

Project2 RV12 ISA Formal

組別: Group23

組員: M16131111 童品綸、N26134992 賴郁明

內容

1.	Design Architecture	2
2.	Datapath	3
2.1	SRAI.....	3
2.2	BLTU.....	4
2.3	JAL	4
2.4	LW	5
2.5	AUIPC	6
3.	Pipeline Follower	6
4.	SVA	7
	SRAI assertion:	7
	LW assertion:.....	10
	AUIPC assertion:	11
	JAL assertion:	13
	BLTU assertion:	15
5.	Bugs Detection	18
6.	Steps to Resolve the Bug	20
7.	Result	21

1. Design Architecture

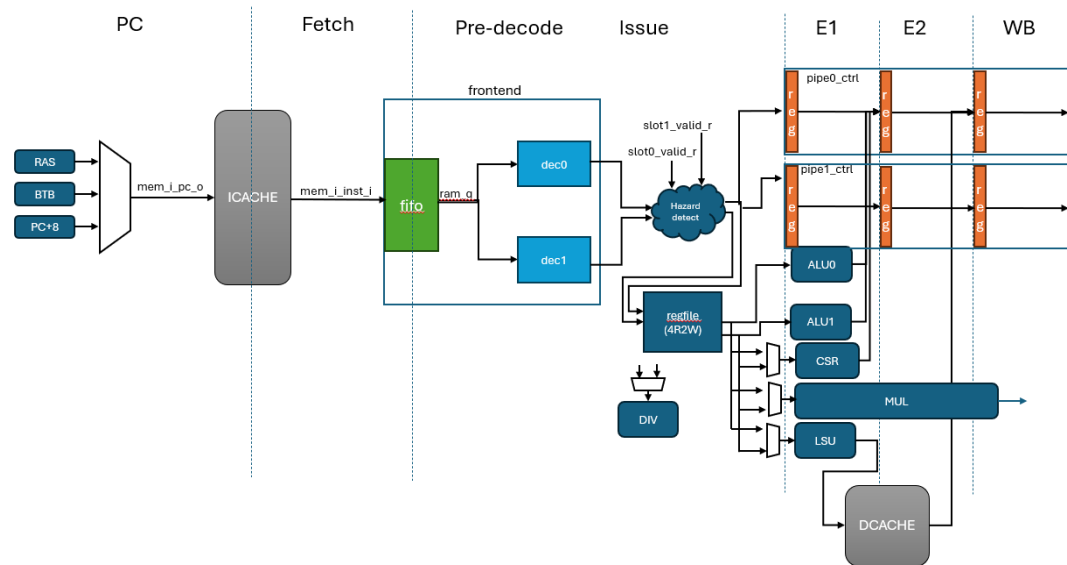


圖 1、CPU core architecture

這次驗證的 CPU 架構為 6 級 pipeline，支援 RV32I 指令集，且支援分支預測功能。其中的 PC stage，負責進行下一個要抓取指令之 PC 值計算，並在發生分支或跳躍時正確的切換下個 PC 位址。pre-decode stage + issue 負責對指令進行預解碼，還有拿解碼指令之地址去暫存器檔案抓地址，還有接著是判斷兩個指令的資料相依性來判斷是否可以同時執行指令，以及判斷前面 branch 預測結果跟實際結果是否相同，不相同則會清除以及加載本來該執行之指令 PC 到 PC stage，在 E1 stage 進行指令運算如 ALU、CSR 等等，在 E2 stage 進行的是去記憶體讀取資料以及寫入資料的動作，WB stage 則是將前面不論是載入記憶體資料或者運算結果等等存回目標暫存器的階段。

2. Datapath

2.1 SRAI

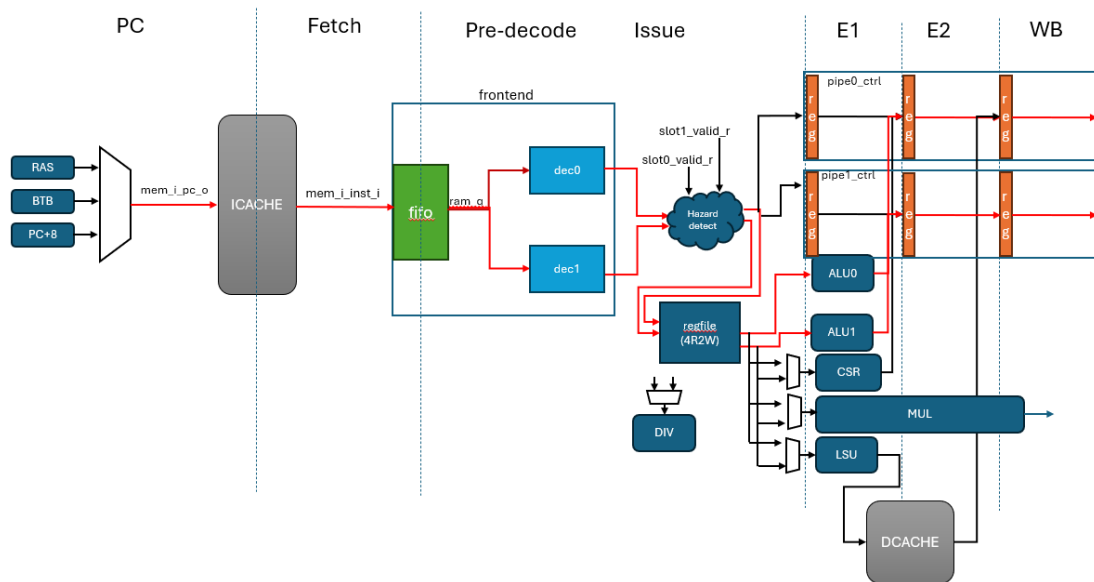


圖 2、SRAI datapath

SRAI 指令經過 pre-decode 以及 issue 解碼該指令獲得立即值，在 E1 時會透過算術邏輯單元進行位移，在 wb 時在寫回目標暫存器。

2.2 BLTU

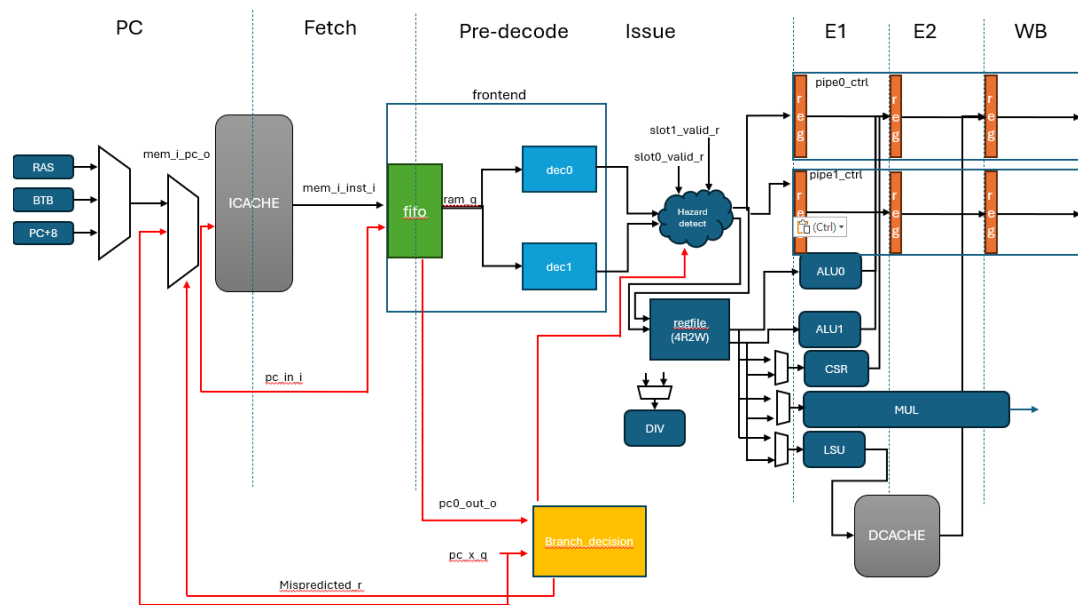


圖 3、BLTU datapath

BLTU 指令在一開始會先由於分支預測先猜測出可能結果，在 Pre-decode 以及 issue 時在看預測跟實際是否相符，相符則沒問題，不符則要清洗過去已經進入流水線之指令，並且載入實際該被執行之指令。

2.3 JAL

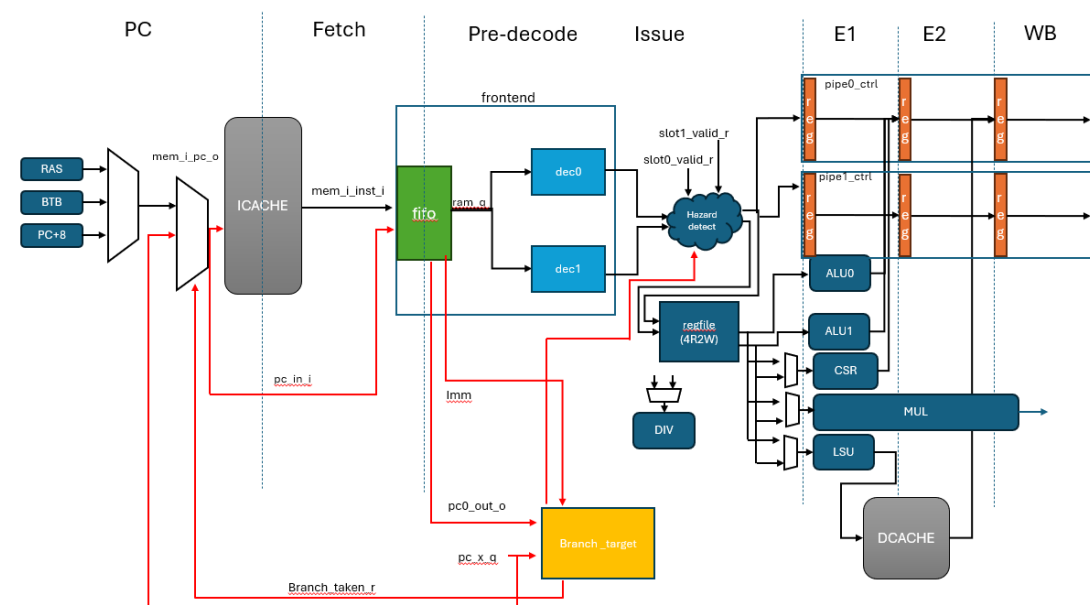


圖 4、JAL datapath

JAL 指令會在 pre-decode 算出跳躍之指令，並且進行跳躍，在 wb 時會將原本 pc 值寫入目的暫存器。

2.4 LW

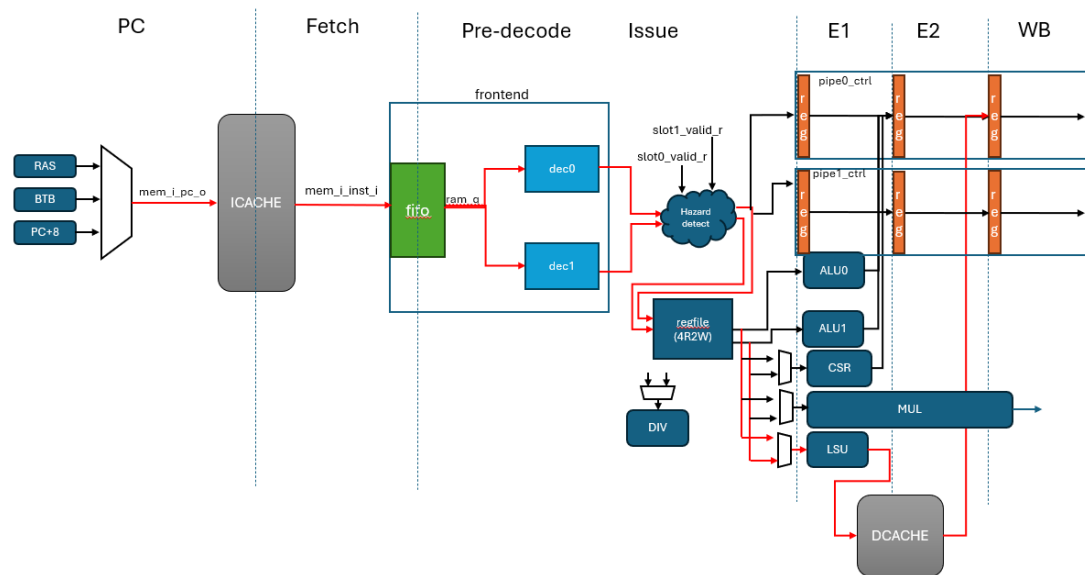


圖 5、LW datapath

LW 指令會 Pre-decode+Issue 算出應該去哪個記憶體位置載入資料，並且在 E2 時將要載入資料寫回暫存器檔案。

2.5 AUIPC

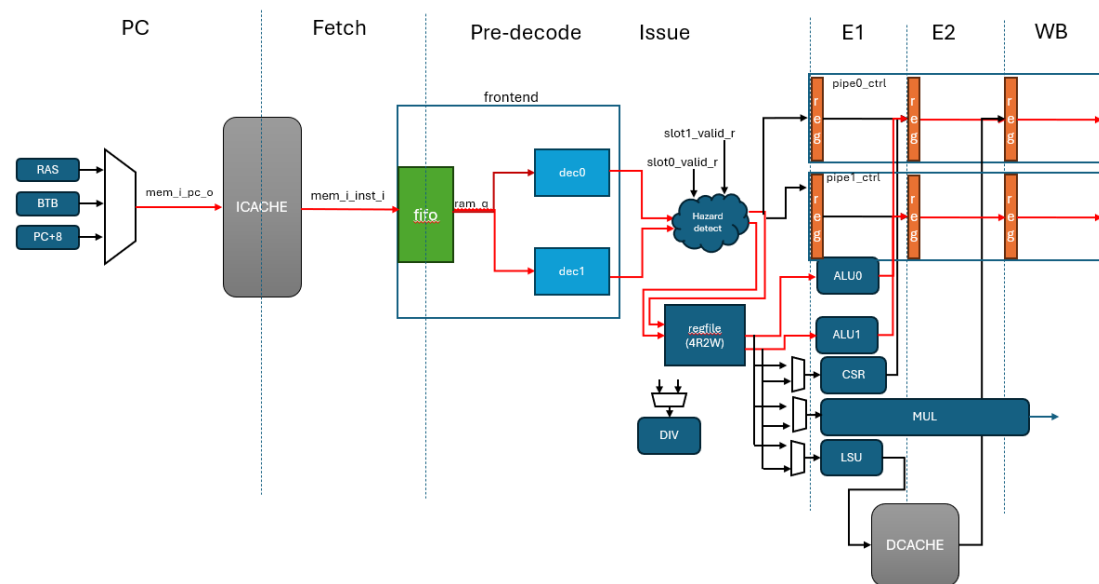


圖 6、AUIPC datapath

AUIPC 指令會在經過 pre-code 以及 issue 解碼該指令獲得立即值，在 E1 時會透過算術邏輯單元跟 PC 進行相加，在 wb 時在寫回目標暫存器。

3. Pipeline Follower

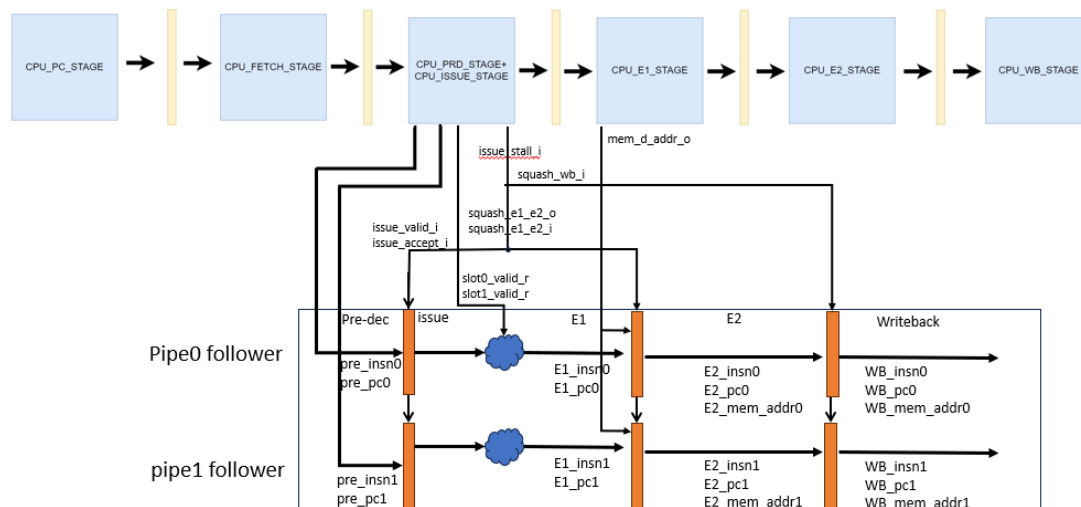


圖 7、pipeline follower

在進行 ISA formal 驗證時，我們會在驗證環境中建立一個 pipeline follower 來記錄 CPU pipeline 中各個 stage 在執行的指令，透過引入 CPU 中的各種控制信號，使 pipeline follower 各個 stage 中的指令與 CPU 同步。

且須注意因為這是 dual issue，所以在 pipeline 上一次是處理兩條指令時由於有些指令會有相依性，有些是沒有，所以他內部控制訊號會決定那些可以同時做，那些沒辦法同時做，來去控制後面指令能不能在流水線中繼續傳遞下去，由於兩條指令控制訊號不同，因此要分兩個 pipeline follower 來傳遞指令，之後跟之前要 single issue 的 CPU 一樣只要在 wb stage 驗證其指令解碼之正確性即可。因為在驗證的指令到達 wb stage 時，該指令之前所完成的指令皆以執行完畢並寫入 register file，讓我們在計算 golden 值時可以避免 data dependency 的問題，因此可以根據解碼該指令的內容計算出該指令正確執行的 golden 值，並在該階段進行 assertion 的驗證。

在 pipeline follower 中除了有指令進行傳遞外，我們也將 program counter 與指令一同傳下去，在指令傳到 wb stage 時，該指令對應的 pc 也會傳到，因此在進行 branch 驗證時可以使用該 pc 計算 golden 值。

4. SVA

SRAI assertion:

第一步先給 GPT Project1 的 isa.sva 引導它生出 biriscv 的 SRAI assertions 但此時會有明顯錯誤，如在 assertion 中 past 的使用會有錯誤，因此必須繼續引導它修正，告訴它在驗證時是使用過去的 golden，與現在的 result 相比。

最後在模擬時使用 counterexample waveform 及一些提示讓 GPT 找到錯誤的地方並進行修正。

Prompt1:



isa.sva
檔案

現在我要你生成驗證SRAI指令的assertions 請根據RV12的範例，
isa.sva，生成驗證biriscv SRAI指令的assertions

Result1:

```
// 定義 SRAI 指令條件
assign is_SRAI_insn = (wb_insn_opcode == 7'b0010011) &&
                      (wb_insn_func3 == 3'b101) &&
                      (wb_insn_func7 == 7'b0100000);

// 計算黃金值 (golden value)
assign golden_SRAI = $signed(wb_rs1_data) >>> wb_insn_shamt;

// 從寄存器檔案讀取結果
assign result_SRAI = rf[wb_insn_rd_addr];

// SRAI 驗證的 property
SRAI_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn | => (golden_SRAI == $past(result_SRAI)))
    );
```

Prompt2:

assertion有錯誤，在驗證時應該是使用過去的golden比較從regfile
取出的result

Result2:


```

assign RESULT_insn0_SRAI = {W0_insn0_u_data};

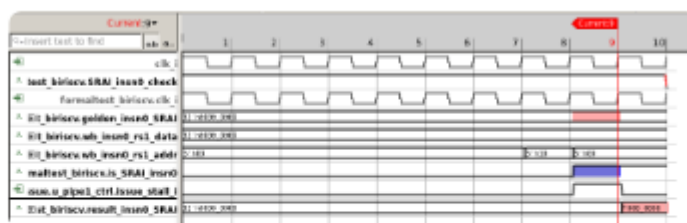
// SRAI 驗證的 property
SRAI_insn0_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn0 |> ($past(golden_insn0_SRAI) == result_insn0_SRAI))
);

SRAI_insn1_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn1 |> ($past(golden_insn1_SRAI) == result_insn1_SRAI))
);

endmodule

```

Prompt3:



請你根據波型找到SVA可能錯誤的地方，似乎在stall的時候不該驗證該指令

Result3:

根據波形圖和描述，SVA 的問題可能出現在 stall 條件下仍然進行了 SRAI 指令的驗證。這種情況下，因為 pipeline 被 stall，SRAI 指令不應該觸發驗證。

以下是修正後的 SVA 寫法，考慮到 `issue_stall_i` 信號的影響：

修正的 SRAI Assertions

```

systemverilog
// SRAI 驗證的 property
SRAI_insn0_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn0 && !u_pipe1_ctrl.issue_stall_i |> $past(golden_insn0_SRAI) == result_i
);

SRAI_insn1_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn1 && !u_pipe1_ctrl.issue_stall_i |> $past(golden_insn1_SRAI) == result_i
);

```

LW assertion:

第一步先給 GPT Project1 的 isa.sva 引導它生出 biriscv 的 LW assertions，但此時會有明顯錯誤，如在 assertion 中是比較當前 cycle 而非下一個 cycle，也不必使用到 \$past 函數，因此使用 counterexample waveform 及一些提示讓 GPT 找到錯誤的地方並進行修正。

Prompt1:



現在我要你生成驗證LW指令的assertion請根據RV12的範例，isa.sva，生成驗證biriscv LW指令的assertions

Result1:

LW 指令驗證 Assertions

以下是 LW 指令的 `insn0` 和 `insn1` 的 assertion 生成：

```
systemverilog 複製程式碼

// LW 指令的 Assertions
LW_insn0_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_LW_insn0 && !u_pipe0_ctrl.issue_stall_i | =>
        $past(golden_insn0_LW) == result_insn0_LW)
);

LW_insn1_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_LW_insn1 && !u_pipe1_ctrl.issue_stall_i | =>
        $past(golden_insn1_LW) == result_insn1_LW)
);
```

Prompt2:



先前生出的SVA有錯誤，counter example 波型如圖所示，LW應該只需在指令到達WB stage時，取得該cycle時之golden與result驗證是否相同，

Result2:

驗證當前 golden_LW 與 result_LW 是否相符：

更新後的 LW 指令 Assertion

```
systemverilog
// 定義 LW 指令判斷
assign is_LW_insn = (wb_insn_opcode == 7'b0000011) && (wb_insn_func3 == 3'b010) && (!core

// 計算 golden 值
assign golden_LW = wb_rs1_data + wb_I_imm;

// 驗證 Assertion
LW_check: assert property(
    @(posedge HCLK) disable iff (!HRESETn)
    (is_LW_insn && !wb_stall_o_delay) |-> (golden_LW == result_LW)
);
```

AUIPC assertion:

第一步先給 GPT Project1 的 isa.sva 引導它生出 biriscv 的 AUIPC assertions
但此時會有明顯錯誤，如在 assertion 中 past 的使用會有錯誤，因此必須繼續引導它修正，告訴它在驗證時是使用過去的 golden，與現在的 result 相比。

Prompt1:



isa.sva

檔案

現在我要你生成驗證AUIPC指令的assertion請根據RV12的範例，isa.sva，生成驗證biriscv AUIPC指令的assertions

Result1:

systemverilog

複製程式碼

```
// 判斷 AUIPC 指令
assign is_AUIPC_insn = (wb_insn_opcode == 7'b0010111) && (!core.wb_unit.wb_insn_o.bubble)

// 計算 AUIPC 的 golden 值
assign golden_AUIPC = wb_pc + wb_U_imm;

// 寫回結果
assign result_AUIPC = core.int_rf.rf[wb_insn_rd_addr_reg];

// AUIPC Assertion
AUIPC_check: assert property(
    @(posedge HCLK) disable iff(!HRESETn)
    (is_AUIPC_insn && !wb_stall_o_delay) |-> (golden_AUIPC == result_AUIPC)
);
```

Prompt2:

assertion有錯誤，在驗證時應該是使用過去的golden比較從reghfile取出的result

Result2:

修訂後的 AUIPC Assertion

```
systemverilog 複製程式碼

// 定義 AUIPC 指令觸發條件
assign is_AUIPC_insn = (wb_insn_opcode == 7'b0010111) && (!core.wb_unit.wb_insn_o.bubble);

// 計算 golden 值 (PC + imm)
assign golden_AUIPC = wb_pc + wb_U_imm;

// 從寄存器文件讀取的結果值
assign result_AUIPC = core.int_rf.rf[wb_insn_rd_addr_reg];

// Assertion for AUIPC 指令
AUIPC_check: assert property(
    @(posedge HCLK) disable iff(!HRESETn)
    (is_AUIPC_insn && !wb_stall_o_delay) | => ($past(golden_AUIPC) == result_AUIPC)
);
```

```
assign result_insn0_SRAI = rf[wb_insn0_rd_addr];

// SRAI 驗證的 property
SRAI_insn0_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn0 | => ($past(golden_insn0_SRAI) == result_insn0_SRAI))
);

SRAI_insn1_check: assert property(@(posedge clk_i) disable iff(!rst_i)
    (is_SRAI_insn1 | => ($past(golden_insn1_SRAI) == result_insn1_SRAI))
);

endmodule
```

JAL assertion:

第一步先給 GPT Project1 的 isa.sva 引導它生出 biriscv 的 JAL assertions

可以看出 GPT 生出的 SVA 不管是 Implication 前的條件到取得來比較的 pc 值訊號都是有問題的，因此必須繼續引導他修正。

在這種程式架構上不符合 isa formal 行為的明顯錯誤，我們修正的過程透過下 prompt 告知 GPT 在 implication 的條件上與我給的範例不符，並告知缺失的部分及程式正確運作的方式，他就會生出比較正確的版本，經過幾次的修正，我們透過觀察比較範例 sva 跟 GPT 生出來的版本的差異，直到程式的架構符合 isa formal 驗證的行為。

Prompt1:



- 9 現在我要你生成驗證JAL指令的assertion請根據RV12的範例，isa.sva，生成驗證bircsv JAL指令的assertions

Result1:

JAL Assertion Code

```
systemverilog 複製程式碼

// 定義 JAL 指令觸發條件
assign is_JAL_insn = (wb_insn_opcode == 7'b1101111) && (!core.wb_unit.wb_insn_o.bubble) &&

// 計算 golden 值
assign golden_PC_JAL = (wb_pc + wb_insn_immJ) & 32'hffff_fffc; // 目標地址 (對齊)
assign golden_rd_JAL = wb_pc + 32'h4; // 返回地址 (PC + 4)

// 從寄存器文件讀取的結果值
assign result_rd_JAL = core.int_rf.rf[wb_insn_rd_addr_reg]; // 寄存器寫入的值
assign result_PC_JAL = core.wb_unit.wb_pc_o; // 核心當前 PC

// JAL assertion: 驗證目標 PC 和寄存器寫回數據
JAL_PC_check: assert property(
    @(posedge HCLK) disable iff(!HRESETn)
    (is_JAL_insn && !wb_stall_o_delay) | => ($past(golden_PC_JAL) == result_PC_JAL)
);

JAL_rd_check: assert property(
    @(posedge HCLK) disable iff(!HRESETn)
    (is_JAL_insn && !wb_stall_o_delay) | => $past(golden_rd_JAL) == result_rd_JAL
);
```

Prompt2:

根據範例SVA的JAL驗證方式，須符合三個條件，第一個是JAL_trigger會在第一個JAL指令進入WB stage時trigger並直到下個合法指令進入WB stage才會放下，第二個是要確認當前指令是否合法，以及要確定被驗證的PC不等於第一個進入WB stage的JAL指令

Result2:

```

// 定義 JAL trigger
always_ff @(posedge HCLK or negedge HRESETn) begin
    if (!HRESETn) begin
        JAL_trigger <= 1'b0;
        JAL_original_pc <= 32'd0;
        JAL_golden_pc_reg <= 32'd0;
    end else if (is_JAL_insn && wb_no_bubble) begin
        JAL_trigger <= 1'b1; // 第一個 JAL 指令進入 WB
        JAL_original_pc <= wb_pc; // 保存 JAL 指令的 PC
        JAL_golden_pc_reg <= golden_PC_JAL; // 保存 JAL 的目標地址
    end else if (wb_no_bubble && is_valid_insn) begin
        JAL_trigger <= 1'b0; // 下一個合法指令進入 WB 後釋放
    end
end

// 排除與其他指令的干擾 (例如 BLTU)
assign JAL_same_pc = (JAL_original_pc == wb_pc);

// Assertion for PC 校驗
JAL_PC_check: assert property (
    @(posedge HCLK) disable iff (!HRESETn)
    (JAL_trigger && wb_no_bubble && !JAL_same_pc) |-> (JAL_golden_pc_reg == core.wb_unit.wb_
);

// Assertion for 寄存器寫回數據校驗
JAL_rd_check: assert property (
    @(posedge HCLK) disable iff (!HRESETn)
    (is_JAL_insn && !wb_stall_o_delay) |=> ($past(golden_rd_JAL) == result_rd_JAL)
);

```

BLTU assertion:

第一步也是給 GPTproject1 的 isa.sva 叫他生出 biriscv 的 BLTU 的 assertion

可以看出 GPT 生出的 SVA 不管是 Implication 前的條件到取得來比較的 pc 值訊號都是有問題的，因此必須繼續引導他修正在這種程式架構上不符合 isa formal 行為的明顯錯誤，我們修正的過程透過下 prompt 告知 GPT 在 implication 的條件上與我給的範例不符，並告知缺失的部分及程式正確運作的方式，他就會生出比較正確的版本，經過幾次的修正，我們透過觀察比較範例 sva 跟 GPT 生出來的版本的差異，直到程式的架構符合 isa formal 驗證的行為

Prompt1:



isa.sva
檔案

現在我要你生成驗證BLTU指令的assertions 請根據RV12的範例 isa.sva，生成驗證bircsv BLTU指令的assertions

Result1:

```
// Assertions for BLTU Pipe 0
BLTU_taken_pipe0: assert property (@(posedge clk_i) disable iff (!rst_i)
    (is_BLTU_insn0 && (wb_rs1_data0 < wb_rs2_data0)) | =>
    (golden_taken_pc0 == riscv_core.u_frontend.u_pc.pc_reg)
) else $error("BLTU Pipe0 taken PC mismatch");

BLTU_no_taken_pipe0: assert property (@(posedge clk_i) disable iff (!rst_i)
    (is_BLTU_insn0 && !(wb_rs1_data0 < wb_rs2_data0)) | =>
    (golden_no_taken_pc0 == riscv_core.u_frontend.u_pc.pc_reg)
) else $error("BLTU Pipe0 not taken PC mismatch");

// Assertions for BLTU Pipe 1
BLTU_taken_pipe1: assert property (@(posedge clk_i) disable iff (!rst_i)
    (is_BLTU_insn1 && (wb_rs1_data1 < wb_rs2_data1)) | =>
    (golden_taken_pc1 == riscv_core.u_frontend.u_pc.pc_reg)
) else $error("BLTU Pipe1 taken PC mismatch");

BLTU_no_taken_pipe1: assert property (@(posedge clk_i) disable iff (!rst_i)
    (is_BLTU_insn1 && !(wb_rs1_data1 < wb_rs2_data1)) | =>
    (golden_no_taken_pc1 == riscv_core.u_frontend.u_pc.pc_reg)
) else $error("BLTU Pipe1 not taken PC mismatch");
```

Prompt2:

根據範例SVA的BLTU implication條件，應該要符合三個條件，第一個是BLTU_trigger在第一個BLTU指令進入WB stage時要trigger並直到下個合法指令進入wb stage才會放下。第二個要確認當前指令是否合法，判斷方式是當前wb stage指令不能全0，第三個是為了不要驗證到當前的BLTU指令因此被驗證指令的PC不能等於第一個進入的BLTU指令的PC，此外要使用BLTU_state來分有跳跟沒跳兩種BLTU assertion，請根據isa.sva修正

Result2:


```

// BLTU assertions for pipe0
BLTU_taken_pipe0: assert property (@(posedge clk_i) disable iff (!rst_i)
    (BLTU_trigger_pipe0 && (BLTU_state_pipe0 == 2'b01) && wb_no_bubble_pipe0 && !same_pc_p
    (golden_pc_pipe0 == riscv_core.u_issue.u_pipe0_ctrl.wb_pc_o)
);

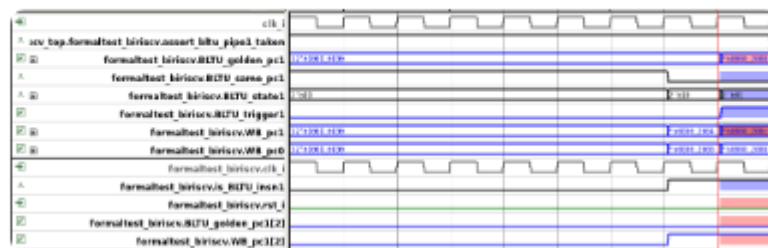
BLTU_no_taken_pipe0: assert property (@(posedge clk_i) disable iff (!rst_i)
    (BLTU_trigger_pipe0 && (BLTU_state_pipe0 == 2'b10) && wb_no_bubble_pipe0 && !same_pc_p
    (golden_pc_pipe0 == riscv_core.u_issue.u_pipe0_ctrl.wb_pc_o)
);

// BLTU assertions for pipe1
BLTU_taken_pipe1: assert property (@(posedge clk_i) disable iff (!rst_i)
    (BLTU_trigger_pipe1 && (BLTU_state_pipe1 == 2'b01) && wb_no_bubble_pipe1 && !same_pc_p
    (golden_pc_pipe1 == riscv_core.u_issue.u_pipe1_ctrl.wb_pc_o)
);

BLTU_no_taken_pipe1: assert property (@(posedge clk_i) disable iff (!rst_i)
    (BLTU_trigger_pipe1 && (BLTU_state_pipe1 == 2'b10) && wb_no_bubble_pipe1 && !same_pc_p
    (golden_pc_pipe1 == riscv_core.u_issue.u_pipe1_ctrl.wb_pc_o)
);

```

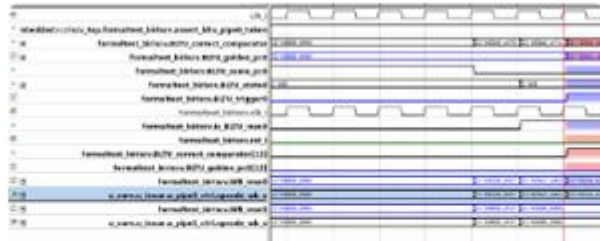
Prompt3:



根據jg產生的counter example波型有以下錯誤，golden會與WB_pc1不符，會不會比較的對象不會是WB_pc1，因為這是個dual issue CPU，所以下個合法指令一定會是2個被讀取的指令中先被讀取到的，你覺得是哪個呢？

Result3:

傳遞指令時，可能有錯誤給資料，由於直接給他看波形圖，他沒辦法從波形圖中找到重點，因此我除了波形圖外，還有將重點資訊整理給 GPT，方便他釐清問題。



嗨~我們現在要你找出這個CPU可能存在的bug，我們透過波形可以看到這個CPU的指令1之指令資料在pipeline follower跟在CPU抓取出來的有落差，所以代表指令在CPU傳遞的過程中可能遭受一些不對的邏輯來造成資料損壞，你可以把每個module看一遍，找出可能的問題嗎？

圖 9、Prompt to make GPT find the bug by searching through every module

這個是他給予他的 prompt，把波形上具體看到的問題跟他講要他找出 bug

問題分析

1. 資料傳遞錯誤：

- 波形顯示指令資料在 pipeline 追蹤的過程中第一位bit發生變化，可能是資料處理或傳遞中的操作導致資料被覆蓋或被修改。
- 在decode模組中，指令資料經過FIFO (fetch_fifo) 暫存並分別傳遞至 u_dec0 和 u_dec1，FIFO的輸出是否正確需進一步確認。

2. 資料對齊錯誤：

- FIFO的 `data0_out_o` 和 `data1_out_o` 是從 `ram_q` 中提取不同區段的資料，這可能導致對齊錯誤。
- 特別是 `data1_out_o` 的分配方式 `{1'd0, ram_q[rd_ptr_q][WIDTH-2:(WIDTH/2)]}`，在某些情況下可能引入不必要的位元或導致資料失真。

3. 初始化問題：

- `rst_i` 信號初始化時，FIFO中的資料是否正確清零，可能導致殘留數據對後續的指令資料造成干擾。

4. 無效資料覆蓋有效資料：

- 當 `fetch_in_fault_fetch_i` 或 `fetch_in_fault_page_i` 設定為1時，`fetch_in_instr_w` 被設為零（64'b0）。如果此時有效的指令資料被錯誤標記為無效，會導致pipeline處理的指令資料異常。



問題分析

1. 資料傳遞錯誤：

- 波形顯示指令資料在 pipeline 追蹤的過程中第一位bit發生變化，可能是資料處理或傳遞中的操作導致資料被覆蓋或被修改。
- 在decode模組中，指令資料經過FIFO (fetch_fifo) 暫存並分別傳遞至 u_dec0 和 u_dec1，FIFO的輸出是否正確需進一步確認。

2. 資料對齊錯誤：

- FIFO的 data0_out_o 和 data1_out_o 是從 ram_q 中提取不同區段的資料，這可能導致對齊錯誤。
- 特別是 data1_out_o 的分配方式 {1'd0, ram_q[rd_ptr_q][WIDTH-2:(WIDTH/2)]}，在某些情況下可能引入不必要的位元或導致資料失真。

3. 初始化問題：

- rst_i 信號初始化時，FIFO中的資料是否正確清零，可能導致殘留數據對後續的指令資料造成干擾。

4. 無效資料覆蓋有效資料：

- 當 fetch_in_fault_fetch_i 或 fetch_in_fault_page_i 設定為1時，fetch_in_instr_w 被設為零 (64'b0)。如果此時有效的指令資料被錯誤標記為無效，會導致pipeline處理的指令資料異常。



```
assign data0_out_o = {ram_q[rd_ptr_q][(WIDTH/2)-1:0]};  
assign data1_out_o = {1'd0, ram_q[rd_ptr_q][WIDTH-2:(WIDTH/2)]};
```

圖 10、GPT response results

這是 GPT 生成的回應，照他的回應去看 design 確實有發現有一個 bit 在指令傳遞時被錯誤給予永遠為 0，而非指令原本的值。

6. Steps to Resolve the Bug

```
assign data0_out_o = {ram_q[rd_ptr_q][(WIDTH/2)-1:0]};  
assign data1_out_o = {1'd0, ram_q[rd_ptr_q][WIDTH-2:(WIDTH/2)]};
```

圖 11、修改前指令傳遞之訊號

```
assign data0_out_o = {ram_q[rd_ptr_q][(WIDTH/2)-1:0]};  
assign data1_out_o = {ram_q[rd_ptr_q][WIDTH-1:(WIDTH/2)]};
```

圖 12、修改後指令傳遞之訊號

因為在指令傳遞下去會發生上述的錯誤，因此我們的解法是把他給成正確

值，就沒有問題了。

7. Result

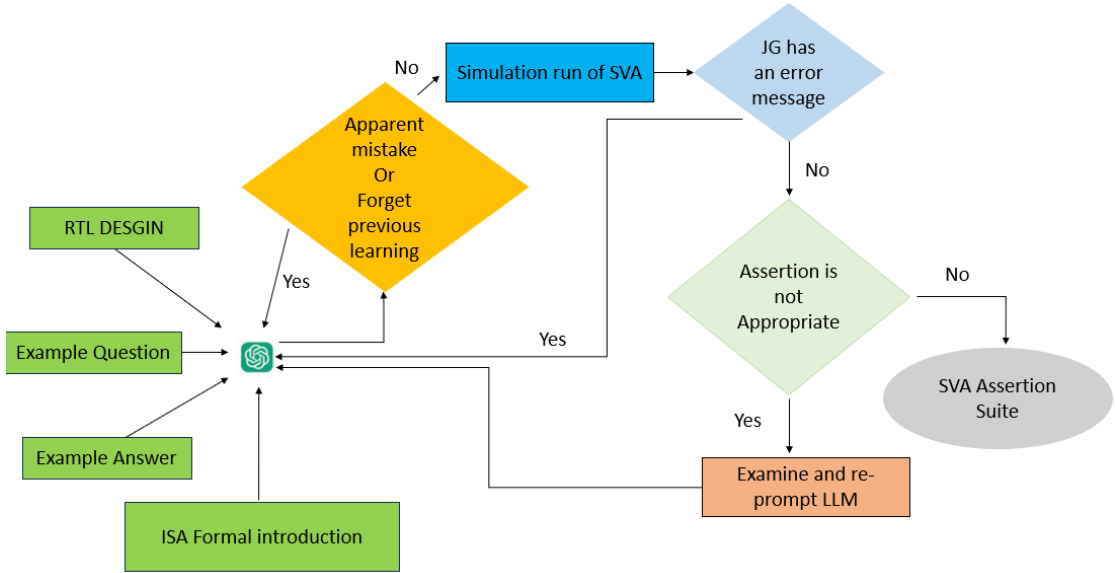


圖 13、Training an LLM to create SVA flowcharts

Prop	Assert	riscv_top.formaltest_biriscv.LW_insn0_check	Bm	12 -	0	279.5	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.LW_insn0_check:precondition1	Ht	8	1	0.3	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.LW_insn1_check	Bm	12 -	0	343.1	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.LW_insn1_check:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.AUIPC_insn0_check	B	12 -	0	9.2	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.AUIPC_insn0_check:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.AUIPC_insn1_check	Bm	12 -	0	381.8	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.AUIPC_insn1_check:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.SRAI_insn0_check	Ht	12 -	0	358.6	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.SRAI_insn0_check:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.SRAI_insn1_check	Ht	12 -	0	567.0	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.SRAI_insn1_check:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.JAL_PC_check0	K	15 -	0	43.1	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.JAL_PC_check0:precondition1	L	9 - 13	1	1.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.JAL_PC_check1	B	13 -	0	8.4	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.JAL_PC_check1:precondition1	B	9	1	0.9	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.JAL_rd_check0	B	12 -	0	7.9	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.JAL_rd_check0:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.JAL_rd_check1	Ht	12 -	0	571.1	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.JAL_rd_check1:precondition1	Ht	8	1	0.4	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.assert_bltu_pipe0_taken	Ht	12 -	0	805.1	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.assert_bltu_pipe0_taken:precondition1	L	10 - 31	1	79.9	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.assert_bltu_pipe0_not_taken	Ht	11 -	0	160.2	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.assert_bltu_pipe0_not_taken:precondition1	Bm	9	1	12.8	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.assert_bltu_pipe1_taken	Ht	11 -	0	179.6	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.assert_bltu_pipe1_taken:precondition1	L	10 - 324	1	61.3	<embedded>
✓	Assert	riscv_top.formaltest_biriscv.assert_bltu_pipe1_not_taken	Ht	11 -	0	211.8	<embedded>
✓	Cover (related)	riscv_top.formaltest_biriscv.assert_bltu_pipe1_not_taken:precondition1	Bm	9	1	12.9	<embedded>

圖 14、sva 驗證結果

經過本次 project2 的練習，我們整理出與 LLM 下 prompt 生成 SVA 的流程，一開始先輸入 RTL design, Example questions, Example answer，使其根據範例生成 SVA，接著觀察有無明顯錯誤進行修正，最後進行模擬，如有錯誤就將 JG 產生

的 error message 或波形回饋給 LLM 進行錯誤分析及更正。