Lab1-1 Report

葉承泓, 半導體研究學院碩士班(設計部), 112501538

✓ Due date: 2024/3/24 23:59

● Answers to the questions in the workbook

1. Show the code that you use to program configuration address [‘h3000_5000].

為了將 AXI 相關訊號都 MUX 到 user_project_1（FIR engine 所放之處）（原本 default 為 MUX 到 user_project_0，對應到題目已設計好的 Edge detect IP），我們需要將 32’ h3000_5000 這個 address program 成「1」，因此在 testbench（tb_fsic.v）中我參考原有的 testbench 中關於 SoC side 如何去 program value to some address 的方式，寫了底下的程式碼以達成「to program configuration address [‘h3000_5000]」的目的：

```
/////// Enable user_project_1 (FIR engine)  ///////
soc_change_UP_configuration_write(32'h01);
```

其中呼叫了 soc_change_UP_configuration_write()這個 task，其定義如下：

```
task soc_change_UP_configuration_write; // soc_change_user_project_configuration_write
    /////input [11:0] offset;       //4K range
    /////input [3:0] sel;
    input [31:0] data;

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= 32'h3000_5000;
        /////wbs_adr[11:2] <= offset[11:2]; //only provide DW address

        wbs_wdata <= data;
        /////wbs_sel <= sel;
        wbs_sel <= 4'b1111;
        wbs_cyc <= 1'b1;
        wbs_stb <= 1'b1;
        wbs_we <= 1'b1;

        @(posedge soc_coreclk);
        while(wbs_ack==0) begin
            @(posedge soc_coreclk);
        end

        $display($time, "=> soc_change_UP_configuration_write : wbs_adr=%x, wbs_sel=%b, wbs_wdata=%x", wbs_adr, wbs_sel, wbs_wdata);
    end
endtask
```

其中的 input argument 為輸入一個 32 bit 的 data，作為要寫入到 3000_5000 這個位置的值，由於是模擬 from SoC side 做 configuration write，也就是透過 CPU 去 write address（雖然實際上是由 Caravel SoC 的

CPU 去傳這些資料，但在此 lab 中只模擬並檢驗 FSIC 的行為是否正確，故在此 lab 中是透過 testbench 模擬給 WB 訊號，而非真正使用 CPU 去給），故為透過 WB interface，因此需要設定 address、write enable(WE)等訊號，由於此 task 是想要 write 3000_5000 這個 address 的值，故設定 address 為 3000_5000，且 WE 設定為 1。等待 WB_ACK 回傳後表示對應的 address 接收到 write request 了，因此表示完成 configuration write 的任務，理論上就會自動將 AXI 相關訊號都 MUX 到 user_project_1 了！

2. **Explain why "By programming configuration address [‘h3000_5000], signal user_prj_sel[4:0] will change accordingly" ?** (Hint: trace code in config_ctrl.v)

觀察/lab_1/fsic_fpga/rtl/user/config_ctrl/rtl/config_ctrl.v 裡的 code 後，可發現下列幾個部分的程式碼與 configuration address [‘h3000_5000]以及 user_prj_sel 這個訊號有關：當我們從 SoC side 送出 WB request 後（且此時沒有未處理完畢的 FPGA side 的 request，即此時沒有 f_axi_request），從下圖的程式碼中可得知會將 axi_grant_o_reg 的值改寫為 0。

```
////////////////////////////////////////////////
// Always for requests grant - axi_grant_o_reg //
////////////////////////////////////////////////
always @( posedge wb_clk or posedge wb_rst )
begin
    if ( wb_rst ) begin
        axi_grant_o_reg <= 1'b0;
    end else begin
        case (axi_grant_o_reg)
            1'b0: begin
                if ((~wb_axi_request)) begin
                    if (f_axi_request) begin
                        axi_grant_o_reg <= 1'b1;
                    end
                end
            end
            1'b1: begin
                if ((~f_axi_request)) begin
                    if (wb_axi_request) begin
                        axi_grant_o_reg <= 1'b0;
                    end
                end
            end
        endcase
    end
end
```

由於 axi_grant_o_reg 的值變為 0，故使得 m_axi_request_add 的值變為 wb_axi_request_add，如下圖所示。

```
////////////////////////////////////
// Assignment for Internal begin //
////////////////////////////////////
assign m_axi_request = axi_grant_o_reg ? f_axi_request : wb_axi_request;
assign m_axi_request_rw = axi_grant_o_reg ? f_axi_request_rw : wb_axi_request_rw;
assign m_axi_wstrb = axi_grant_o_reg ? f_axi_wstrb : wb_axi_wstrb;
assign m_axi_request_done = axi_grant_o_reg ? f_axi_request_done : wb_axi_request_done;
assign m_axi_request_add = axi_grant_o_reg ? f_axi_request_add : wb_axi_request_add;
assign m_axi_wdata = axi_grant_o_reg ? f_axi_wdata : wb_axi_wdata;
```

其中的 wb_axi_request_add 可由下方程式碼得知其行為：

```
////////////////////////////////////
// Always for Wishbone Interface handling //
////////////////////////////////////
always @ ( posedge wb_clk or posedge wb_rst)
begin
    if ( wb_rst )
    begin
        wb_fsm_reg <= wb_fsm_idle;
        wb_axi_request <= 1'b0;
        wb_axi_request_rw <= 1'b0;
        wb_axi_wstrb <= 4'b0;
        wb_axi_request_add <= 32'b0;
        wb_axi_wdata <= 32'b0;

        wbs_ack_o <= 1'b0;
        wbs_rdata_o <= 32'b0;
    end else
    begin
        case (wb_fsm_reg)
            wb_fsm_idle:
            begin
                wbs_ack_o <= 1'b0;
                wbs_rdata_o <= 32'h0;
                if ( !wbs_ack_o ) begin
                    if ( wbs_cyc && wbs_stb ) begin
                        wb_axi_request <= 1'b1;
                        wb_axi_request_rw <= wbs_we;
                        wb_axi_request_add <= wbs_adr;  //Latch wbs_adr
                        if ( wbs_we ) begin
                            wb_axi_wdata <= wbs_wdata;  //Latch wbs_wdata;
                            wb_axi_wstrb <= wbs_sel;
                        end
                        wb_fsm_reg <= wb_fsm_inprogress;
                    end
                end
            end
            wb_fsm_inprogress:
            begin
                if ( wb_axi_request_done )
                begin
                    wbs_ack_o <= 1'b1;
                    if ( !wb_axi_request_rw )
                        wbs_rdata_o <= wb_axi_rdata;    //Output wbs_rdata_o
                    else
                        wb_axi_wdata <= 32'h0;
                    wb_axi_request <= 1'b0;
                    wb_axi_request_add <= 32'b0;
                    wb_fsm_reg <= wb_fsm_idle;
                end
            end
        endcase
    end
end
```

當 WB interface 在當下沒有 transaction（即 wb_fsm_reg 這個 FSM 正處於 wb_fsm_idle 的狀態）時，若有 WE request（即當 wbs_cyc 和 wbs_stb 這兩個訊號同時為 1 時）則會將 wbs_addr 寫到 wb_axi_request_add 這個訊號線上。也就是說，由於我們在此要 program 3000_5000 的值，因此這個 WB request 會將 wb_axi_request_add 寫入 wbs_addr（在此即為 3000_5000）的值。

綜合以上的說明，可知當 WB request 來臨時，會將 wb_axi_request_add 的值寫為 3000_5000，並進一步變為 m_axi_request_add 的值。

接著由於此時 m_axi_request_add 的值滿足 m_axi_request_add[31:12] = 20’ h30005，因此由下圖可知 cc_enable 的值會變為 1。

```
////////////////////////////////
// Always for Target Selection //
////////////////////////////////
always @ ( posedge axi_clk or negedge axi_reset_n)
begin
    if ( !axi_reset_n )
    begin
        cc_aa_enable_o <= 1'b0;
        cc_as_enable_o <= 1'b0;
        cc_is_enable_o <= 1'b0;
        cc_la_enable_o <= 1'b0;
        cc_up_enable_o <= 1'b0;
        cc_enable <= 1'b0;
        cc_sub_enable <= 1'b0;
    end else
    begin
        cc_aa_enable_o <= ( m_axi_request_add[31:12] == 20'h30002 )? 1'b1 : 1'b0;
        cc_as_enable_o <= ( m_axi_request_add[31:12] == 20'h30004 )? 1'b1 : 1'b0;
        cc_is_enable_o <= ( m_axi_request_add[31:12] == 20'h30003 )? 1'b1 : 1'b0;
        cc_la_enable_o <= ( m_axi_request_add[31:12] == 20'h30001 )? 1'b1 : 1'b0;
        cc_up_enable_o <= ( m_axi_request_add[31:12] == 20'h30000 )? 1'b1 : 1'b0;
        cc_enable <= ( m_axi_request_add[31:12] == 20'h30005 )? 1'b1 : 1'b0;
        cc_sub_enable <= ( (m_axi_request_add[31:12] >= 20'h30006) && (m_axi_request_add[31:12] <= 20'h3FFFF ) )? 1'b1 : 1'b0;
    end
end
```

而經過下圖中的 assign 指令，又因為 cc_enable 訊號為 1，使得 cc_axi_awvalid 的值變為與 axi_awvalid 相同，且 cc_axi_wvalid 的值變為與 axi_wvalid 相同。由於此時我們透過 WB request 要執行 configuration write 3000_5000 這個 address 的值，故 axi_awvalid、axi_wvalid 皆應為 1，進而導致 cc_axi_awvalid、cc_axi_wvalid 的值也為 1, respectively。

```
assign cc_axi_awvalid = axi_awvalid && cc_enable;
assign cc_axi_wvalid = axi_wvalid && cc_enable;
```

接著由於 cc_axi_awvalid、cc_axi_wvalid 的值皆為 1，又因為 program 的位址是 3000_5000，使得此時的 axi_awaddr[11:0]為 12’ h000，故由下圖的程式碼可知會使得 user_prj_sel_o 的值變為 axi_wdata[4:0]，也就是我們所輸入的「1」。(由此部分程式碼也可看出在 reset phase 時，user_prj_sel_o 的 default 值設定為 0，也就是 default 使用的是 user_project_0，除非我們有在 reset 後特別去 program 3000_5000 位址的值，才會改變所 access 到

的 user project 編號，並且一旦改變後，就會因為 user_prj_sel_o<=user_prj_sel_o 這段程式碼而持續下去，不會立刻又變回 user_project_0）。

```
////////////////////////////////////////
// Always for AXI-Lite CC Slave response //
////////////////////////////////////////
always @ ( posedge axi_clk or negedge axi_reset_n )
begin
    if ( !axi_reset_n ) begin
        user_prj_sel_o <= 5'b0;
    end else begin
        if ( cc_axi_awvalid && cc_axi_wvalid ) begin
            if (axi_awaddr[11:0] == 12'h000 && (axi_wstrb[0] == 1) ) begin //offset 0
                user_prj_sel_o <= axi_wdata[4:0];
            end
            else begin
                user_prj_sel_o <= user_prj_sel_o;
            end
        end
    end
end
```

最後，由下圖程式碼可知 user_prj_sel 訊號與 user_prj_sel_o 是接線在一起的，故當 program address 3000_5000 的值為 1 時，user_prj_sel 訊號會變為 1，進而傳遞給其他 module 做 MUX select 使得相關的 AXI 訊號都 access 到 user_project_1。

```
assign user_prj_sel = user_prj_sel_o;
```

3. **Briefly describe how you do FIR initialization (tap parameter, length) from SoC side (Test#1).**

參考 tb_fsic.v 中原有的各 task 的寫法，我依據目標需求（program 並檢測 FIR engine）稍作修改得到如下的 testbench about **Test#1**：

(1)首先如同 workbook 中所述的要先做 reset：透過呼叫 test0_initialization() 這個 task 即可：

```
/////// SoC & FPGA reset ///////
test0_initialization();
```

其中 test0_initialization() 即如同 workbook 中的寫法，也如同參考程式碼中各 test00X()中的 reset 寫法：

```
task test0_initialization;
    begin
        $display("test0: initialization");

        #100;
        $display("SoC & FPGA reset");
        fork
            soc_apply_reset(40, 40);            //change coreclk phase in soc
            fpga_apply_reset(40, 40);       //fix coreclk phase in fpga
        join
        #40;
        fpga_as_to_is_init();
        //soc_cc_is_enable=1;
        fpga_cc_is_enable=1;
        fork
            soc_is_cfg_write(0, 4'b0001, 1);                //ioserdes rxen
            fpga_cfg_write(0,1,1,0);
        join
        $display($time, "=> soc rxen_ctl=1");
        $display($time, "=> fpga rxen_ctl=1");

        #400;
        fork
            soc_is_cfg_write(0, 4'b0001, 3);                //ioserdes txen
            fpga_cfg_write(0,3,1,0);
        join
        $display($time, "=> soc txen_ctl=1");
        $display($time, "=> fpga txen_ctl=1");

        #200;
        fpga_as_is_tdata = 32'h5a5a5a5a;
        #40;
        #200;
    end
endtask
```

主要是將 SoC 及 FPGA side 皆做 reset，並透過 configuration write 寫入 IS (io-serdes)使 SoC 與 FPGA side 的 Tx 和 Rx 皆開啟，才能開始與 FSIC 內部互相傳遞資料。

(2)由於經過 reset，故所選擇的 user project 編號回到 default 值 user_project_0，因此要再透過第 1. 題的回答中所述的 soc_change_UP_configuration_write(32'h01); 來將 user_prj_sel 訊號值重新 program 成 1。

```
/////// Enable user_project_1 (FIR engine)  ///////
/////soc_change_UP_configuration_write(0, 4'b1111, 32'h01);
soc_change_UP_configuration_write(32'h01);
```

(3)在開始 program tap parameter 以及 data_length 之前，要先檢查 FIR engine 是否正在運作，若仍正在運作，則要一直等到 FIR engine 回復到 IDLE 狀態才能開始 program。這個部分可透過從 SoC side 用 WB interface 來 read 0x3000_0000 位址(存放 ap 相關資訊，包括 ap_start、ap_done、ap_idle) 的值，其中回傳值的 bit 2 會是 ap_idle，因此可藉

由此 bit 的值來判斷 FIR engine 是否正在運作或是呈現 idle 狀態。若尚未 idle，則再不斷 read 0x3000_0000 直到 idle 為止。程式碼如下圖所示：

```
/////// Check FIR is idle, if not, wait until FIR is idle ///////
soc_up_cfg_read(12'd0, 4'b1111);
while (cfg_read_data_captured[2]==0) begin // which means "ap_idle_done_start[2]==0"
    soc_up_cfg_read(12'd0, 4'b1111);
end
```

其中 soc_up_cfg_read()這個 task 的定義如下：

```
task soc_up_cfg_read;
    input [11:0] offset;        //4K range
    input [3:0] sel;

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= UP_BASE;
        wbs_adr[11:2] <= offset[11:2];  //only provide DW address

        wbs_sel <= sel;
        wbs_cyc <= 1'b1;
        wbs_stb <= 1'b1;
        wbs_we <= 1'b0;

        @(posedge soc_coreclk);
        while(wbs_ack==0) begin
            @(posedge soc_coreclk);
        end

        $display($time, "=> soc_up_cfg_read : wbs_adr=%x, wbs_sel=%b", wbs_adr, wbs_sel);
        //#1;       //add delay to make sure cfg_read_data_captured get the correct data
        @(soc_cfg_read_event);
        $display($time, "=> soc_up_cfg_read : got soc_cfg_read_event");
    end
endtask
```

其中 UP 表示 user project，因此對應到 0x30000000 的位址，透過 offset 這個 input argument 可以決定要 program 到 user project 內部的哪個位址，例如 12’ h000 就代表 ap register，12’ h 010 就代表 data_length 等等。同樣因為是 from SoC side，故為透過 WB interface 送資料，而 address 即為 UP_BASE=0x30000000 再加上 offset，WE 設定為 0（因為要 read），一直等到傳回 ACK 後，還必須等到 soc_cfg_read_event 這個 event 發生，以確保 data 有成功被放入 cfg_read_data_captured 這個暫存用的訊號線中。soc_cfg_read_event 的來源如下，其實就是當 FIR 回傳 ACK 時，將附帶的資料 wbs_data 存到 cfg_read_data_captured，再送出 event 告知 data 已順利被存起來了：

```
initial begin       //get soc wishbone read data result.
    while (1) begin
        @(posedge soc_coreclk);
        if (wbs_ack==1 && wbs_we == 0) begin
            //$display($time, "=> get wishbone read data result be : cfg_read_data_captured =%x, wbs_rdata=%x", cfg_read_data_captured, wbs_rdata);
            cfg_read_data_captured = wbs_rdata ;        //use block assignment
            //$display($time, "=> get wishbone read data result af : cfg_read_data_captured =%x, wbs_rdata=%x", cfg_read_data_captured, wbs_rdata);
            #0 -> soc_cfg_read_event;
            $display($time, "=> soc wishbone read data result : send soc_cfg_read_event");
        end
    end
end
```

因此 FIR 回傳 ACK 後且 soc_cfg_read_event 發生後，cfg_read_data_captured 即為回傳的 data，也就是 ap register 所存的值，故可藉由第 2 bit 來判斷是否 idle，若 idle 則再重發 request，直到不 idle 後即可進入下一步。

(4) 確認 FIR engine 正在 idle 後，即可從 SoC side 輸入 data_length 及各 tap parameter 至 configuration address map 中的相對應 address（data_length 寫到 0x30000010，故 offset 填入 12’h10；tap parameters 寫到 0x30000020~0x30000048，故 offset 填入 12’h20~12’h48, respectively）：

```
/////// Program length, and tap parameters ///////
soc_up_cfg_write(12'h10, 4'b1111, DATA_LENGTH);
soc_up_cfg_write(12'h20, 4'b1111, 32'd0);
soc_up_cfg_write(12'h24, 4'b1111, -32'd10);
soc_up_cfg_write(12'h28, 4'b1111, -32'd9);
soc_up_cfg_write(12'h2C, 4'b1111, 32'd23);
soc_up_cfg_write(12'h30, 4'b1111, 32'd56);
soc_up_cfg_write(12'h34, 4'b1111, 32'd63);
soc_up_cfg_write(12'h38, 4'b1111, 32'd56);
soc_up_cfg_write(12'h3C, 4'b1111, 32'd23);
soc_up_cfg_write(12'h40, 4'b1111, -32'd9);
soc_up_cfg_write(12'h44, 4'b1111, -32'd10);
soc_up_cfg_write(12'h48, 4'b1111, 32'd0);
```

其中 soc_up_cfg_write()這個 task 的定義如下：

```
task soc_up_cfg_write;
    input [11:0] offset;        //4K range
    input [3:0] sel;
    input [31:0] data;

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= UP_BASE;
        wbs_adr[11:2] <= offset[11:2];  //only provide DW address

        wbs_wdata <= data;
        wbs_sel <= sel;
        wbs_cyc <= 1'b1;
        wbs_stb <= 1'b1;
        wbs_we <= 1'b1;

        @(posedge soc_coreclk);
        while(wbs_ack==0) begin
            @(posedge soc_coreclk);
        end

        $display($time, "=> soc_up_cfg_write : wbs_adr=%x, wbs_sel=%b, wbs_wdata=%x", wbs_adr, wbs_sel, wbs_wdata);
    end
endtask
```

此 task 的寫法與第 1.題中的 soc_change_UP_configuration_write()很類似，只是在此因為要 program 的位址是 user project (即 FIR) 內部的訊號，因此 address 應為 UP_BASE=0x30000000，並且加上 offset 這個 input argument 以便在 parent function 細部更改要 program 的位址 (可在 parent function 更動最後 3 bytes) 。同樣是等到接收到 FIR 索回傳的 ACK 後表示有成功接收到，即可完成此 task 的定義。

(5)在 program 寫入 data_length 及各 tap parameter 後，要檢查是否成功寫入，我們可透過 read 這些位址的值並與 golden values 比對得知是否有成功傳給 FIR engine，相關 testbench 程式碼如下：

```
/////// read-back and check ///////
soc_UP_configuration_read_and_check(12'h10, 4'b1111, DATA_LENGTH);
soc_UP_configuration_read_and_check(12'h20, 4'b1111, 32'd0);
soc_UP_configuration_read_and_check(12'h24, 4'b1111, -32'd10);
soc_UP_configuration_read_and_check(12'h28, 4'b1111, -32'd9);
soc_UP_configuration_read_and_check(12'h2C, 4'b1111, 32'd23);
soc_UP_configuration_read_and_check(12'h30, 4'b1111, 32'd56);
soc_UP_configuration_read_and_check(12'h34, 4'b1111, 32'd63);
soc_UP_configuration_read_and_check(12'h38, 4'b1111, 32'd56);
soc_UP_configuration_read_and_check(12'h3C, 4'b1111, 32'd23);
soc_UP_configuration_read_and_check(12'h40, 4'b1111, -32'd9);
soc_UP_configuration_read_and_check(12'h44, 4'b1111, -32'd10);
soc_UP_configuration_read_and_check(12'h48, 4'b1111, 32'd0);
```

其中 soc_UP_configuration_read_and_check()這個 task 的定義如下：

```
task soc_UP_configuration_read_and_check; // soc_change_user_project_configuration_read_and_check
    /// Please first make sure "cfg_read_data_expect_value" has been set correctly !!! <-- No need to do this because we have set "golden_value" as an input
    input [11:0] offset;        //4K range
    input [3:0] sel;
    input [31:0] golden_value;

    begin
        cfg_read_data_expect_value = golden_value;
        soc_up_cfg_read(offset, sel);

        check_cnt = check_cnt + 1;
        if (cfg_read_data_captured !== cfg_read_data_expect_value) begin
            $display($time, "=> SoC configuration check: [ERROR] cfg_read_data_expect_value=%x, cfg_read_data_captured=%x", cfg_read_data_expect_value, cfg_read_data_captured);
            error_cnt = error_cnt + 1;
        end
        else
            $display($time, "=> SoC configuration check: [PASS] cfg_read_data_expect_value=%x, cfg_read_data_captured=%x", cfg_read_data_expect_value, cfg_read_data_captured);
        $display("-----------------");
    end
endtask
```

在呼叫此 task 時要傳入的第 3 個 input argument 為 golden value，用來比對 FIR 的回傳值是否正確。在此 task 內部會呼叫第(3)小點中說明的 soc_up_cfg_read()函式，並將 offset 輸入進去，當 soc_up_cfg_read()執行完畢後，此時的 cfg_read_data_captured 即為回傳值（如第(3)小點的說明），因此將其與 golden value 比對，並將結果 print 在螢幕上。同時因為有進行與 golden value 的比對，因此將 check_cnt 加 1，若比對結果為兩者不同則表示 FIR 回傳的結果錯誤，此時將 error_cnt 的值加 1，以便最終 report 統計資料。

(6) 最後，當 data_length 及各 tap parameter 皆正確被 program 後，即可將 ap_start program 為 1，以告知 FIR engine 可開始接收 data_in 與 data_out 並開始計算。

```
/////// Program ap_start = 1 ///////
$display(" Start FIR engine");
soc_up_cfg_write(12'h0, 4'b1111, 32'd1);
```

而「將 ap_start program 為 1」是透過 soc_up_cfg_write()這個 task 去寫入 1 到 address 0x30000000 來達成。soc_up_cfg_write()在第(4)小點中有相關說明。

4. **Briefly describe how you do FIR initialization (tap parameter, length) from FPGA side (Test#2).**

參考 tb_fsic.v 中原有的各 task 的寫法，我依據目標需求（program 並檢測 FIR engine）稍作修改得到如下的 testbench about **Test#2**：

(1)如同第 3.題的流程，先做 reset：

```
/////// SoC & FPGA reset ///////
test0_initialization();
```

(2)如同第 3.題的流程，透過 soc_change_UP_configuration_write(32'h01); 來將 user_prj_sel 訊號值重新 program 成 1。

```
/////// Enable user_project_1 (FIR engine)  ///////
soc_change_UP_configuration_write(32'h01);
```

(3)參考原本 testbench 中的流程，發現此時應先確認 AA (AXIS-AXIL) 的 internal register 在 reset 後的 default 值是否正確。如下圖所示，首先將 fpga_as_is_tready 這個訊號寫為 1，表示可以接收回傳的訊號，接著去 read AA internal register 的位址的值，並與 golden value (default 值的 golden value 為 32’ h0) 比對，將結果 print 在螢幕上。

```
@ (posedge fpga_coreclk);
fpga_as_is_tready <= 1;

/// step 1. check default value
$display($time, "=> start checking SoC AA internal register default value (after reset), which should be 0");
cfg_read_data_expect_value =    32'h0;          //default value after reset = 0
soc_aa_cfg_read(AA_Internal_Reg_Offset, 4'b1111);

check_cnt = check_cnt + 1;
if (cfg_read_data_captured !== cfg_read_data_expect_value) begin
    $display($time, "=> [ERROR] cfg_read_data_expect_value=%x, cfg_read_data_captured=%x", cfg_read_data_expect_value, cfg_read_data_captured);
    error_cnt = error_cnt + 1;
end
else
    $display($time, "=> [PASS] cfg_read_data_expect_value=%x, cfg_read_data_captured=%x", cfg_read_data_expect_value, cfg_read_data_captured);
$display("-----------------");
```

其中會用到 soc_aa_cfg_read()這個 task，其寫法與第 3. (3)小點的 soc_up_cfg_read()非常類似，只是寫入的位址不同 (為 AA_BASE= 32'h3000_2000) ：

```
task soc_aa_cfg_read;
    input [11:0] offset;        //4K range
    input [3:0] sel;

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= AA_BASE;
        wbs_adr[11:2] <= offset[11:2];  //only provide DW address

        wbs_sel <= sel;
        wbs_cyc <= 1'b1;
        wbs_stb <= 1'b1;
        wbs_we <= 1'b0;

        @(posedge soc_coreclk);
        while(wbs_ack==0) begin
            @(posedge soc_coreclk);
        end
        $display($time, "=> soc_aa_cfg_read : wbs_adr=%x, wbs_sel=%b", wbs_adr, wbs_sel);
        //#1;       //add delay to make sure cfg_read_data_captured get the correct data
        @(soc_cfg_read_event);
        $display($time, "=> soc_aa_cfg_read : got soc_cfg_read_event");
    end
endtask
```

(4)如同第 3.(3)小點，在開始 program tap parameter 以及 data_length 之前，要先檢查 FIR engine 是否正在運作——不斷 read 0x3000_0000 直到 idle 為止。程式碼如下圖所示：

```
/// step 2. FPGA issues FPGA-to-SoC configuration read/write request to SoC
/////// Check FIR is idle, if not, wait until FIR is idle ///////
fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'd0); // or {4'b0000,FPGA_to_SOC_UP_BASE}
@(soc_to_fpga_axilite_read_cpl_event);      //wait for FPGA get the read completion
while (soc_to_fpga_axilite_read_cpl_captured[2]==0) begin // which means "ap_idle_done_start[2]==0"
    fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'd0);
    @(soc_to_fpga_axilite_read_cpl_event);      //wait for FPGA get the read completion
end
```

其中 fpga_axilite_read_req()這個 task 的定義如下：

```
task fpga_axilite_read_req;
    input [31:0] address;
    begin
        fpga_as_is_tdata <= address;    //for axilite read address req phase
        $strobe($time, "=> fpga_axilite_read_req in address req phase = %x - tvalid", fpga_as_is_tdata);
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
            fpga_as_is_tupsb <=  5'b00000;
        `endif
        fpga_as_is_tstrb <=  4'b0000;
        fpga_as_is_tkeep <=  4'b0000;
        fpga_as_is_tid <=  TID_DN_AA;       //target to Axis-Axilite
        fpga_as_is_tuser <=  TUSER_AXILITE_READ_REQ;        //for axilite read req
        fpga_as_is_tlast <=  1'b0;
        fpga_as_is_tvalid <= 1;

        @ (posedge fpga_coreclk);
        while (fpga_is_as_tready == 0) begin        // wait util fpga_is_as_tready == 1 then change data
            @ (posedge fpga_coreclk);
        end
        $display($time, "=> fpga_axilite_read_req in address req phase = %x - transfer", fpga_as_is_tdata);
        fpga_as_is_tvalid <= 0;

    end
endtask
```

這裡使用的傳輸 protocol 是類似 AXI-Stream，只是 tdata 的部分傳的是是 address（因為 FPGA side 只有 AXI-Stream 的傳輸方式，並無 AXI-Lite，因此若要傳 address 也必須透過 AXI-Stream 傳送；甚至若同時有 data 和 address 需要傳（例如 write request)，則要分兩次分別傳送 data 及 address，到 FSIC 內部會再轉換成 AXI-Stream 或 AXI-Lite 的 protocol 再進一步傳給 FIR engine）。當 fpga_is_as_tready 仍為 0 時表示對方仍無法接收，要等到 fpga_is_as_tready 為 1 才表示 handshake 完成，此時可將 fpga_as_is_tvalid 拉回 0 並 return。

直到 fpga_is_as 完成 handshake 後，還必須等到 soc_to_fpga_axilite_read_cpl_event（其中 cpl 表示”completion”）

這個 event 發生，以確保 data 有成功被放入 soc_to_fpga_axilite_read_cpl_captured 這個暫存用的訊號線中。soc_to_fpga_axilite_read_cpl_event 的來源如下圖所示，其實就是當 fpga_is_as 完成 handshake 時，將回傳的資料 fpga_is_as_tdata 存到 soc_to_fpga_axilite_read_cpl_captured，再送出 event 告知 data 已順利被存起來了：

```
initial begin       //get upstream soc_to_fpga_axilite_read_completion
    while (1) begin
        @(posedge fpga_coreclk);
        if (fpga_is_as_tvalid == 1 && fpga_is_as_tid == TID_UP_AA && fpga_is_as_tuser == TUSER_AXILITE_READ_CPL) begin
            $display($time, "=> get soc_to_fpga_axilite_read_cpl_captured be : soc_to_fpga_axilite_read_cpl_captured =%x, fpga_is_as_tdata=%x", soc_to_fpga_axilite_read_cpl_captured, fpga_is_as_tdata);
            soc_to_fpga_axilite_read_cpl_captured = fpga_is_as_tdata ;      //use block assignment
            $display($time, "=> get soc_to_fpga_axilite_read_cpl_captured af : soc_to_fpga_axilite_read_cpl_captured =%x, fpga_is_as_tdata=%x", soc_to_fpga_axilite_read_cpl_captured, fpga_is_as_tdata);
            #0 -> soc_to_fpga_axilite_read_cpl_event;
            $display($time, "=> soc_to_fpga_axilite_read_cpl_captured : send soc_to_fpga_axilite_read_cpl_event");
        end
    end
end
```

因此 fpga_is_as 完成 handshake 後且 soc_to_fpga_axilite_read_cpl_event 發生後，soc_to_fpga_axilite_read_cpl_captured 即為回傳的 data，也就是 ap register 所存的值，故可藉由第 2 bit 來判斷是否 idle，若 idle 則再重發 request，直到不 idle 後即可進入下一步。

(5) 如同第 3.(4)小點的步驟，確認 FIR engine 正在 idle 後，即可從 FPGA side 輸入 data_length 及各 tap parameter 至 configuration address map 中的相對應 address（data_length 寫到 0x30000010，故 offset 填入 12’h10；tap parameters 寫到 0x30000020~0x30000048，故 offset 填入 12’h20~12’h48, respectively）：

```
/////// Program length, and tap parameters ///////
FPGA_to_SoC_configuration_write(28'h10, DATA_LENGTH);
FPGA_to_SoC_configuration_write(28'h20, 32'd0);
FPGA_to_SoC_configuration_write(28'h24, -32'd10);
FPGA_to_SoC_configuration_write(28'h28, -32'd9);
FPGA_to_SoC_configuration_write(28'h2C, 32'd23);
FPGA_to_SoC_configuration_write(28'h30, 32'd56);
FPGA_to_SoC_configuration_write(28'h34, 32'd63);
FPGA_to_SoC_configuration_write(28'h38, 32'd56);
FPGA_to_SoC_configuration_write(28'h3C, 32'd23);
FPGA_to_SoC_configuration_write(28'h40, -32'd9);
FPGA_to_SoC_configuration_write(28'h44, -32'd10);
FPGA_to_SoC_configuration_write(28'h48, 32'd0);
```

其中 FPGA_to_SoC_configuration_write()這個 task 的定義如下：

```
//task FPGA_AXI_Lite_write_request;
task FPGA_to_SoC_configuration_write;
    input [27:0] offset;
    input [31:0] data;

    begin
        @ (posedge fpga_coreclk);

        /// FPGA issues FPGA-to-SoC configuration write request to SoC
        fpga_axilite_write_req(FPGA_to_SOC_UP_BASE + offset , 4'b0001, data);

        /// FPGA waits for write to soc
        repeat(100) @ (posedge soc_coreclk);    //TODO FPGA waits for write to soc
    end
endtask
```

此 task 內部是透過呼叫 fpga_axilite_write_req()這個 task 並輸入相對應的 address 以及 data 來達成。由於此 task 主要是用來從 FPGA side 去寫入 FSIC，故 address 為 FPGA_to_SOC_UP_BASE + offset，即為 28’ h00000000+ offset。而 fpga_axilite_write_req()這個 task 的定義如下：

```
    task fpga_axilite_write_req;
        input [27:0] address;
        input [3:0] BE;
        input [31:0] data;

        begin
            fpga_as_is_tdata[27:0] <= address;  //for axilite write address phase
            fpga_as_is_tdata[31:28] <= BE;
            $strobe($time, "=> fpga_axilite_write_req in address phase = %x - tvalid", fpga_as_is_tdata);
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                fpga_as_is_tupsb <=  5'b00000;
            `endif
            fpga_as_is_tstrb <=  4'b0000;
            fpga_as_is_tkeep <=  4'b0000;
            fpga_as_is_tid <=  TID_DN_AA;       //target to Axis-Axilite
            fpga_as_is_tuser <=  TUSER_AXILITE_WRITE;       //for axilite write req
            fpga_as_is_tlast <=  1'b0;
            fpga_as_is_tvalid <= 1;

            @ (posedge fpga_coreclk);
            while (fpga_is_as_tready == 0) begin        // wait util fpga_is_as_tready == 1 then change data
                @ (posedge fpga_coreclk);
            end
            $display($time, "=> fpga_axilite_write_req in address phase = %x - transfer", fpga_as_is_tdata);

            fpga_as_is_tdata <= data;   //for axilite write data phase
            $strobe($time, "=> fpga_axilite_write_req in data phase = %x - tvalid", fpga_as_is_tdata);
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                fpga_as_is_tupsb <=  5'b00000;
            `endif
            fpga_as_is_tstrb <=  4'b0000;
            fpga_as_is_tkeep <=  4'b0000;
            fpga_as_is_tid <=  TID_DN_AA;       //target to Axis-Axilite
            fpga_as_is_tuser <=  TUSER_AXILITE_WRITE;       //for axilite write req
            fpga_as_is_tlast <=  1'b1;      //tlast = 1
            fpga_as_is_tvalid <= 1;

            @ (posedge fpga_coreclk);
            while (fpga_is_as_tready == 0) begin        // wait util fpga_is_as_tready == 1 then change data
                @ (posedge fpga_coreclk);
            end
            $display($time, "=> fpga_axilite_write_req in data phase = %x - transfer", fpga_as_is_tdata);


            fpga_as_is_tvalid <= 0;

        end
    endtask
```

如同第(4)小點中所述，由於 FPGA side 只有 AXI-Stream 的 interface，故需要將 request 的 address 與 data 分開傳，再經由 FSIC 內部將其重

新整合成 AXI-Lite 的 request 送至 user project 中，故圖中可看出先傳 address，待達成 handshake 後，再傳 data。

(6)如同第 3.(5)小點的步驟，在 program 寫入 data_length 及各 tap parameter 後，要檢查是否成功寫入，我們可透過 read 這些位址的值並與 golden values 比對得知是否有成功傳給 FIR engine，相關 testbench 程式碼如下：

```
/////// read-back and check ///////
FPGA_to_SoC_configuration_read_and_check(32'h10, DATA_LENGTH);
FPGA_to_SoC_configuration_read_and_check(32'h20, 32'd0);
FPGA_to_SoC_configuration_read_and_check(32'h24, -32'd10);
FPGA_to_SoC_configuration_read_and_check(32'h28, -32'd9);
FPGA_to_SoC_configuration_read_and_check(32'h2C, 32'd23);
FPGA_to_SoC_configuration_read_and_check(32'h30, 32'd56);
FPGA_to_SoC_configuration_read_and_check(32'h34, 32'd63);
FPGA_to_SoC_configuration_read_and_check(32'h38, 32'd56);
FPGA_to_SoC_configuration_read_and_check(32'h3C, 32'd23);
FPGA_to_SoC_configuration_read_and_check(32'h40, -32'd9);
FPGA_to_SoC_configuration_read_and_check(32'h44, -32'd10);
FPGA_to_SoC_configuration_read_and_check(32'h48, 32'd0);
```

其中 FPGA_to_SoC_configuration_read_and_check()這個 task 的定義如下：

```
508     //task FPGA_AXI_Lite_read_request_and_check;
509     task FPGA_to_SoC_configuration_read_and_check;
510         input [31:0] offset;
511         input [31:0] golden_value;
512 
513         begin
514             @ (posedge fpga_coreclk);
515             /////fpga_as_is_tready <= 1;
516 
517             //step 1. FPGA issues configuration read request to SoC
518             soc_to_fpga_axilite_read_cpl_expect_value = golden_value;
519             fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + offset); //read address = h0000_3000 ~ h0000_3XXX for io serdes
520 
521             //step 2. FPGA wait for read completion from SoC
522             $display($time, "=> wait for soc_to_fpga_axilite_read_cpl_event");
523             @(soc_to_fpga_axilite_read_cpl_event);      //wait for FPGA get the read completion
524             $display($time, "=> got soc_to_fpga_axilite_read_cpl_event");
525             $display($time, "=> soc_to_fpga_axilite_read_cpl_captured=%x", soc_to_fpga_axilite_read_cpl_captured);
526 
527             //Data part
528                 check_cnt = check_cnt + 1;
529                 if ( soc_to_fpga_axilite_read_cpl_expect_value !== soc_to_fpga_axilite_read_cpl_captured) begin
530                     $display($time, "=> FPGA_to_SoC_configuration_read [ERROR] soc_to_fpga_axilite_read_cpl_expect_value=%x, soc_to_fpga_axilite_read_cpl_captured[27:0]=%x",
                        soc_to_fpga_axilite_read_cpl_expect_value, soc_to_fpga_axilite_read_cpl_captured[27:0]);
531                     error_cnt = error_cnt + 1;
532                 end
533                 else
534                     $display($time, "=> FPGA_to_SoC_configuration_read [PASS] soc_to_fpga_axilite_read_cpl_expect_value=%x, soc_to_fpga_axilite_read_cpl_captured[27:0]=%x",
                        soc_to_fpga_axilite_read_cpl_expect_value, soc_to_fpga_axilite_read_cpl_captured[27:0]);
535         end
536     endtask
```

在呼叫此 task 時要傳入的第 2 個 input argument 為 golden value，用來比對 FIR 的回傳值是否正確。在此 task 內部會呼叫第(4)小點中說明的 fpga_axilite_read_req()函式，並將 FPGA_to_SOC_UP_BASE+offset 這

個位址輸入進去，當 fpga_axilite_read_req() 執行完畢且 soc_to_fpga_axilite_read_cpl_event 這個 event 發生後，此時的 soc_to_fpga_axilite_read_cpl_captured 即為回傳值（如第(4)小點的說明），因此將其與 golden value 比對，並將結果 print 在螢幕上。同時因為有進行與 golden value 的比對，因此將 check_cnt 加 1，若比對結果為兩者不同則表示 FIR 回傳的結果錯誤，此時將 error_cnt 的值加 1，以便最終 report 統計資料。

(7) 最後，當 data_length 及各 tap parameter 皆正確被 program 後，即可將 ap_start program 為 1，以告知 FIR engine 可開始接收 data_in 與 data_out 並開始計算。

```
/////// Program ap_start = 1 ///////
$display(" Start FIR engine");
FPGA_to_SoC_configuration_write(28'h0, 32'd1);
```

而「將 ap_start program 為 1」是透過 FPGA_to_SoC_configuration_write()這個 task 去寫入 1 到 address 0x30000000 來達成。FPGA_to_SoC_configuration_write()在第(5)小點中有相關說明。

5. Briefly describe how you feed in X data from FPGA side.

首先透過呼叫 test1_fpga_axis_req()，以方便未來 maintain 有關 AXI-Stream 的程式碼：

```
soc_to_fpga_axis_expect_count = 0;
test1_fpga_axis_req();      //target to Axis Switch
```

如下圖所示為 test1_fpga_axis_req()這個 task 內部的行為，它主要是將 FIR 運算的 input 及 output 計算出來，並作為 golden value 存在相對應的 register “soc_to_fpga_axis_expect_value” 中。接著將 input data x[j]=j,

for j=0,1,2,…,63 依次輸入至 fpga_axis_req_modified() task 中，直到輸入完 64 筆 data_in。

```
reg [31:0] golden_output_data;
task test1_fpga_axis_req;
    begin
        $display("Call task test1_fpga_axis_req()");
        @ (posedge fpga_coreclk);
        fpga_as_is_tready <= 1;

        for(j=0; j<DATA_LENGTH; j=j+1)begin    //
            /////golden_output_data = 0*j+(-10)*(((j-1)<0)? 0:(j-1))+(-9)*(((j-2)<0)? 0:(j-2))+23*(((j-3)<0)? 0:(j-3))+56*(((j-4)<0)? 0:(j-4))+63*(((j-5)<0)? 0:(j-5))+56*(((j-6)<0)? 0:(j-6))+23*(((j-7)<0)? 0:(j-7))
            golden_output_data = 0*j+(-10)*(((j-1)<0)? 0:(j-1))+(-9)*(((j-2)<0)? 0:(j-2))+23*(((j-3)<0)? 0:(j-3))+56*(((j-4)<0)? 0:(j-4))+63*(((j-5)<0)? 0:(j-5))+56*(((j-6)<0)? 0:(j-6))+23*(((j-7)<0)? 0:(j-7))
            +(-9)*(((j-8)<0)? 0:(j-8))+(-10)*(((j-9)<0)? 0:(j-9))+0*(((j-10)<0)? 0:(j-10));
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                if (j==DATA_LENGTH-1) begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {j[4:0], 4'b0000, 4'b0000, 1'b1, golden_output_data};
                end
                else begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {j[4:0], 4'b0000, 4'b0000, 1'b0, golden_output_data};
                end
            `else
                if (j==DATA_LENGTH-1) begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {4'b0000, 4'b0000, 1'b1, golden_output_data};
                end
                else begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {4'b0000, 4'b0000, 1'b0, golden_output_data};
                end
            `endif
            soc_to_fpga_axis_expect_count <= soc_to_fpga_axis_expect_count+1;
            fpga_axis_req_modified(j, TID_DN_UP, 0);        //target to User Project
        end

        $display($time, "=> task test1_fpga_axis_req() done");
    end
endtask
```

在 fpga_axis_req_modified() 這個 task 內部，主要是將上層傳進來的 input argument “data” 從 FPGA side 透過 AXI-Stream interface 傳給 FSIC。其中的 tdata 即為上層所傳來的 input argument “data” ，而當最後一筆 data_in (即當 data= DATA_LENGTH-1) 時，會將 tlast 設為 1。當 data 準備好且其他設定皆完成後，就將 fpga_as_is_tvalid 拉成 1，直到對方(FSIC 的 IO serdes)的 fpga_is_as_tready 為 1 時表示 handshake 成功建立，數據成功傳輸，因此可再將 fpga_as_is_tvalid 拉回 0，即完成此次的 data_in 傳輸。

```
task fpga_axis_req_modified;
    input [31:0] data;
    input [1:0] tid;
    input mode; //0 ffor noram, 1 for random data
    reg [31:0] tdata;
    `ifdef USER_PROJECT_SIDEBAND_SUPPORT
        reg [`USER_PROJECT_SIDEBAND_WIDTH-1:0]tupsb;
    `endif
    reg [3:0] tstrb;
    reg [3:0] tkeep;
    reg tlast;

    begin
        if (mode) begin    //for random data
            tdata = $random;
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                tupsb = $random;
            `endif
            tstrb = $random;
            tkeep = $random;
            tlast = $random;
        end
        else begin
            tdata = data;
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                //tupsb = 5'b00000;
                tupsb = tdata[4:0];
            `endif
            tstrb = 4'b0000;
            tkeep = 4'b0000;
            if (data==DATA_LENGTH-1) begin
                tlast = 1'b1;
            end
            else begin
                tlast = 1'b0;
            end
        end
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
            fpga_as_is_tupsb <= tupsb;
        `endif
        fpga_as_is_tstrb <=  tstrb;
        fpga_as_is_tkeep <=  tkeep;
        fpga_as_is_tlast <=  tlast;
        fpga_as_is_tdata <= tdata;  //for axis write data
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
            $strobe($time, "-> fpga_axis_req send data, fpga_as_is_tupsb = %b, fpga_as_is_tstrb = %b, fpga_as_is_tkeep = %b, fpga_as_is_tlast = %b, fpga_as_is_tdata = %x", fpga_as_is_tupsb, fpga_as_is_tstrb,
            fpga_as_is_tkeep, fpga_as_is_tlast, fpga_as_is_tdata);
        `else
            $strobe($time, "-> fpga_axis_req send data, fpga_as_is_tstrb = %b, fpga_as_is_tkeep = %b, fpga_as_is_tlast = %b, fpga_as_is_tdata = %x", fpga_as_is_tstrb, fpga_as_is_tkeep, fpga_as_is_tlast,
            fpga_as_is_tdata);
        `endif

        fpga_as_is_tid <=  tid;    //set target
        fpga_as_is_tuser <=  TUSER_AXIS;       //for axis req
        fpga_as_is_tvalid <= 1;
        ///// Let's move the following part to task test1_fpga_axis_req() /////
        /////`ifdef USER_PROJECT_SIDEBAND_SUPPORT
        /////   soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tupsb, tstrb, tkeep, tlast, tdata};
        /////`else
        /////   soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tstrb, tkeep, tlast, tdata};
        /////`endif
        /////soc_to_fpga_axis_expect_count <= soc_to_fpga_axis_expect_count+1;

        @ (posedge fpga_coreclk);
        while (fpga_is_as_tready == 0) begin       // wait util fpga_is_as_tready == 1 then change data
            @ (posedge fpga_coreclk);
        end
        fpga_as_is_tvalid <= 0;

    end
endtask
```

6. **Briefly describe how you get output Y data in testbench, and how to do comparison with golden values.**

在 test1_fpga_axis_req()這個 task 中：

```
reg [31:0] golden_output_data;
task test1_fpga_axis_req;
    begin
        $display("Call task test1_fpga_axis_req()");
        @ (posedge fpga_coreclk);
        fpga_as_is_tready <= 1;

        for(j=0; j<DATA_LENGTH; j=j+1)begin     //
            /////golden_output_data = 0*j+(-10)*((j-1)<0)? 0:(j-1))+(-9)*(((j-2)<0)? 0:(j-2))+23*(((j-3)<0)? 0:(j-3))+56*(((j-4)<0)? 0:(j-4))+63*(((j-5)<0)? 0:(j-5))+56*(((j-6)<0)? 0:(j-6))+23*(((j-7)<0)? 0:(j-7))
            golden_output_data = 0*j+(-10)*(((j-1)<0)? 0:(j-1))+(-9)*(((j-2)<0)? 0:(j-2))+23*(((j-3)<0)? 0:(j-3))+56*(((j-4)<0)? 0:(j-4))+63*(((j-5)<0)? 0:(j-5))+56*(((j-6)<0)? 0:(j-6))+23*(((j-7)<0)? 0:(j-7))
            +(-9)*(((j-8)<0)? 0:(j-8))+(-10)*(((j-9)<0)? 0:(j-9))+0*(((j-10)<0)? 0:(j-10));
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                if (j==DATA_LENGTH-1) begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {j[4:0], 4'b0000, 4'b0000, 1'b1, golden_output_data};
                end
                else begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {j[4:0], 4'b0000, 4'b0000, 1'b0, golden_output_data};
                end
            `else
                if (j==DATA_LENGTH-1) begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {4'b0000, 4'b0000, 1'b1, golden_output_data};
                end
                else begin
                    soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {4'b0000, 4'b0000, 1'b0, golden_output_data};
                end
            `endif
            soc_to_fpga_axis_expect_count <= soc_to_fpga_axis_expect_count+1;
            fpga_axis_req_modified(j, TID_DN_UP, 0);        //target to User Project
        end

        $display($time, "=> task test1_fpga_axis_req() done");
    end
endtask
```

由於在這次 lab 中我們定義 data_in x[j]=j, with j=0,1,2,...,63，因此相對應的 data_out 可計算為

「y_golden[j]=0*j+(-10)*(((j-1)<0)? 0:(j-1))+(-9)*(((j-2)<0)? 0:(j-2))+23*(((j-3)<0)? 0:(j-3))+56*(((j-4)<0)? 0:(j-4))+63*(((j-5)<0)? 0:(j-5))+56*(((j-6)<0)? 0:(j-6))+23*(((j-7)<0)? 0:(j-7))+(-9)*(((j-8)<0)? 0:(j-8))+(-10)*(((j-9)<0)? 0:(j-9))+0*(((j-10)<0)? 0:(j-10));」。

將每筆 data_in 以及相對應的 golden data_out，還有一些相關訊號（fpga_as_is_tupsb、fpga_as_is_tstrb、fpga_as_is_tkeep、fpga_as_is_tlast）全部合併起來並存在 soc_to_fpga_axis_expect_value[j]這個 register 中，以便之後與 FIR 所計算出並回傳給 FPGA side 的 output data 做比對。

執行完 test1_fpga_axis_req()這個 task 後，代表 64 筆 data 的相關資訊都已被儲存在 soc_to_fpga_axis_expect_value[]陣列中。在 Stream-in data_in 的同時，FIR 也會吐出已計算好的 data_out 在 output AXI-stream 中，由於 input stream 與 output stream 是分開的 interface，所以兩者是可以同

時並行運作的，因此 feed in data_in 的同時，也可偵測是否有 data out 出現，這個部分是透過下方程式碼來進行偵測及記錄：

```
initial begin       //get upstream soc_to_fpga_axis - for loop back test
    soc_to_fpga_axis_captured_count = 0;
    soc_to_fpga_axis_event_triggered = 0;
    while (1) begin
        `else
            @(posedge fpga_coreclk);
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                if (fpga_is_as_tvalid == 1 && fpga_is_as_tid == TID_UP_UP && fpga_is_as_tuser == TUSER_AXIS) begin
                    $display($time, "-> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis_captured[%d] =%x, fpga_is_as_tupsb=%x, fpga_is_as_tstrb=%x, fpga_is_as_tkeep=%x ,
                    fpga_is_as_tlast=%x, fpga_is_as_tdata=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count], fpga_is_as_tupsb,
                    fpga_is_as_tstrb, fpga_is_as_tkeep , fpga_is_as_tlast, fpga_is_as_tdata);
                    soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tupsb, fpga_is_as_tstrb, fpga_is_as_tkeep , fpga_is_as_tlast, fpga_is_as_tdata} ;      //use block assignment
                    $display($time, "-> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis_captured[%d] =%x, fpga_is_as_tupsb=%x, fpga_is_as_tstrb=%x, fpga_is_as_tkeep=%x ,
                    fpga_is_as_tlast=%x, fpga_is_as_tdata=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count], fpga_is_as_tupsb,
                    fpga_is_as_tstrb, fpga_is_as_tkeep , fpga_is_as_tlast, fpga_is_as_tdata);
                    soc_to_fpga_axis_captured_count = soc_to_fpga_axis_captured_count+1;
                end
                if ( (soc_to_fpga_axis_captured_count == DATA_LENGTH) && !soc_to_fpga_axis_event_triggered) begin
                    $display($time, "-> soc_to_fpga_axis_captured : send soc_to_fpga_axiis_event");
                    #0 -> soc_to_fpga_axis_event;
                    soc_to_fpga_axis_event_triggered = 1;
                end
            `else
                if (fpga_is_as_tvalid == 1 && fpga_is_as_tid == TID_UP_UP && fpga_is_as_tuser == TUSER_AXIS) begin
                    $display($time, "-> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis_captured[%d] =%x, fpga_is_as_tstrb=%x, fpga_is_as_tkeep=%x , fpga_is_as_tlast=%x,
                    fpga_is_as_tdata=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count], fpga_is_as_tstrb, fpga_is_as_tkeep ,
                    fpga_is_as_tlast, fpga_is_as_tdata);
                    soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tstrb, fpga_is_as_tkeep , fpga_is_as_tlast, fpga_is_as_tdata} ;       //use block assignment
                    $display($time, "-> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis_captured[%d] =%x, fpga_is_as_tstrb=%x, fpga_is_as_tkeep=%x , fpga_is_as_tlast=%x,
                    fpga_is_as_tdata=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count], fpga_is_as_tstrb, fpga_is_as_tkeep ,
                    fpga_is_as_tlast, fpga_is_as_tdata);
                    soc_to_fpga_axis_captured_count = soc_to_fpga_axis_captured_count+1;
                end
                if ( (soc_to_fpga_axis_captured_count == DATA_LENGTH) && !soc_to_fpga_axis_event_triggered) begin
                    $display($time, "-> soc_to_fpga_axis_captured : send soc_to_fpga_axiis_event");
                    #0 -> soc_to_fpga_axis_event;
                    soc_to_fpga_axis_event_triggered = 1;
                end
            `endif


            if (soc_to_fpga_axis_captured_count != DATA_LENGTH)
                soc_to_fpga_axis_event_triggered = 0;
        `endif

    end
end
```

(註：此 initial block 中有裁去`ifdef USE_EDGEDETECT_IP 的那個部份，因為我並無 define USE_EDGEDETECT_IP，故不會執行到那個部分的程式碼)
此部分程式碼主要是偵測當 fpga_is_as_tvalid 為 1 時，也就是當 FSIC 欲吐出 output data 給 FPGA side 時，會將吐出來的資訊(含 fpga_is_as_tupsb、fpga_is_as_tstrb、fpga_is_as_tkeep、fpga_is_as_tlast，以及 output data) 收集起來並存到 soc_to_fpga_axis_captured[] 這個 register/array 中，並將 soc_to_fpga_axis_captured_count 的值增加 1，表示多收到一筆 output data。當 soc_to_fpga_axis_captured_count 的值為 DATA_LENGTH(在此 lab 中為 64) 時，表示已接收到所有 data，因此 trigger soc_to_fpga_axis_event 表示已完成 output data 的接收及儲存。

當執行完 test1_fpga_axis_req()的 task 後(表示已將 64 筆 golden data 寫在 soc_to_fpga_axis_expect_value[]中)，並發生 soc_to_fpga_axis_event 後 (表示已成功接收並儲存 64 筆 FIR 運算所得到的結果在 soc_to_fpga_axis_captured[]中)，即可開始進行兩筆資料的比對：首先先比對是否 soc_to_fpga_axis_expect_count 為 64，若不為 64 則表示 golden data 在儲存的過程中出現錯誤；而 soc_to_fpga_axis_captured_count 並不

需要在此測試，因為若不到 64 則就不會 trigger soc_to_fpga_axis_event 了；接下來依序比對 64 筆 output data 的正確性，並將結果 report 在螢幕上，以及記錄 error count 即可。

```
test1_fpga_axis_req();      //target to Axis Switch
$display($time, "=> wait for soc_to_fpga_axis_event");
@(soc_to_fpga_axis_event);
$display($time, "=> soc_to_fpga_axis_expect_count = %d", soc_to_fpga_axis_expect_count);
$display($time, "=> soc_to_fpga_axis_captured_count = %d", soc_to_fpga_axis_captured_count);

check_cnt = check_cnt + 1;
if ( soc_to_fpga_axis_expect_count != DATA_LENGTH) begin
    $display($time, "=> [ERROR] soc_to_fpga_axis_expect_count = %d, soc_to_fpga_axis_captured_count = %d", soc_to_fpga_axis_expect_count, soc_to_fpga_axis_captured_count);
    error_cnt = error_cnt + 1;
end
else
    $display($time, "=> [PASS] soc_to_fpga_axis_expect_count = %d, soc_to_fpga_axis_captured_count = %d", soc_to_fpga_axis_expect_count, soc_to_fpga_axis_captured_count);


for(j=0; j<DATA_LENGTH; j=j+1)begin
    check_cnt = check_cnt + 1;
    if (soc_to_fpga_axis_expect_value[j] != soc_to_fpga_axis_captured[j] ) begin
        $display($time, "=> [ERROR] index(j)=%d, soc_to_fpga_axis_expect_value[%d] = %x, soc_to_fpga_axis_captured[%d]  = %x", j, j, soc_to_fpga_axis_expect_value[j], j, soc_to_fpga_axis_captured[j]);
        error_cnt = error_cnt + 1;
    end
    else
        $display($time, "=> [PASS] index(j)=%d, soc_to_fpga_axis_expect_value[%d] = %x, soc_to_fpga_axis_captured[%d]  = %x", j, j, soc_to_fpga_axis_expect_value[j], j, soc_to_fpga_axis_captured[j]);

end
//soc_to_fpga_axis_captured_count = 0;      //reset soc_to_fpga_axis_captured_count
#200;
$display("------Finish the data input(AXI-Stream ss) & data output(AXI-Stream sm) with checking------");
```

最後再 check 是否 FIR engine 有到 DONE 的狀態(ap_start=0、ap_done=1、ap_idle=1)，即完成從 FPGA side 輸入 data_in/輸出 data_out 至 FPGA side 的任務：

```
/////// check ap_done = 1 && ap_idle = 1 ///////
soc_UP_configuration_read_and_check(12'd0, 4'b1111, 32'b110);
```

註：在/lab_1/fsic_fpga/rtl/user/user_subsys/user_prj/user_prj1/rtl/user_prj1.v 中，由於每個 input data 到 output data 輸出的過程會需要運算時間而需要約 14 個 cycle 的 delay，當 output 時會因為已經轉換為下一筆 input data 而有不同的 tupsb、tkeep、tstrb 訊號，故為了能回傳在 input 當下的 tupsb、tkeep、tstrb 資訊，因此我透過下圖的方式將 input 當下的這些資訊先 buffer 起來，待 stream out (sm)輸出時再將之前 input 時相對應的 ss 訊號傳回去，才能使得 soc_to_fpga_axis_captured[] 中的相關資訊與 soc_to_fpga_axis_expect_value[]完全相同，以避免出現 error。

```
`ifdef USER_PROJECT_SIDEBAND_SUPPORT
  reg [pUSER_PROJECT_SIDEBAND_WIDTH-1: 0] sm_tupsb_reg;
`endif
reg [3:0] sm_tstrb_reg;
reg [3:0] sm_tkeep_reg;

`ifdef USER_PROJECT_SIDEBAND_SUPPORT
  assign sm_tupsb = sm_tupsb_reg;
`endif
assign sm_tstrb = sm_tstrb_reg;
assign sm_tkeep = sm_tkeep_reg;

always@(posedge axis_clk or negedge axis_rst_n) begin
    if(~axis_rst_n) begin
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
          sm_tupsb_reg <= 5'b0;
        `endif
        sm_tstrb_reg <= 4'b0;
        sm_tkeep_reg <= 0;
    end
    else if(ss_tvalid & ss_tready) begin
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
          sm_tupsb_reg <= ss_tupsb;
        `endif
        sm_tstrb_reg <= ss_tstrb;
        sm_tkeep_reg <= ss_tkeep;
    end
    else begin
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
          sm_tupsb_reg <= sm_tupsb_reg;
        `endif
        sm_tstrb_reg <= sm_tstrb_reg;
        sm_tkeep_reg <= sm_tkeep_reg;
    end
end
```

7. Screenshot simulation results printed on screen, to show that your Test#1 & Test#2 complete successfully

在順利完成 Test#1 後，會執行下圖的$display，以在螢幕上告知我們執行成功與否：

```
/////// All pass!!! ///////
if (error_cnt != 0 ) begin
    $display($time, "=> [Test 1 FAILED], check_cnt = %04d, error_cnt = %04d, please search [ERROR] in the log", check_cnt, error_cnt);
end
else begin
    $display("------------------------------------------------------");
    $display("------ [Success] Congratulations! Pass test 1 !! ------");
    $display("------------------------------------------------------");
end
```

在順利完成 Test#2 後，亦會執行下圖的$display：

```
/////// All pass!!! ///////
if (error_cnt != 0 ) begin
    $display($time, "=> [Test 2 FAILED], check_cnt = %04d, error_cnt = %04d, please search [ERROR] in the log", check_cnt, error_cnt);
end
else begin
    $display("------------------------------------------------------");
    $display("------ [Success] Congratulations! Pass test 2 !! ------");
    $display("------------------------------------------------------");
end
```

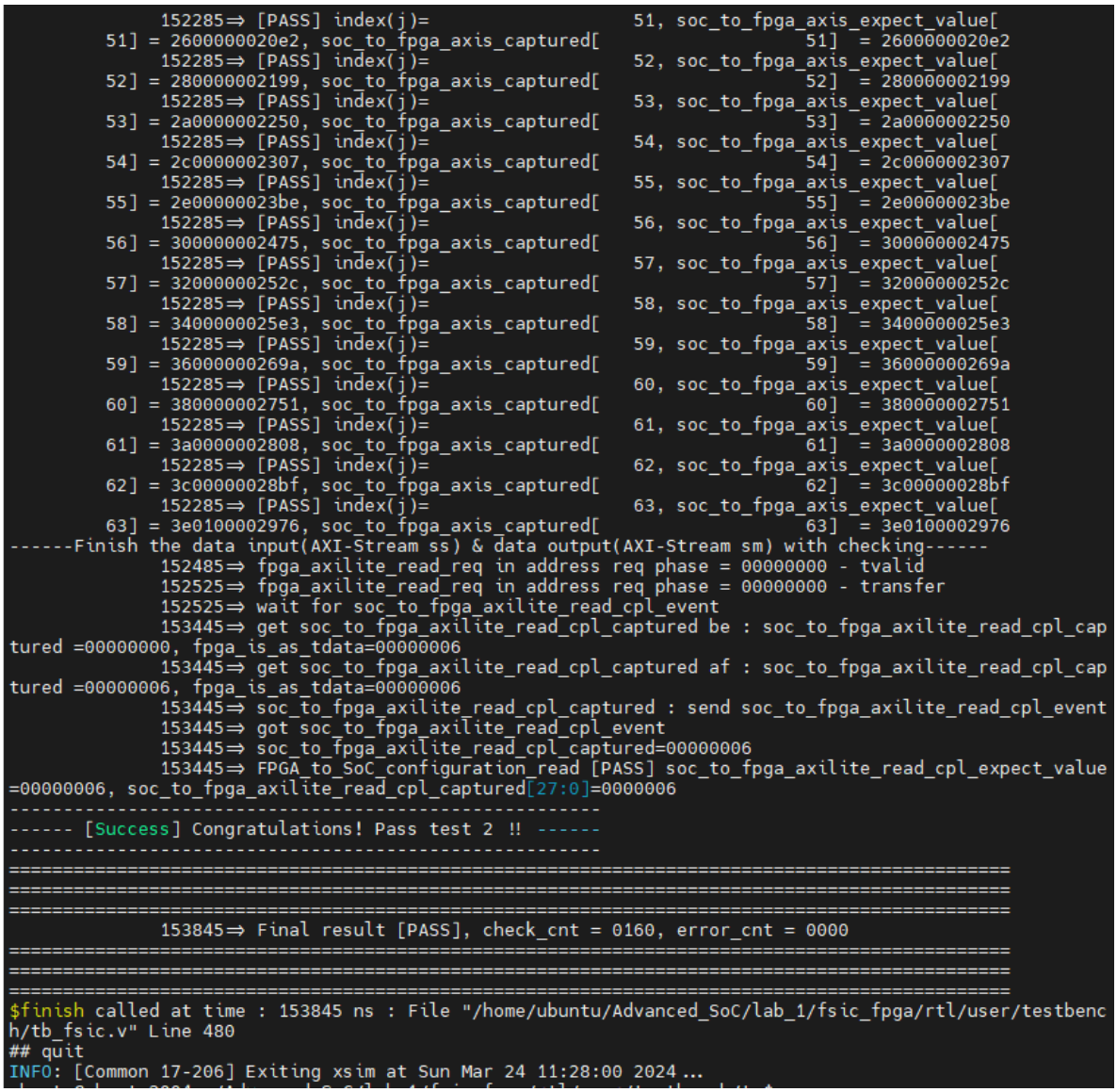
在執行完 test1_initialization_from_SoC_side() 與 test2_initialization_from_FPGA_side() 兩個 Test# 後，會執行下圖的$display：

```
$display("================================================================================================");
$display("================================================================================================");
$display("================================================================================================");
if (error_cnt != 0 )  begin
    $display($time, "=> Final result [FAILED], check_cnt = %04d, error_cnt = %04d, please search [ERROR] in the log", check_cnt, error_cnt);
    ////////////////// Added by me ////////////////////
    $display("(test 1 check count = %04d; test 2 check count = %04d)", check_count_test1, check_count_test2);
    $display("(test 1 error count = %04d; test 2 error count = %04d)", error_count_test1, error_count_test2);
    ////////////////////////////////////////////////
end
else
    $display($time, "=> Final result [PASS], check_cnt = %04d, error_cnt = %04d", check_cnt, error_cnt);
$display("================================================================================================");
$display("================================================================================================");
$display("================================================================================================");

$finish;
```

利用 " ./run_xsim " 執行模擬後，可在螢幕上得到下圖(由於 print 出的資訊實在太多，故只截取其中一部份) ：

```
         47] = 1e0000001e06, soc_to_fpga_axis_captured[                    47]  = 1e0000001e06
               46245⇒ [PASS] index(j)=                    48, soc_to_fpga_axis_expect_value[
         48] = 200000001ebd, soc_to_fpga_axis_captured[                    48]  = 200000001ebd
               46245⇒ [PASS] index(j)=                    49, soc_to_fpga_axis_expect_value[
         49] = 220000001f74, soc_to_fpga_axis_captured[                    49]  = 220000001f74
               46245⇒ [PASS] index(j)=                    50, soc_to_fpga_axis_expect_value[
         50] = 24000000202b, soc_to_fpga_axis_captured[                    50]  = 24000000202b
               46245⇒ [PASS] index(j)=                    51, soc_to_fpga_axis_expect_value[
         51] = 2600000020e2, soc_to_fpga_axis_captured[                    51]  = 2600000020e2
               46245⇒ [PASS] index(j)=                    52, soc_to_fpga_axis_expect_value[
         52] = 280000002199, soc_to_fpga_axis_captured[                    52]  = 280000002199
               46245⇒ [PASS] index(j)=                    53, soc_to_fpga_axis_expect_value[
         53] = 2a0000002250, soc_to_fpga_axis_captured[                    53]  = 2a0000002250
               46245⇒ [PASS] index(j)=                    54, soc_to_fpga_axis_expect_value[
         54] = 2c0000002307, soc_to_fpga_axis_captured[                    54]  = 2c0000002307
               46245⇒ [PASS] index(j)=                    55, soc_to_fpga_axis_expect_value[
         55] = 2e00000023be, soc_to_fpga_axis_captured[                    55]  = 2e00000023be
               46245⇒ [PASS] index(j)=                    56, soc_to_fpga_axis_expect_value[
         56] = 300000002475, soc_to_fpga_axis_captured[                    56]  = 300000002475
               46245⇒ [PASS] index(j)=                    57, soc_to_fpga_axis_expect_value[
         57] = 32000000252c, soc_to_fpga_axis_captured[                    57]  = 32000000252c
               46245⇒ [PASS] index(j)=                    58, soc_to_fpga_axis_expect_value[
         58] = 3400000025e3, soc_to_fpga_axis_captured[                    58]  = 3400000025e3
               46245⇒ [PASS] index(j)=                    59, soc_to_fpga_axis_expect_value[
         59] = 36000000269a, soc_to_fpga_axis_captured[                    59]  = 36000000269a
               46245⇒ [PASS] index(j)=                    60, soc_to_fpga_axis_expect_value[
         60] = 380000002751, soc_to_fpga_axis_captured[                    60]  = 380000002751
               46245⇒ [PASS] index(j)=                    61, soc_to_fpga_axis_expect_value[
         61] = 3a0000002808, soc_to_fpga_axis_captured[                    61]  = 3a0000002808
               46245⇒ [PASS] index(j)=                    62, soc_to_fpga_axis_expect_value[
         62] = 3c00000028bf, soc_to_fpga_axis_captured[                    62]  = 3c00000028bf
               46245⇒ [PASS] index(j)=                    63, soc_to_fpga_axis_expect_value[
         63] = 3e0100002976, soc_to_fpga_axis_captured[                    63]  = 3e0100002976
------Finish the data input(AXI-Stream ss) & data output(AXI-Stream sm) with checking------
               46645⇒ soc_up_cfg_read : wbs_adr=30000000, wbs_sel=1111
               46645⇒ soc wishbone read data result : send soc_cfg_read_event
               46645⇒ soc_up_cfg_read : got soc_cfg_read_event
               46645⇒ SoC configuration check: [PASS] cfg_read_data_expect_value=00000006, cfg_read_d
ata_captured=00000006
-----------------
------------------------------------------------------
------ [Success] Congratulations! Pass test 1 !! ------
------------------------------------------------------

++++++++++ Test 2: Initialize FIR from FPGA side ++++++++++
test0: initialization
SoC & FPGA reset
               46785⇒ soc POR Assert
               46785⇒ fpga POR Assert
               46785⇒ fpga reset Assert
               46825⇒ soc POR De-Assert
               46825⇒ fpga POR De-Assert
               46865⇒ fpga reset De-Assert
```

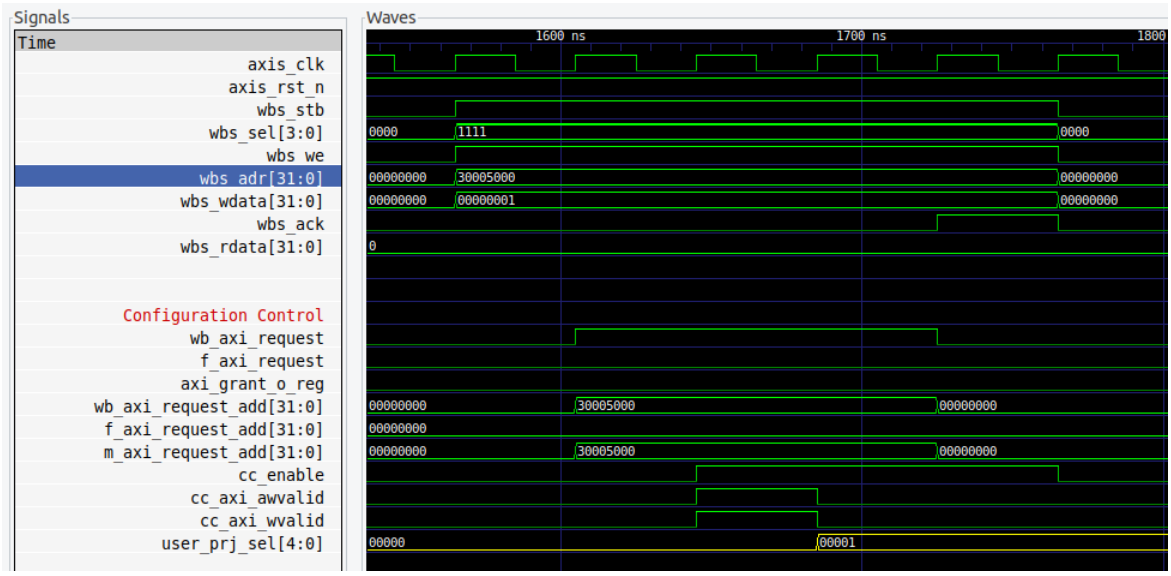
由此部分可知 **Test#1 成功，並無 error**！



由此部分可知 **Test#2 成功，並無 error**！且最終統計結果為總共執行了 160 次的與 golden value 的比對，而**錯誤數目為 0 次**！

8. Screenshot simulation waveform:

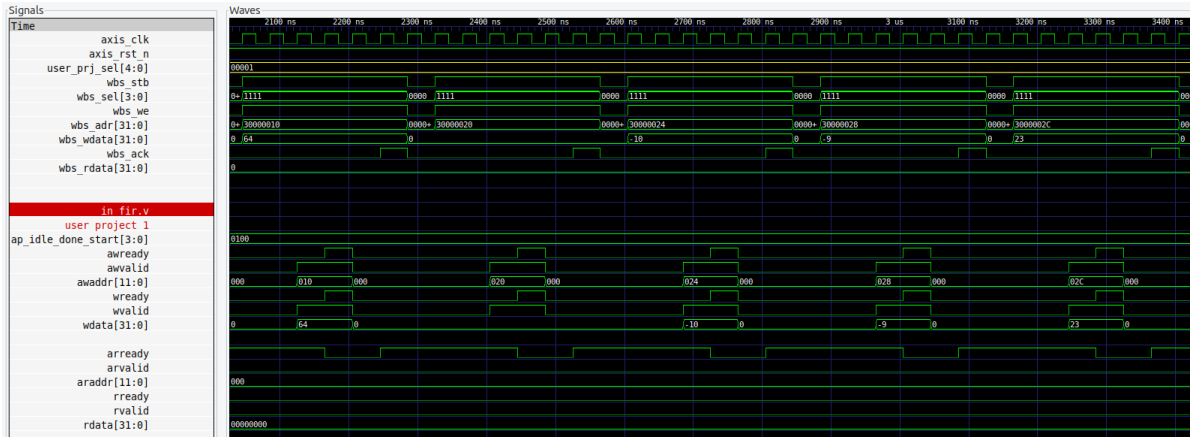
(1) Configuration cycle (when we program [‘h3000_5000] = 32’h01, signal user_prj_sel changes accordingly)



由圖中可看出當 WB 傳入 1 要寫到 address 為 0x30005000 時，第 1.題中所述的相對應訊號會改變，並且最終使得 user_prj_sel 的值變為 1。

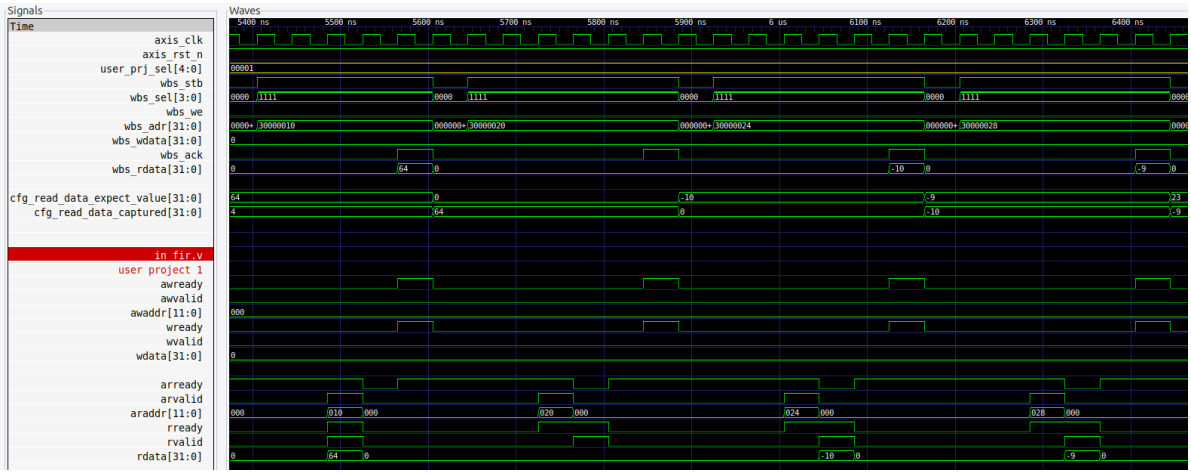
(2) AXI-Lite transaction cycles (feed in tap parameters, data_length)

(AXI-Lite write)



如上圖所示，可看出 fir.v 有成功接收到 AXI-Lite 的 write 訊號並成功達成 handshake。

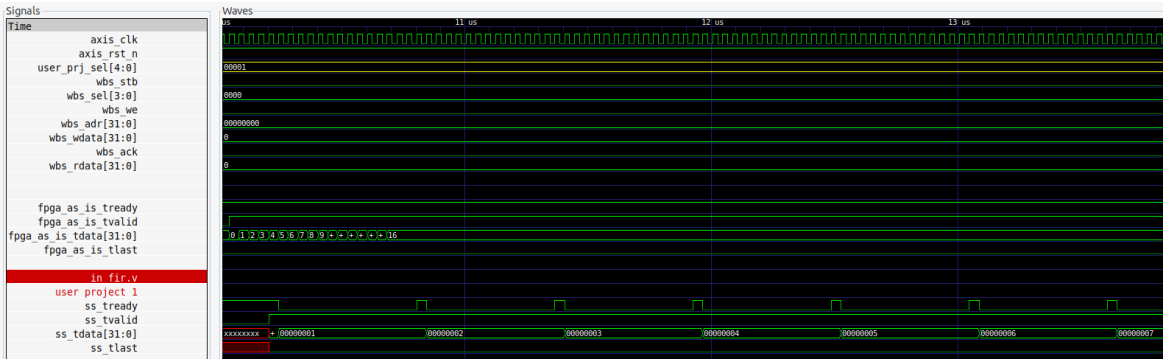
(AXI-Lite read)



如上圖所示，可看出 fir.v 有成功接收到 AXI-Lite 的 read 訊號並成功達成 handshake。當 capture 的值與 expected 的值相同時，會馬上進入下一個 read，因此圖中 expected 的值會立刻發生改變。

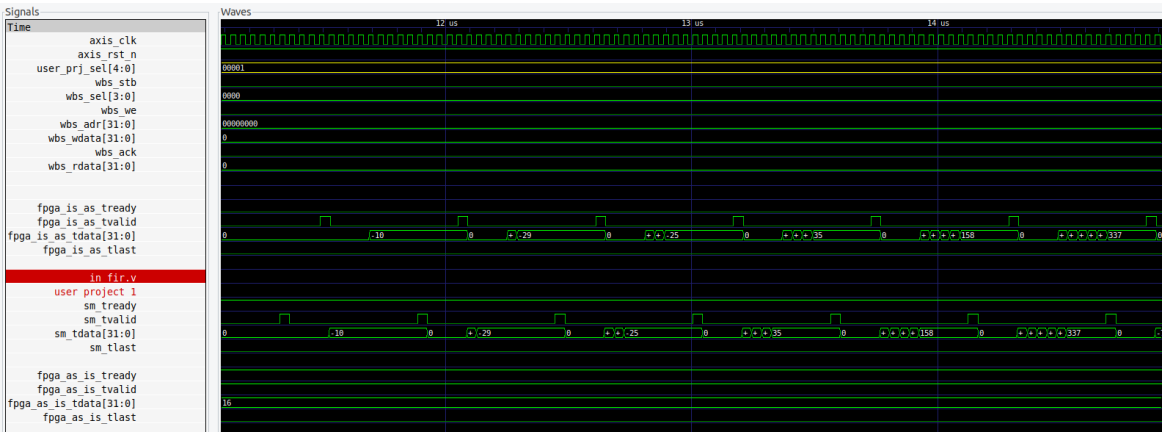
(3) Stream-in, Stream-out

(stream-in)



由圖中可看出有成功透過 FPGA side 將 input data 從 AS 傳送到 FSIC 的 IO，而使 fir.v 的 ss 收到 input 訊號。

(stream-out)



由圖中可看出有成功透過 FPGA side 將 output data 從 FSIC 的 IO 傳送到 FPGA 的 AS，而使 fir.v 的 sm 送出 output 訊號。

9. **(optional) Debug experience (bug found, and how to fix it)**
   寫在 https://hackmd.io/@whywhytellmewhy/r15D2Ao56 中，主要是關於 MUX 的部分。

- **Github link for my work about this lab**
  https://github.com/whywhytellmewhy/Advanced-SoC-design
  在上述 Github 連結中的 README.md 有更多關於其中的檔案的相關說明。