Advance SoC

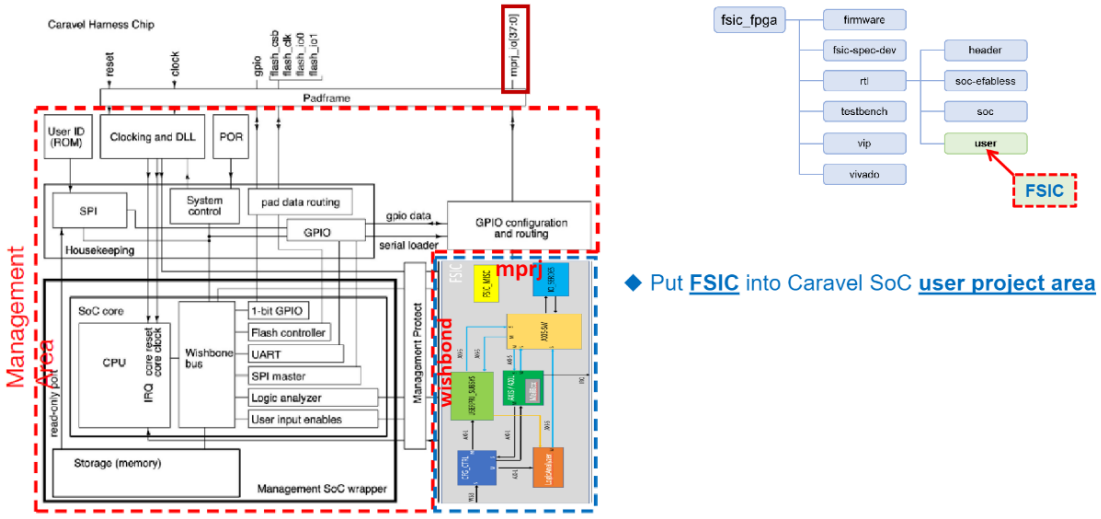
Lab 1 – FSIC simulation

110590022 陳冠晰

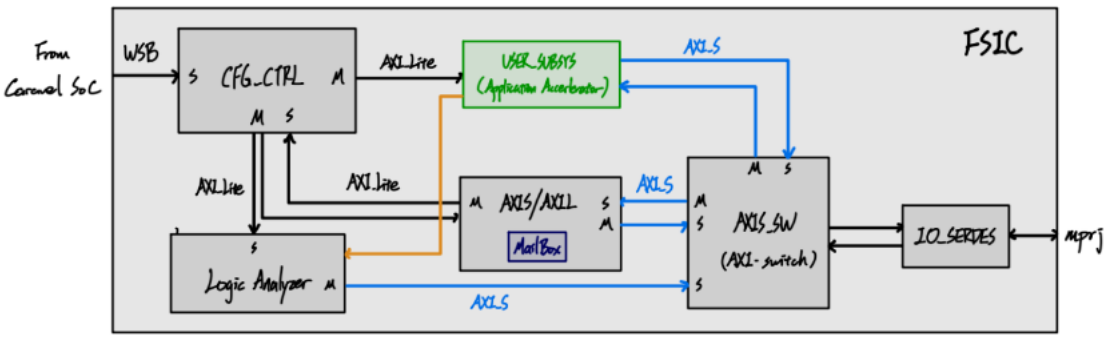
GitHub Link: https://github.com/vic9112/Advance_SOC

Brief Introduction about FSIC

- FSIC is an architecture to implement an IC validation system based on CaravelSOC



Block Diagram of FSIC



I also wrote a Markdown about the architecture of FSIC:

Link: https://hackmd.io/@vic9112/HykJ2Lpt6

Lab Questions

1. **Show the code that used to program configuration address 32'h3000_5000.**

   First, we need to write a task which can configurate data from SOC side:

   In tb_fsic.v: https://github.com/vic9112/Advance_SOC/blob/main/lab01%20-%20fsic-sim/fsic_fpga/rtl/user/testbench/tb_fsic.v

```
soc_cfg_write(32'h3000_5000, 4'b0001, 1); // Enable UserProj1
```

   Task – SOC configuration write:

```
/*========== SOC side configuration write - using WB ===========*/
// Refer to soc_up_cfg_write
task soc_cfg_write;
    //input  [3:0] target; // 4-bit for AA, IS, CC, register range
    input [31:0] adr;  // 4K range
    input  [3:0] sel;  // byte enable;
    input [31:0] data;
    begin
        @(posedge soc_coreclk);
        // Wishbone Cycle
        wbs_adr   <= adr;  // write address
        wbs_wdata <= data; // write data
        wbs_sel   <= sel;
        wbs_cyc   <= 1'b1;
        wbs_stb   <= 1'b1;
        wbs_we    <= 1'b1;
        // Stall if haven't receive ACK
        @(posedge soc_coreclk);
        while (wbs_ack == 0) @(posedge soc_coreclk);

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

2. **Explain why programming configuration address 32'h3000_5000, signal user_prj_sel[4:0] will change accordingly.**

   By the specified configuration control group:

RegisterName	Offset Address	Description
User Project Selction Control	12'h000	User Project Selection Control Register Definition This 5bits register is used for User Project selection. The selection mapping is defined as following: [4:0] 5'h0: All disable (Defalut) 5'h1: User Project 0 enabled 5'h2: User Project 1 enabled 5'h2: User Project 2 enabled ... 5'h1F: User Project 30 enabled [31:5] 27'hxxxxxxx: Reserved
Reserved	12'h004 ~ 12'hFFC	Reserved

We can know that programming configuration address 32'h3000_5000 will program the "User Project Selection Control."

And then inside **config_ctrl.v**:

```
////////////////////////////////
// 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
```

Address[31:12] == 20'h3000_5 will trigger "cc_enable", further generate an AXI transaction in the following code:

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

//////////////////////////////////////////
// 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
                                user_prj_sel_o <= axi_wdata[4:0];
                        end
                        else begin
                                user_prj_sel_o <= user_prj_sel_o;
                        end
                end
        end
end
```

Finally, "user_prj_sel_o" changes

3. **Describe how FIR initialization from SOC side (Test#1) works.**

We've written a task about configurating the data from SOC side, let's use it to program data length and coefficients:

```
// Step 1. FIR initialization (tap parameter, length) from "SOC" side
        // Data length
        soc_cfg_write(32'h3000_0010, 4'b0001, 64);
        // Coefficients
        for (i = 0; i < 11; i = i + 1) begin
            soc_cfg_write(32'h3000_0020 + 4 * i, 4'b0001, coef[i]);
        end
        // Start the FIR (ap_start)
        soc_cfg_write(32'h3000_0000, 4'b0001, 1);
```

Notice that we only have 1K range for User Projects (32'h3000_0xxx)

4. **Describe how FIR initialization from FPGA side (Test#2) works.**

Since we need to transfer data from FPGA side to SOC side, we need to use the AXILite-AXIS(AA) module to generate the AXI-Lite write transaction. The AA module is defined by FSIC which can convert AXI-Lite transaction to modified AXI-Stream transaction, and vice versa.

The transaction types are defined in the following table:

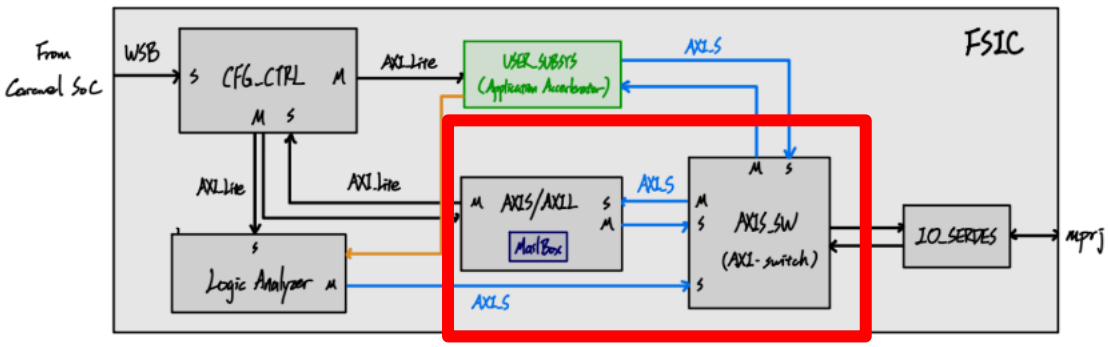
TUSER<1:0>	# of T	Transaction Type
00	n	Data payload for axis transaction. Limitation: all User pojects Data payload in asix MUST <= Max_axis_Data_payload(=32)
01	2	Axilite write transaction Address + Data. TAD 1^st^ T is the Byte-enable + address, i.e. {BE[3:0],ADDR[27:0]}, 2^nd^ T is the Data[31:0]. Note: Axilite write transaction only support 1T in data phase.
10	1	Axilite read Command (Address Phase). TAD<31:0> is the address ADDR[31:0]
11	1	Axilite read Completion (Data Phase). TAD<31:0> is the return data DATA[31:0]. Note: 1. Axilite read transaction only support 1T in data phase(Axilite read Completion). 2. If Axilite read Command is Upstream then the Axilite read Completion is Downstream, and vice versa.

We should program TUSER=2'b01.

Inside the task-fpga_axilite_write_req:

```
fpga_as_is_tid <=  TID_DN_AA;                  //target to Axis-Axilite
fpga_as_is_tuser <=  TUSER_AXILITE_WRITE;      //for axilite write req
localparam TUSER_AXIS = 2'b00;
localparam TUSER_AXILITE_WRITE = 2'b01;
localparam TUSER_AXILITE_READ_REQ = 2'b10;
localparam TUSER_AXILITE_READ_CPL = 2'b11;
```

Let's look at the following block diagram of FSIC,



Because we send data from FPGA side, our transaction flows downstream, we need to decide its destination refer to the table defined under AS(AXI-Switch) module below:

Direction	TID[1:0]	Source Module	Destination Module
Downstream	00	User DMA (M_AXIS_MM2S) in remote host (option extended user project)	User Project - the current active user project
Downstream	01	Axilite Master R/W in remote host (include Mail box write)	Axis-Axilite (include Mail box)
Upstream	00	User Project - the current active user project	User DMA (S_AXIS_S2MM) in remote host (option extended user project)
Upstream	01	Axis-Axilite (for Mail box)	Axilite slave in remote host (for mail box write)
Upstream	10	Logic Analyzer	Logic Analyzer data receiver - DMA (S_AXIS_S2MM) in remote host

We should program TID=2'b01 such that AA module can do the work I've mentioned before.

Inside the task fpga_axilite_write_req:

```
fpga_as_is_tid <=  TID_DN_AA;                  //target to Axis-Axilite
fpga_as_is_tuser <=  TUSER_AXILITE_WRITE;      //for axilite write req

localparam TID_DN_UP = 2'b00;
localparam TID_DN_AA = 2'b01;
localparam TID_UP_UP = 2'b00;
localparam TID_UP_AA = 2'b01;
localparam TID_UP_LA = 2'b10;
```

And we only need use the task above to achieve our goal:

```
/*================ FPGA to SOC configuration write ===============*/
// Refer to task6_fpga_to_soc_cfg_write
task fpga_to_soc_cfg_write;
    input [27:0] f2s_addr;
    input [31:0] f2s_data;
    begin
        @(posedge fpga_coreclk);
        // cfg_read_data_expect_value = 32'h1;
        fpga_axilite_write_req(f2s_addr , 4'b0001, f2s_data);
        // write address = h0000_2100 ~ h0000_2FFF for AA internal register
        // fpga wait for write to soc
        repeat(100) @ (posedge soc_coreclk);
    end
endtask
```

5. **Describe how to feed in X data from FPGA side.**

   Since we need to stream data X from FPGA side, we may want to send the AXI-Stream request, here I wrote a task which can stream request cycles into SOC side.

```
634     /*==================== AXI Request for FIR ========================*/
635     // Refer to fpga_axis_req
636     task fir_fpga_axis_req;
637         input [31:0] data;
638         input [1:0]  tid;
639         input mode;     //o ffor noram, 1 for random data
640         `ifdef USER_PROJECT_SIDEBAND_SUPPORT
641             input [pUSER_PROJECT_SIDEBAND_WIDTH-1:0] upsb;
642         `endif
643         reg [31:0] tdata;
644         `ifdef USER_PROJECT_SIDEBAND_SUPPORT
645             reg [pUSER_PROJECT_SIDEBAND_WIDTH-1:0]tupsb;
646         `endif
647         reg [3:0] tstrb;
648         reg [3:0] tkeep;
649         reg       tlast;
650
651         reg [31:0] exp_data;
652
653         begin
654             if (mode) begin  //for random data
655                 tdata = $random;
656                 `ifdef USER_PROJECT_SIDEBAND_SUPPORT
657                     tupsb = $random;
658                 `endif
659                 tstrb = $random;
660                 tkeep = $random;
661                 tlast = $random;
662                 exp_data = tdata;
663             end else begin
664                 tdata = data;
665                 `ifdef USER_PROJECT_SIDEBAND_SUPPORT
666                     //tupsb = 5'b00000;
667                     //tupsb = tdata[4:0];
668                     tupsb = upsb;
669                 `endif
670                 tstrb = 4'b0000;
671                 tkeep = 4'b0000;
672                 //tstrb = 4'b1111;
673                 //tkeep = 4'b1111;
674                 tlast = upsb[1];   //set tlast = eol
675                 // exp_data = {tst_img_out_buf[idx3+3], tst_img_out_buf[idx3+2], tst_img_out_buf[idx3+1], tst_img_out_buf[idx3+0]};
676             end

677             `ifdef USER_PROJECT_SIDEBAND_SUPPORT
678                 fpga_as_is_tupsb <= tupsb;
679             `endif
680             fpga_as_is_tstrb <= tstrb;
681             fpga_as_is_tkeep <= tkeep;
682             fpga_as_is_tlast <= tlast;
683             fpga_as_is_tdata <= tdata;  //for axis write data
684             `ifdef USER_PROJECT_SIDEBAND_SUPPORT
685                 $strobe($time, "=> fpga_as_is_tdata = %x", fpga_as_is_tdata);
686             `else
687                 $strobe($time, "=> fpga_as_is_tdata = %x", fpga_as_is_tdata);
688             `endif
689
690             fpga_as_is_tid    <= tid;        //set target
691             fpga_as_is_tuser  <= TUSER_AXIS; //for axis req
692             fpga_as_is_tvalid <= 1;
693
694             soc_to_fpga_axis_expect_count <= soc_to_fpga_axis_expect_count+1;
695
696             @ (posedge fpga_coreclk);
697
698             // Wait until fpga_is_as_tready == 1 then change data
699             while (fpga_is_as_tready == 0) @ (posedge fpga_coreclk);
700
701             fpga_as_is_tvalid <= 0;
702
703         end
704     endtask
```

And since our goal is to stream the data from FPGA side to FIR module, which is under User Project Area, by the TID/TUSER table from the last part, we need to program TID=0 (destination: UserProject), TUSER=00 (AXIS transaction)

Finally, in our FIR data stream task, we keep using “fpga_axis_req” until finish data transfer.

```
/*==================== FPGA stream data X in ====================*/
task fpga_stream_x_in;
    `ifdef USER_PROJECT_SIDEBAND_SUPPORT
        reg [pUSER_PROJECT_SIDEBAND_WIDTH-1:0] upsb;
    `endif
    integer x;
    begin
        soc_to_fpga_axis_expect_count = 0;
        @ (posedge fpga_coreclk);
        fpga_as_is_tready <= 1;
        for (x = 0; x < 64; x = x + 1) begin
            soc_to_fpga_axis_expect_count <= soc_to_fpga_axis_expect_count + 1;
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                fir_fpga_axis_req(x, TID_DN_UP, 0, upsb);
                $display($time, "=> FIR data x = %x stream from FPGA to UserProj", x);
            `else
                fir_fpga_axis_req(x, TID_DN_UP, 0); // Downstream target to UserProject
                $display($time, "=> FIR data x = %x stream from FPGA to UserProj", x);
            `endif
        end
        $display($time, "=> Finish FIR data Streaming from FPGA side");
    end
endtask
```

6. **Describe how to get output Y in testbench, and how to comparison with golden values.**

First, we can see the infinite “while” loop and the “if” statement below:

```
while (1) begin
        @(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
=%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
t, fpga_is_as_tdata);
                soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tupsb, fpga_is_as_tstrb, fp

                $display($time, "=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis
=%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
t, 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 == fpga_axis_test_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
=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_c
                soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tstrb, fpga_is_as_tkeep , f
                $display($time, "=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_to_fpga_axis
=%x", soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured_count, soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_c
                soc_to_fpga_axis_captured_count = soc_to_fpga_axis_captured_count+1;
            end
            if ( (soc_to_fpga_axis_captured_count == fpga_axis_test_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
```

Above will keep detect the streaming output, and store into

“soc_to_fpga_axis_captured” and count the captured transaction cycles.

By these characteristics, we can detect whether we’ve got 64(data length) Y and

check the correctness by comparing captured data with golden values.

```
while (soc_to_fpga_axis_captured_count != 64) @ (posedge fpga_coreclk);
$display("##################################################################################################");
$display($time, "=> Got 64 Y");
4. Check if output data Y are correct
$display($time, "=> Check the value...");
$display($time, "=> expect Y[0] : 0     actual: %d", soc_to_fpga_axis_captured[0][31:0]);
$display($time, "=> expect Y[63]: 10614 actual: %d", soc_to_fpga_axis_captured[63][31:0]);
if ((soc_to_fpga_axis_captured[0][31:0] != 32'd0)||(soc_to_fpga_axis_captured[63][31:0] != 32'd10614)) begin
    error_cnt = error_cnt + 1;
    $display($time, "=> Calculation FAIL!!!");
end
else begin
    $display($time, "=> Pass the FIR from [SOC] side");
end
```

Here I checked the first and the last value after every single Y is captured.

7. **Screenshot of simulation results printed on screen and show that Test#1 and Test#2 complete successfully.**

- Test#1

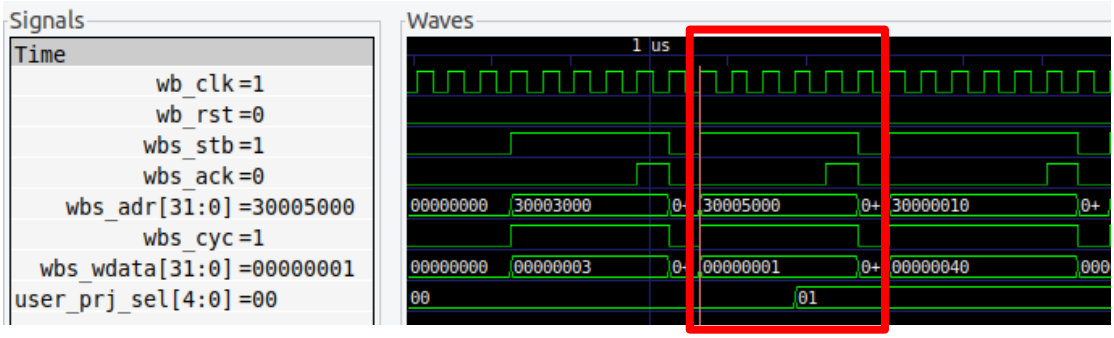
```
================================================================================
              42225=> Got 64 Y
              42225=> Check the value...
              42225=> expect Y[0] : 0     actual:          0
              42225=> expect Y[63]: 10614 actual:      10614
              42225=> Pass the FIR from [SOC] side
================================================================================
================================================================================
================================================================================
              42625=> Final result [PASS], check_cnt = 0000, error_cnt = 0000
================================================================================
================================================================================
$finish called at time : 42625 ns : File "/home/ubuntu/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tb_fsic.v" Line 448
## quit
INFO: [Common 17-206] Exiting xsim at Fri Mar 15 10:56:33 2024...
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tc$
```

- Test#2

```
================================================================================
              91105=> Got 64 Y
              91105=> Check the value...
              91105=> expect Y[0] : 0     actual:          0
              91105=> expect Y[63]: 10614 actual:      10614
              91105=> Pass the FIR from [FPGA] side
================================================================================
================================================================================
================================================================================
              91505=> Final result [PASS], check_cnt = 0000, error_cnt = 0000
================================================================================
================================================================================
$finish called at time : 91505 ns : File "/home/ubuntu/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tb_fsic.v" Line 448
## quit
INFO: [Common 17-206] Exiting xsim at Fri Mar 15 10:58:43 2024...
ubuntu@ubuntu2004:~/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tc$
```

8. Screenshot simulation waveform:

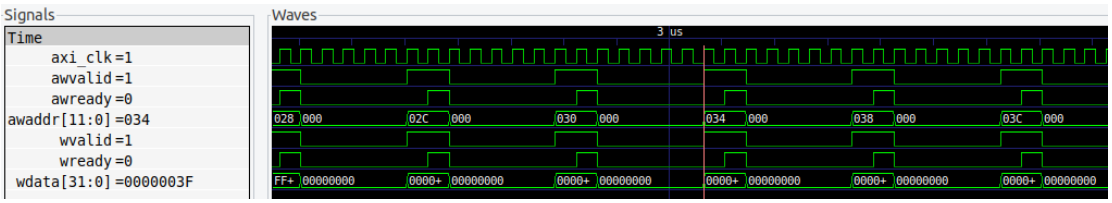
- Configuration cycle (program 32'h3000_5000, signal **user_prj_sel** changes)



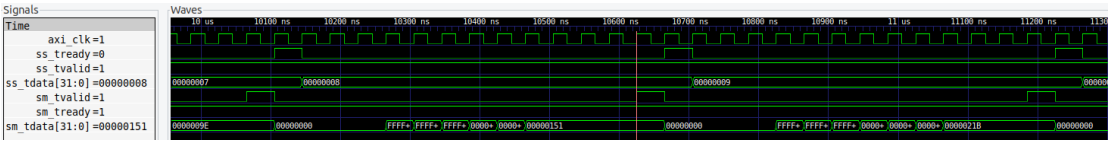
Takes 3T to change the user_prj_sel[4:0], and takes 1T to send back ACK.

Latency: 5T

- AXI-Lite transaction cycles



- Stream-in, Stream-out



X-in: ss_tdata / Y-out: sm_tdata

9. Debug experience.

After completing the testbench and starting the simulation, it's common to encounter situations where the entire simulation stalls. Initially, due to unfamiliarity with the design of the "tb_fsic", many signals were not understood, making it almost impossible to debug directly from the dumped waveform. Therefore, I began printing out information before and after tasks where stalling might occur to pinpoint where it stops.

Initializing the entire FSIC task posed no problem, but most issues arose during FPGA stream in and selecting the destination through the AXIS-Switch

module to configure the data. However, after reviewing the entire “tb_fsic” multiple times, I found many tasks that could serve as references. Also, it took some times comparing the golden value with Y-out, but after carefully reading the code, I found a method to verify using the register “soc_to_fpga_axis_captured”.