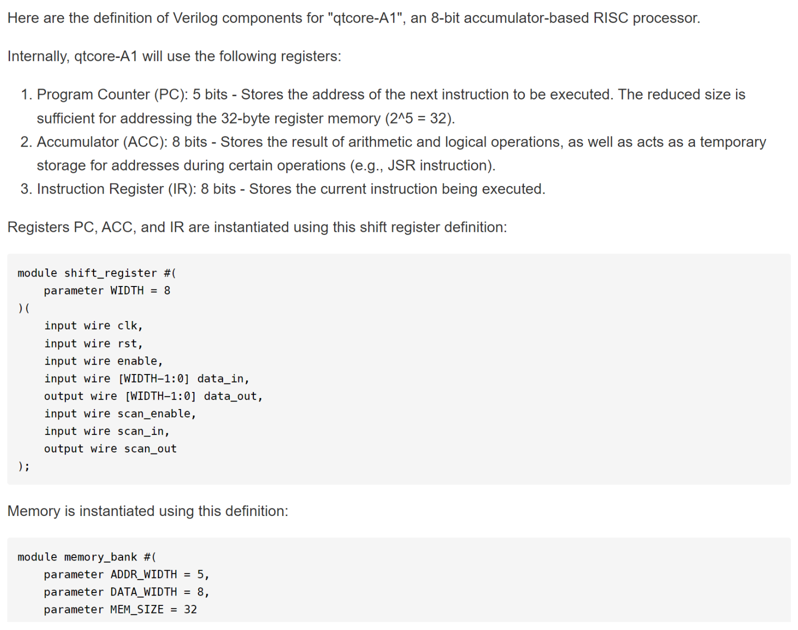
08 - DATAPATH COMPONENTS
12 - DATAPATH BRANCH UPDATE
16 - DATAPATH BUG FIXING AND SMALL CHANGES

---

08 - DATAPATH COMPONENTS

User #1:



Here are the definition of Verilog components for "qtcore-A1", an 8-bit accumulator-based RISC processor.

Internally, qtcore-A1 will use the following registers:

1. Program Counter (PC): 5 bits - Stores the address of the next instruction to be executed. The reduced size is sufficient for addressing the 32-byte register memory (2^5 = 32).
2. Accumulator (ACC): 8 bits - Stores the result of arithmetic and logical operations, as well as acts as a temporary storage for addresses during certain operations (e.g., JSR instruction).
3. Instruction Register (IR): 8 bits - Stores the current instruction being executed.

Registers PC, ACC, and IR are instantiated using this shift register definition:

```
module shift_register #(
    parameter WIDTH = 8
)(
    input wire clk,
    input wire rst,
    input wire enable,
    input wire [WIDTH-1:0] data_in,
    output wire [WIDTH-1:0] data_out,
    input wire scan_enable,
    input wire scan_in,
    output wire scan_out
);
```

Memory is instantiated using this definition:

```
module memory_bank #(
    parameter ADDR_WIDTH = 5,
    parameter DATA_WIDTH = 8,
    parameter MEM_SIZE = 32
```

Let's finish off the top-level module. It will contain the following components: the ALU, the memory, the PC, ACC, and IR, as well as the control unit.
Each component will need to be wired together. This will require declaring some wires. What wires will be needed?

User #2:
Can you now write Verilog code to instantiate the ALU and any multiplexers that it needs? Use the wire definitions you just provided to provide the necessary I/O, as well as suitable top-level I/O from the module definition.

User #3:
For the immediate value, it should be the bottom 4 bits of IR extended by 4 zeros, can you update the definition of the immediate wire?

User #4:
Looks good. Can you now **instantiate** the PC, IR, and ACC and their multiplexers? Assume it will be put after the `// Remaining module instantiations and connections go here`, you don't need to rewrite the rest of the file.

User #5:
If you are going to use always blocks to set pc_data_in and acc_data_in, they

can't be the wire type - can you fix their definitions?

User #6:
If your ACC multiplexer, it assumes that pc_data_out is 8 bits, but that's not right - can you rewrite that multiplexer with the correct width for pc_data_out and extend it with zeros to make 8 bits?

User #7:
The PC's scan_in is coming from the wrong wire, can you fix that?

User #8:
Looking good! Can you now **instantiate** the memory unit and the multiplexers that it requires?

User #9:
If you are going to use an always block to set memory_address it can't be the wire type - can you fix the definitions?

User #10:
Looks good. Can you now **instantiate** the control unit?

User #11:
The control unit's scan_in should come from the top level scan_in?

User #12:
We have just a few things left! Can you write code to provide the zero_flag, and can you connect the appropriate signal to the top-level scan_out?

Summary:

# Restart	0
# Fix	7/12
# Instantiate	3/12

错误类型：
immediate value 的低位补 0
always 中不可以使用 wire 类型
信号的位宽与初始 prompt 设置的位宽不匹配（两次）
连线错误（两次）
遗漏信号

12 - DATAPATH BRANCH UPDATE

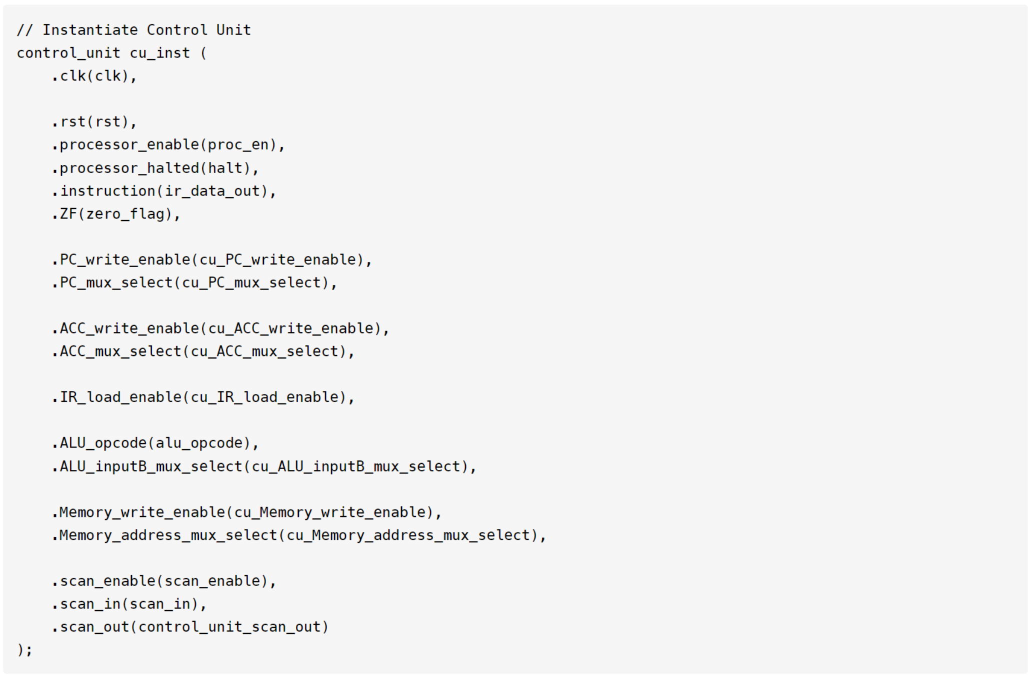
User #1:

Can we update the PC multiplexer? The BEQ_FWD and BNE_FWD instructions have been updated to jump an extra instruction. Here's the definition of the PC control signals in the control unit documentation:

16 - DATAPATH BUG FIXING AND SMALL CHANGES

User #1:

In the control unit instantiation, I think the signal connected to ALU_opcode is wrong:



```
// Instantiate Control Unit
control_unit cu_inst (
    .clk(clk),

    .rst(rst),
    .processor_enable(proc_en),
    .processor_halted(halt),
    .instruction(ir_data_out),
    .ZF(zero_flag),

    .PC_write_enable(cu_PC_write_enable),
    .PC_mux_select(cu_PC_mux_select),

    .ACC_write_enable(cu_ACC_write_enable),
    .ACC_mux_select(cu_ACC_mux_select),

    .IR_load_enable(cu_IR_load_enable),

    .ALU_opcode(alu_opcode),
    .ALU_inputB_mux_select(cu_ALU_inputB_mux_select),

    .Memory_write_enable(cu_Memory_write_enable),
    .Memory_address_mux_select(cu_Memory_address_mux_select),

    .scan_enable(scan_enable),
    .scan_in(scan_in),
    .scan_out(control_unit_scan_out)
);
```

User #2:

I want to change the top level definition of the accumulator_microcontroller to the following.

The MEM_SIZE, btn_in and led_out should be appropriately connected to the memory. Can you update the previous code.

User #3:

Can we simplify the PC mux to just make the default case be the last case?

User #4:

Can you make the last case be the default case with the default keyword?

User #5:

Can we update the Memory Address Multiplexer to use the pc_data_out as the default case?

User #6:

Can we update the ACC multiplexer to make the PC data out be the default case ?

Summary：

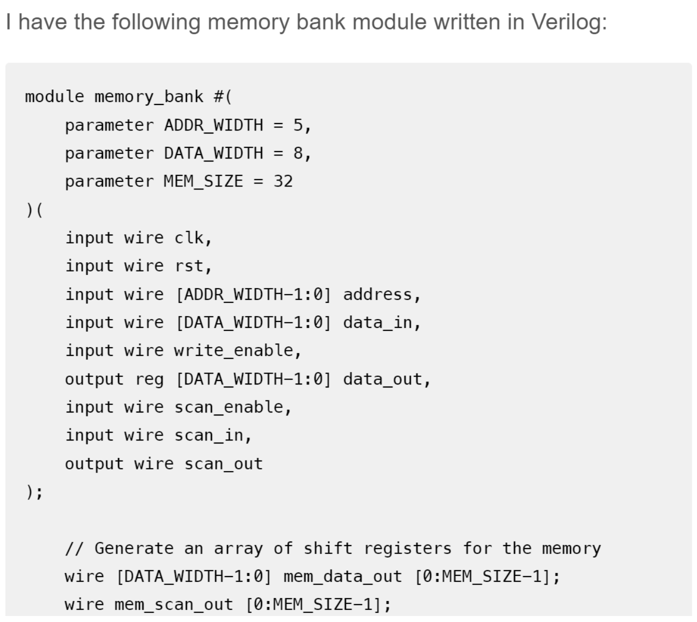
12 和 16 两部分主要用来手动使用“update”对不同的模块做出细微的调整。整体来看：

- ChatGPT 在 Datapath 阶段的出错率很高，在 DATAPATH COMPONENTS 中出错率超过 50%
- 错误类型主要类型为丢失信号，位宽不匹配和连线错误等
- 过程中重启次数 # Restart = 0

---

14 - MEMORY MAPPED COMPONENTS

User #1:



I have the following memory bank module written in Verilog:

```
module memory_bank #(
    parameter ADDR_WIDTH = 5,
    parameter DATA_WIDTH = 8,
    parameter MEM_SIZE = 32
)(
    input wire clk,
    input wire rst,
    input wire [ADDR_WIDTH-1:0] address,
    input wire [DATA_WIDTH-1:0] data_in,
    input wire write_enable,
    output reg [DATA_WIDTH-1:0] data_out,
    input wire scan_enable,
    input wire scan_in,
    output wire scan_out
);

    // Generate an array of shift registers for the memory
    wire [DATA_WIDTH-1:0] mem_data_out [0:MEM_SIZE-1];
    wire mem_scan_out [0:MEM_SIZE-1];
```

User #2:
Looks good! Can you make the following changes for me:

1. declare a wire to take the scan_out from the led_shift_register to the scan_in of the btn_shift_register
2. Make the led_shift_register read from the upper 7 bits of data_in, not the lower bits
3. Make the data_out for IO_ADDR have the btn_data_out at the LSB, not the MSB

User #3:
Looks very good - one last change, the btn_shift_register enable should be 1'b1 (i.e. enabled), it is active high not active low

User #4:
One other change - can you reverse the order of the btn_shift_register and the led_shift_register in the scan chain, so that the btn_shift_register comes first and the led_shift_register is the one connected to the top-level module scan_out?

User #5:
I think it would be more efficient if instead of using a for loop to index the mem_data_out array, you just indexed it directly. Can you rewrite just the //Read

operation code?

User #6:
Can we actually make it return "00000001" for all memory addresses outside the range?

User #7:
Can you re-write the whole module now? Remember to include the led_out signal.

---

17 - MEMORY MAPPED CONSTANTS

User #1:
I want to add a few more constant memory-mapped values in the //Read operation outputs. At IO_ADDR + 1, can you return the value IO_ADDR + 2? This will be useful to load as an offset address later.

User #2:
Now I want to add, starting at IO_ADDR + 2, the cases for a 7segment display driver. These are the following values for each digit:
0: segments = 7'b0111111;
1: segments = 7'b0000110;
2: segments = 7'b1011011;
3: segments = 7'b1001111;
4: segments = 7'b1100110;
5: segments = 7'b1101101;
6: segments = 7'b1111100;
7: segments = 7'b0000111;
8: segments = 7'b1111111;
9: segments = 7'b1100111;
Put the digit 0 at IO_ADDR + 2 and increment appropriately.

User #3:
I like that! Can you merge into the case statement the if statement for IO_ADDR and IO_ADDR + 1?

User #4:
Last change - can all the 7 segment values be concatenated with a zero bit at the LSB?

Summary:
所有的部分都是 change 部分

15 - SHIFT REGISTER BUG FIX
User #1:
I have the following shift_register written in Verilog:

```
module shift_register #(
    parameter WIDTH = 8
)(
    input wire clk,
    input wire rst,
    input wire enable,
    input wire [WIDTH-1:0] data_in,
    output wire [WIDTH-1:0] data_out,
    input wire scan_enable,
    input wire scan_in,
    output wire scan_out
);

    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            internal_data <= {internal_data[WIDTH-2:0], scan_in};
        end
    end
```

Unfortunately, it doesn't work if the WIDTH is set to 1. Can you fix it?

User #2:
Thanks! Can you also make it so scan_enable has a higher priority than the normal enable?