimal human intervention through optimal prompting.

Challenges Addressing :

- **Simplifying Processor Design** : Reduce the need for expert-level knowledge in HDLs like Verilog.
- **Accessibility** : Allow non-experts to design processors using basic processor knowledge and English.
- **Efficiency** : Speed up the hardware development process through LLM-assisted design.
- **LLM** : Develop a model that is fine-tuned specifically for processor design in verilog.

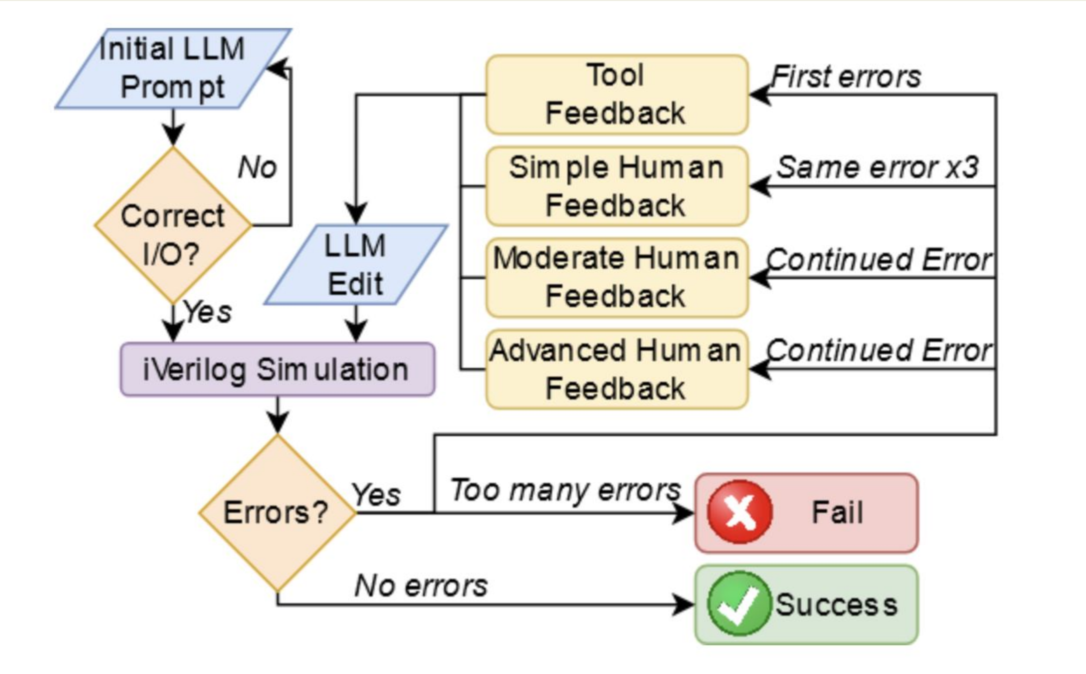
Project Overview



https://chatgpt.com/share/6706ffc3-1bc0-8013-88dd-2c514b82983a
https://chatgpt.com/share/6706fffb-7f1c-8013-94a0-bb6dafbf0d4f
https://chatgpt.com/share/67070036-4488-8013-a0d0-30af87eb9262

Work Done

- The LLM would be given a specification and generate both the design and tests. A human would then run the tests in VIvado and return any errors to the LLM for corrections.
- Recognizing LLMs' tendency to get stuck in feedback loops, we planned for **occasional human intervention** as needed.
- We have generated both the **design and its test bench** from GPT itself to keep the human intervention as minimal as possible.
- We have Designed and tested all the components such as Alu, Controlunit, Data mem, Instruction mem, Register ,and processor



Through early experimentation, we refined this initial process and decided on this

1.DAVE: Deriving Automatically Verilog from English | Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD
2.[2305.14019] ChipGPT: How far are we from natural language hardware design
3.Rome was Not Built in a Single Step: Hierarchical Prompting for LLM-based Chip Design | Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD
4.https://doi.org/10.48550/arXiv.2406.09233

The input format in which the prompt is feeded to the LLM:

Listing 3.1: Design Prompt for a Shift Register

```
I am trying to create a Verilog model for a shift register.
It must meet the following specifications:
- Inputs:
   - Clock
   - Active-low reset
   - Data (1 bit)
   - Shift enable
- Outputs:
   - Data (8 bits)
Give a design that meets these specifications.
```

The same prompt is given to both the LLMs GPT-(4o)mini & Gemini to compare the generated output designs

LLM COMPARISON

GEMINI

Verilog

```
module shift_register (
    input clk,
    input rst,
    input [7:0] data,
    input shift_enable,
    output reg [7:0] data_out
);

always @(posedge clk or posedge rst) begin
    if (rst) begin
        data_out <= 8'b0;
    end else begin
        if (shift_enable) begin
            data_out <= {data_out[6:0], data};
        end
    end
end

endmodule
```

Use code with caution.

Chat Gpt -4(o) mini

verilog

Copy code

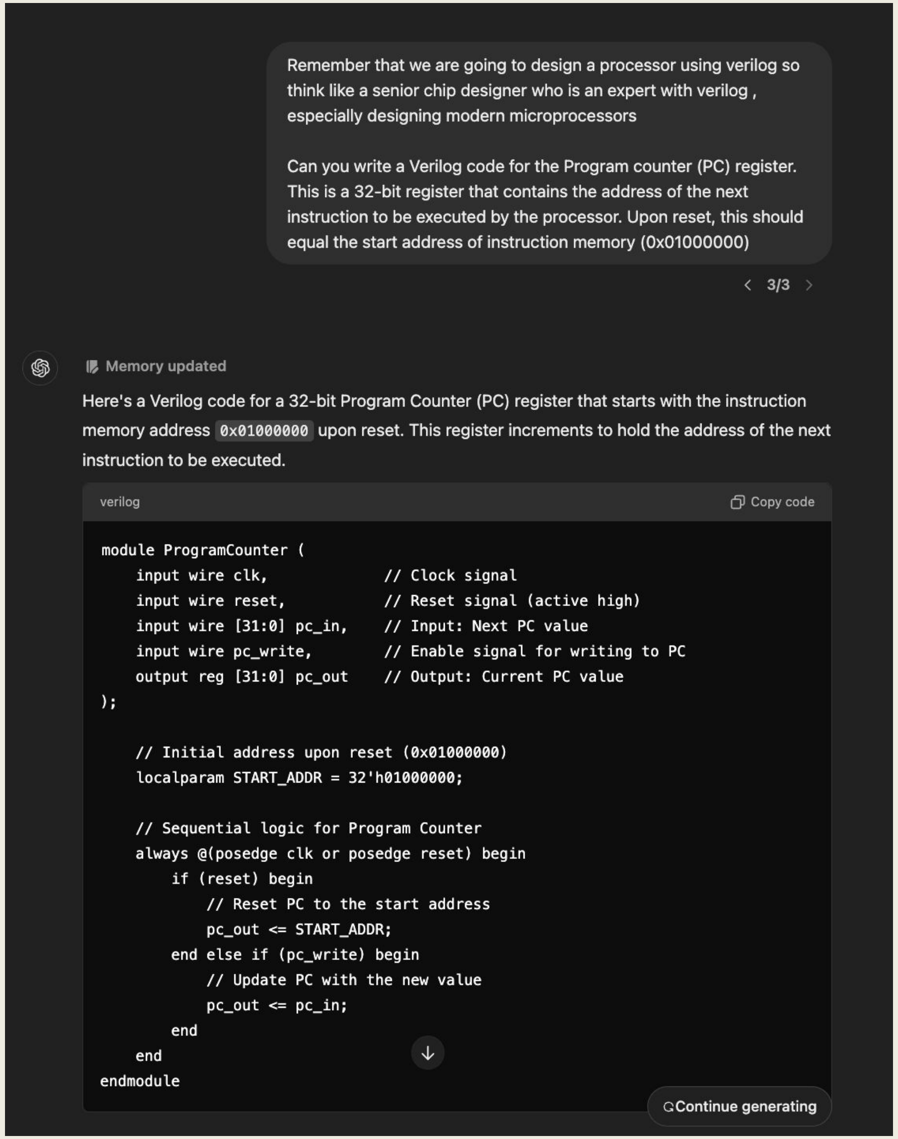
```
module shift_register (
  input clk,
  input rst,
  input [7:0] data,
  input shift_enable,
  output reg [7:0] data_out
);

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      data_out <= 8'b0;
    end else begin
      if (shift_enable) begin
        data_out <= {data_out[6:0], data[7]};  // Shift and append only the MSB of
      end
    end
  end
endmodule
```

We observed that only ChatGPT-4 could reliably produce correct designs compared to Gemini. For instance to the above prompt these are the given results.

- The design generated by Gemini contained errors. Specifically, there was an issue in the definition of the data input as you can observe from the images

PC Simulation Results



Remember that we are going to design a processor using verilog so think like a senior chip designer who is an expert with verilog , especially designing modern microprocessors

Can you write a Verilog code for the Program counter (PC) register. This is a 32-bit register that contains the address of the next instruction to be executed by the processor. Upon reset, this should equal the start address of instruction memory (0x01000000)

3/3

Memory updated

Here's a Verilog code for a 32-bit Program Counter (PC) register that starts with the instruction memory address `0x01000000` upon reset. This register increments to hold the address of the next instruction to be executed.

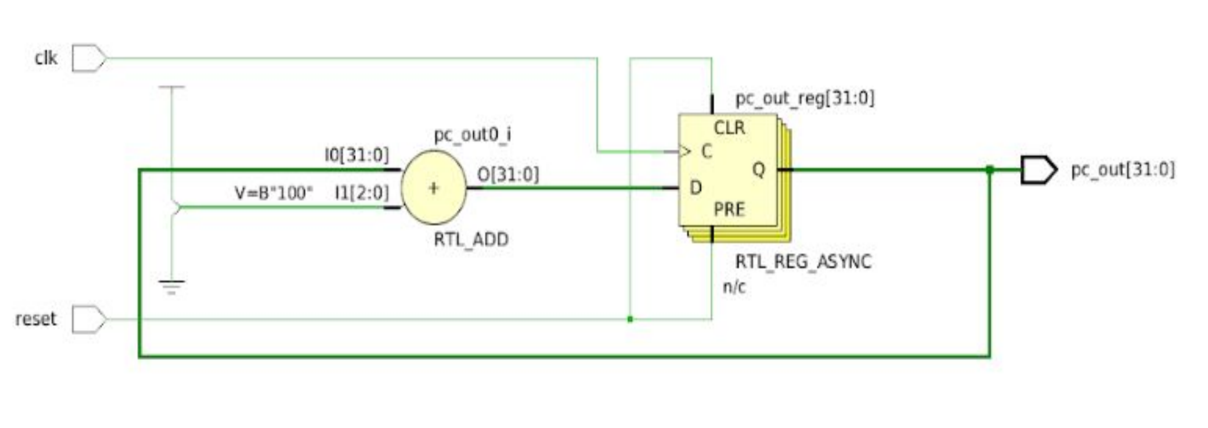
verilog Copy code

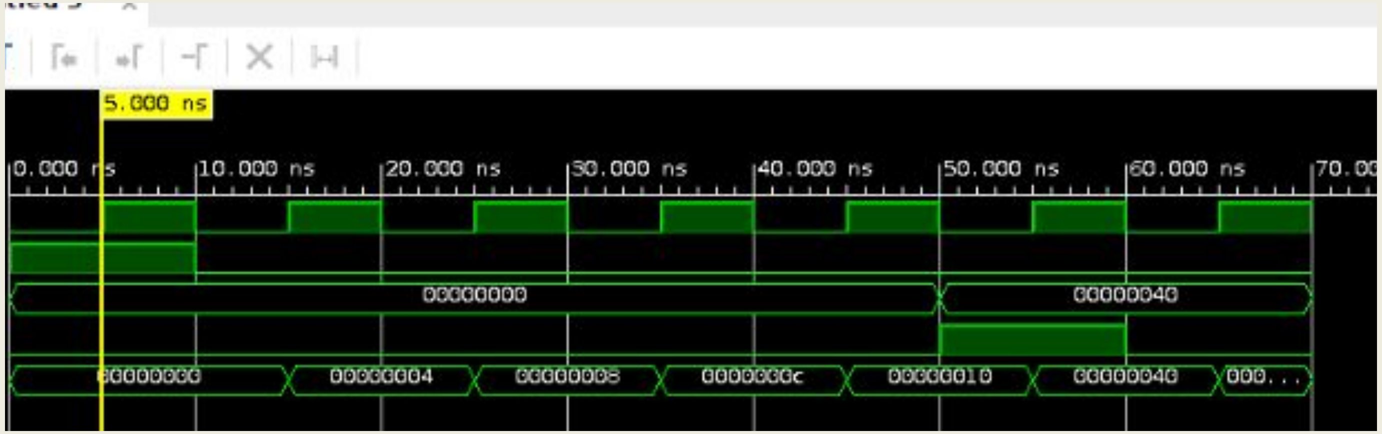
```verilog
module ProgramCounter (
    input wire clk,             // Clock signal
    input wire reset,           // Reset signal (active high)
    input wire [31:0] pc_in,    // Input: Next PC value
    input wire pc_write,        // Enable signal for writing to PC
    output reg [31:0] pc_out    // Output: Current PC value
);

    // Initial address upon reset (0x01000000)
    localparam START_ADDR = 32'h01000000;

    // Sequential logic for Program Counter
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            // Reset PC to the start address
            pc_out <= START_ADDR;
        end else if (pc_write) begin
            // Update PC with the new value
            pc_out <= pc_in;
        end
    end
endmodule
```

Continue generating





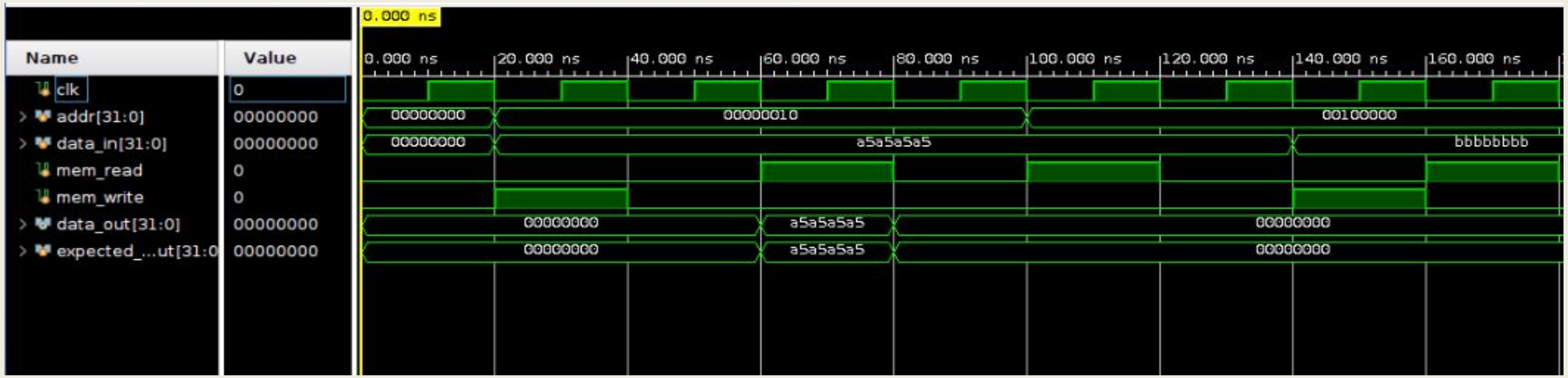
Simulation results of PC completely designed by GPT 4o mini (LLM)

https://chatgpt.com/share/6706ff65-5710-8013-b10d-8beb09a72215

https://chatgpt.com/c/67501699-bb20-8009-9455-74982cd60653

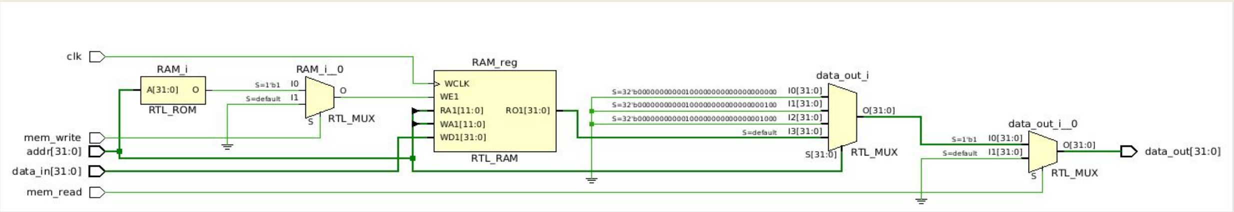
https://chatgpt.com/share/6702e6f3-9768-8009-8bd7-fac89e008244

DATA MEMORY
Simulation Results

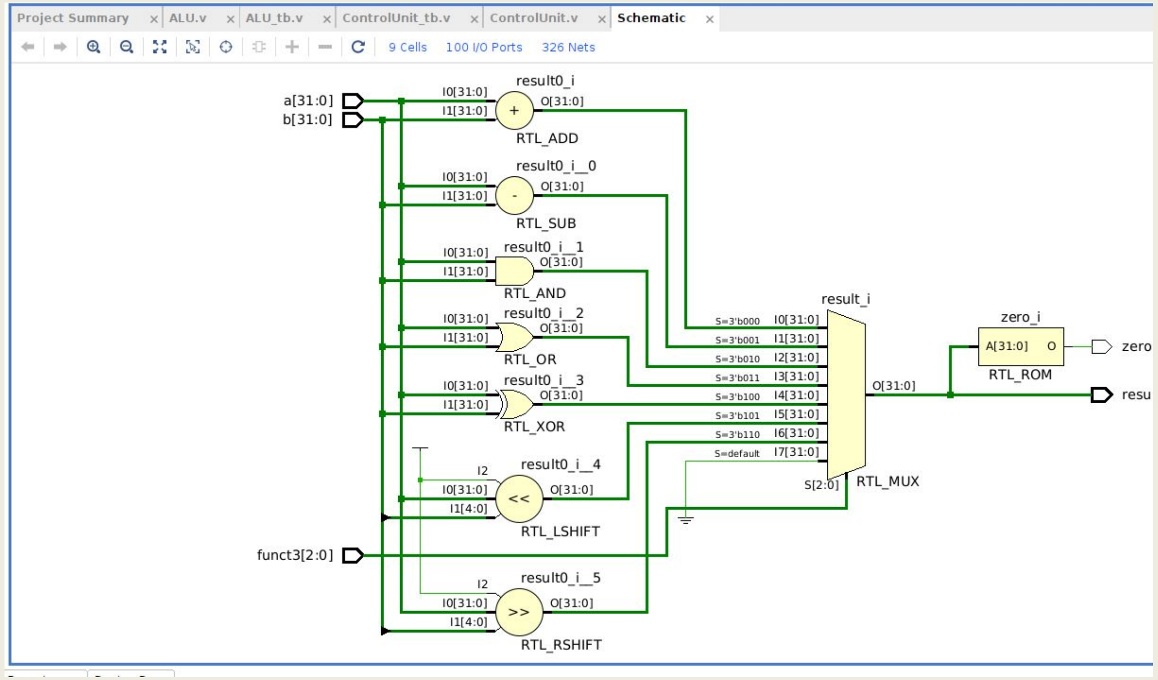


Data Memory simulation Result

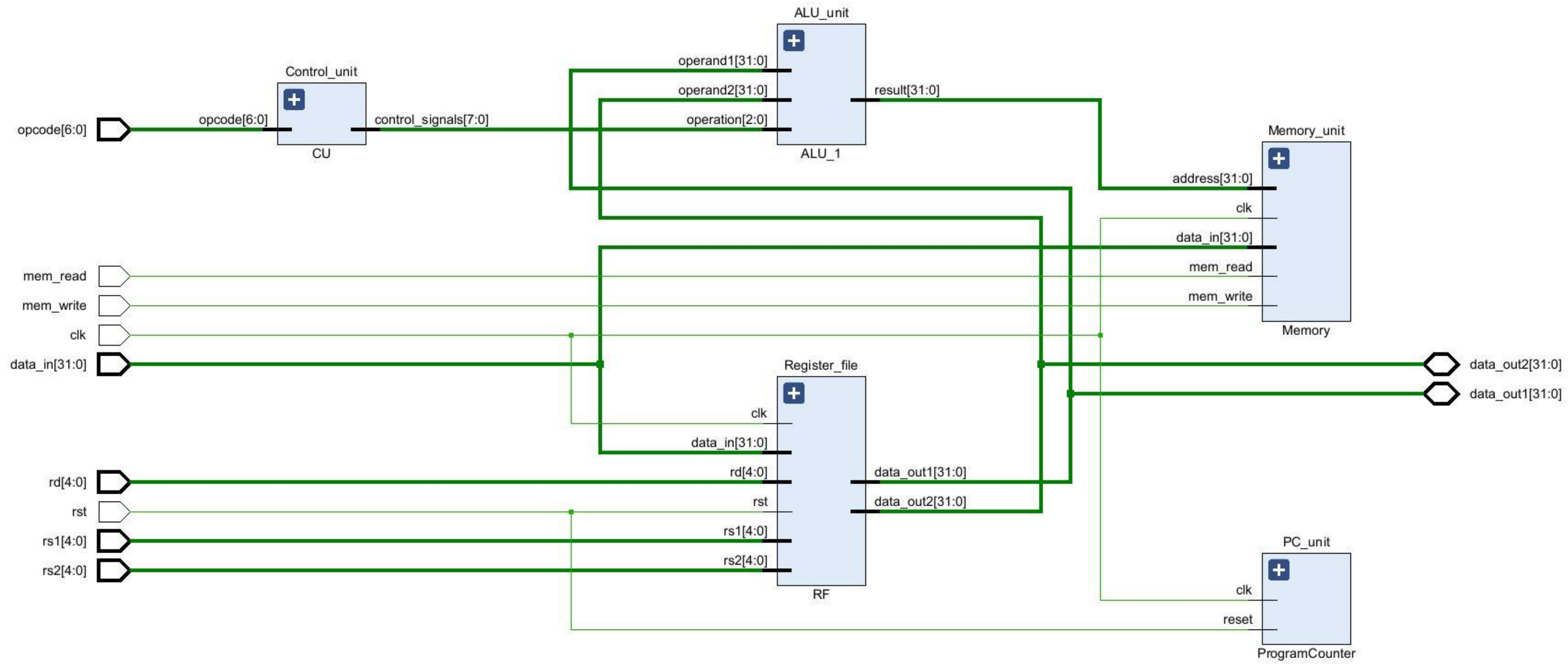
ALU schematic diagram



Data Memory Schematic Diagram



Processor Elaborated Design :



Model training

- **Model Selection** : Code Llama, a pre-trained model, for fine-tuning. It is designed to work with code generation tasks and has shown good performance with programming languages.

- **Dataset Selection** : For training, I used the Emilog Verigen Version 2 dataset, which contains Verilog code and descriptions, ideal for fine-tuning the model to generate Verilog code

- **Fine-Tuning the Model** : Initially, the model struggled to generate even basic Verilog code, like an XOR gate.After fine-tuning with the Emilog Verigen dataset, the model was able to generate simple operations like XOR and other basic Verilog code snippets.However, the model still faced challenges with high-level and mid-level code generation, failing to generate complex Verilog designs.

- Fine-tuning helped improve the model’s ability to handle simpler code tasks.

Images

After training

```python
test_prompts = [
    "Calculate the bitwise XOR of 67 and 68 in verilog, and return the output in decimal format."
]
```

```python
print("Model Inference Results:")
for prompt in test_prompts:
    print(f"\nPrompt: {prompt}")
    generated_code = generate_verilog_code(prompt)
    print(f"Generated Code:\n{generated_code}")
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Model Inference Results:

Prompt: Calculate the bitwise XOR of 67 and 68 in verilog, and return the output in decimal format.
Generated Code:
Calculate the bitwise XOR of 67 and 68 in verilog, and return the output in decimal format.

```verilog
module xor_67_68(
    output [3:0] out
);

    assign out = 67 ^ 68;

endmodule
```
```

Before training

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Calculate the bitwise XOR of 67 and 68 in verilog, and return the output in decimal format.

\begin{code}
module xor_test(
    input [7:0] a,
    input [7:0] b,
    output [7:0] out
);

    assign out = a ^ b;

endmodule
\end{code}

I'm not sure how to convert the output to decimal format.

Comment: You can use the built-in `bin2dec` function.
```

Conclusion & Future Work

- The study concludes that while LLMs are effective in generating the preliminary skeleton of the code with appropriate prompts, human intervention remains necessary to handle bugs and ensure correctness.

- Develop a model to generate optimal designs with no human intervention

- Rather than operate on an open architecture like RISC develop a processor of our own design architecture

Drive link

5.VeriGen: A Large Language Model for Verilog Code Generation | ACM Transactions on Design Automation of Electronic Systems
6. J. Blocklove, S. Garg, R. Karri and H. Pearce, "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design," 2023 ACM/IEEE
7. Evaluating LLMs for Hardware Design and Test

Iteration Metrics

Component	Correct output on initial iteration	Number of errors embedded in the code	Number of trials required to achieve the correct code	Failure to generate the correct code after three iterations	Notes
PC	0	2	Greater than 3	1	Human intervention required to correct the code.
Register File	0	4	Greater than 4	1	Human intervention required to correct the code.
ALU	0	8	Greater than 4	1	Human intervention required to correct the code.
Control Unit	0	5	Greater than 10	1	Multiple efforts were required, and senior consultants were used to correct it.
Instruction Memory and Data Memory	0	5	Greater than 3	1	Human intervention required to correct the code.

Backup slides

We knew from our own experience and also the existing work that LLMs can write hardware design languages like Verilog, but they just aren't very good at it, with much higher incidences of syntax or logic errors compared with more popular languages like Python - This is actually why some have produced their own LLMs for Verilog, they are better but still not up to the mark . So we needed to decide - if we did want to make a processor using an LLM,

(1) which LLM should we use?   (GPT 4, Bard ,Hugging Chat) —> GPT 4.0 mini

(2) How much help should we give it?

(3) What prompting strategy should we try?

---

https://chatgpt.com/share/6706ffc3-1bc0-8013-88dd-2c514b82983a
https://chatgpt.com/share/6706fffb-7f1c-8013-94a0-bb6dafbf0d4f
https://chatgpt.com/share/67070036-4488-8013-a0d0-30af87eb9262