Advanced SoC Lab-fsic_sim

111061545 陳揚哲

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

There are multiple steps in every task: First, it will reset the system. Second, the system is initialized. Third, it enables the RX and TX side in system.

```
task test002;       //test002_fpga_axis_req
    //input [7:0] compare_data;

    begin
        for (i=0;i<CoreClkPhaseLoop;i=i+1) begin
            $display("test002: fpga_axis_req - loop %02d", i);
            fork
                soc_apply_reset(40+i*10, 40);
                fpga_apply_reset(40,40);
            join
            #40;

                        test001_up_soc_cfg; //config again because of soc_apply_reset()

            fpga_as_to_is_init();
                  Ubuntu, last month • update fsic-sim
            //soc_cc_is_enable=1;
            fpga_cc_is_enable=1;
            fork
                soc_is_cfg_write(0, 4'b0001, 1);
                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);
                fpga_cfg_write(0,3,1,0);
            join
            $display($time, "=> soc txen_ctl=1");
            $display($time, "=> fpga txen_ctl=1");
```

reset //change coreclk phase in soc

//fix coreclk phase in fpga

initialize

RX //ioserdes rxen

TX //ioserdes txen

From the step which is enable the RX and TX side in system, we can find out the how the system programs the specific address.

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

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= IS_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_is_cfg_write : wbs_adr=%x, wbs_sel=%b, wbs_wdata=%x", wbs_adr, wbs_sel,
    end      Ubuntu, last month • update fsic-sim
endtask
```

Therefore, I add new task changing the target address to [‘h3000_5000].

```
localparam CS_BASE=32'h3000_5000;
```

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

    begin
        @ (posedge soc_coreclk);
        wbs_adr <= CS_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_is_cfg_write : wbs_adr=%x, wbs_sel=%b, wbs_wdata=%x", wbs_adr, wbs_sel,
    end
endtask
```

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)

    From these code, we change the user_prj_sel[4:0] signal if (address[11:0] == 12’h000 && axi_wstrb[0] ==1). In the case, the program data would be the user_prj_sel signal.

```
////////////////////////////////////////
// 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;      Ubuntu, last month • update fsic-sim
    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

endmodule
```

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

    I use the soc_up_cfg_read and soc_up_cfg_write task to perform FIR initialization, since these tasks are programming the address base of 32’h3000_0000. First, I will check idle signal, then program the data length and taps coefficient and read back.

```
task FIR_test1;

    //read the address 3000_0000, and wait the idle signal = 1
    soc_up_cfg_read(12'h0,4'b1111);
    while (cfg_read_data_captured[2] == 0) begin      You, 4 minutes ago • Uncommitted changes
        soc_up_cfg_read(12'h0,4'b1111);
    end

    //write the data length
    soc_up_cfg_write(12'h010,4'b1111,32'd64);
    //write the taps coefficient
    for(idx=0;idx<11;idx=idx+1) begin
        soc_up_cfg_write(12'h020+idx*4,4'b1111,coef[idx]);
    end

    //read the data length
    soc_up_cfg_read(12'h010,4'b1111);
    //read the taps coefficient
    for(idx=0;idx<11;idx=idx+1) begin
        soc_up_cfg_read(12'h020+idx*4,4'b1111);
    end

    //write the ap_start signal
    soc_up_cfg_write(12'h0,4'b1111,32'd1);
endtask
```

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

   Similar to test#1, I use the fpga_axilite_read_req and fpga_axilite_write_req task to perform FIR initialization, since these tasks are programming the address base of 28'h0000_0000. First, I will check idle signal, then program the data length and taps coefficient and read back.

```
task FIR_test2;
    //read the address 3000_0000, and wait the idle signal = 1
    fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'h0000_0000);
    while (soc_to_fpga_axilite_read_cpl_captured[2] == 0) begin
        fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'h0000_0000)
    end

    //write the data length
    fpga_axilite_write_req(28'h010, 4'b1111,32'd64);
    //write the taps coefficient
    for(idx=0;idx<11;idx=idx+1) begin
        fpga_axilite_write_req(28'h020+idx*4, 4'b1111,coef[idx]);
    end

    //read the data length
    fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'h0000_0010);
    //read the taps coefficient
    for(idx=0;idx<11;idx=idx+1) begin
        fpga_axilite_read_req(FPGA_to_SOC_UP_BASE + 32'h0000_0020 + idx*4);
    end

    //write the ap_start signal
    fpga_axilite_write_req(28'h0,4'b1111,32'd1);
endtask
```

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

   In order to feed in the X data, I modify the code of test002_fpga_axis_req, which is the test of axis. I only need to change the test002_fpga_axis_req, rather than the task it used. The task fpga_axis_req will config fpga_as signal (AXIS-Switch).

```
task FIR_X;
    reg [31:0] data;
    `ifdef USER_PROJECT_SIDEBAND_SUPPORT
    reg [pUSER_PROJECT_SIDEBAND_WIDTH-1:0]upsb;
    `endif
    begin
        $display("input X data of FIR start");
        @ (posedge fpga_coreclk);
        fpga_as_is_tready <= 1;

        for(i=0;i<64;i=i+1) begin
            data = i;
            $display("index = %d, data = %d", i, data);
            sof = (i==0);
            eol = (i==63);
            `ifdef USER_PROJECT_SIDEBAND_SUPPORT
                upsb = {eol,sof};
                fpga_axis_req(data, TID_DN_UP, 0, upsb);
            `else
                fpga_axis_req(data, TID_DN_UP, 0);
            `endif
        end

        $display($time, "=> input X data of FIR done");
    end
endtask
```

```
        `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, f
        `else
            $strobe($time, "=> fpga_axis_req send data, fpga_as_is_tstrb = %b, fpga_as_is_tkeep = %b, f
        `endif

        fpga_as_is_tid <=  tid;     //set target
        fpga_as_is_tuser <=  TUSER_AXIS;        //for axis req
        fpga_as_is_tvalid <= 1;
        `ifdef USER_PROJECT_SIDEBAND_SUPPORT
            //soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tupsb, tstrb, tkeep, tla
            soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tupsb, tstrb, tkeep, tlast
        `else
            //soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tstrb, tkeep, tlast, tda
            soc_to_fpga_axis_expect_value[soc_to_fpga_axis_expect_count] <= {tstrb, tkeep, tlast, exp_d
        `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 dat
            @ (posedge fpga_coreclk);
        end
        fpga_as_is_tvalid <= 0;

    end      Ubuntu, last month • update fsic-sim
```

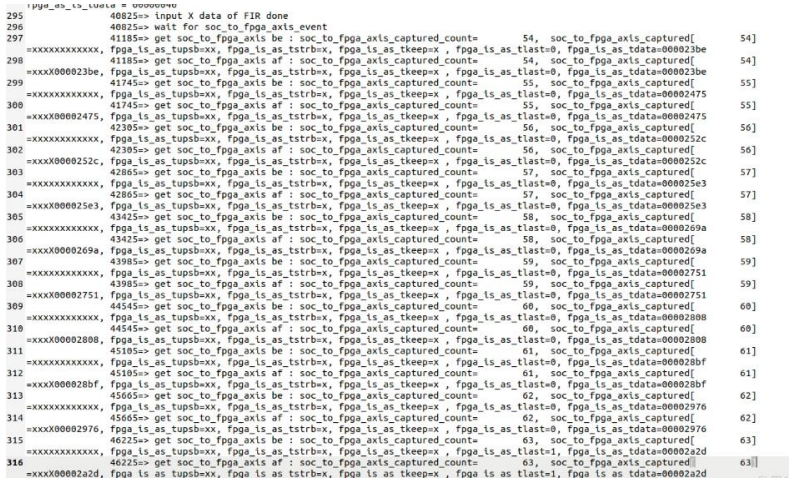
6. Briefly describe how you get output Y data in testbench, and how to do comparison with golden values.

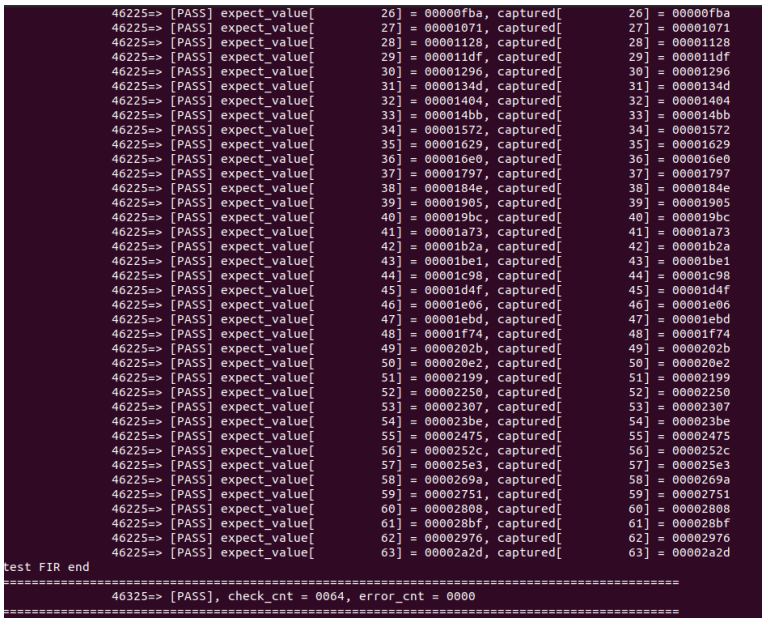
   I read back the output data Y by following code, which is the original code I didn't do any change. These code will read out and save the output in soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count][31:0].

```
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
        @(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
                $display($time, "=> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=%d,  soc_
                soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tupsb, fpga_is
                $display($time, "=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_
                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_
                $display($time, "=> soc_to_fpga_axis_captured : send soc_to_fpga_axiis_event");      Ub
                #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
                $display($time, "=> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=%d,  soc_
                soc_to_fpga_axis_captured[soc_to_fpga_axis_captured_count] = {fpga_is_as_tstrb, fpga_is
                $display($time, "=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=%d,  soc_
                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_
                $display($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 != fpga_axis_test_length)
            soc_to_fpga_axis_event_triggered = 0;

    end
```

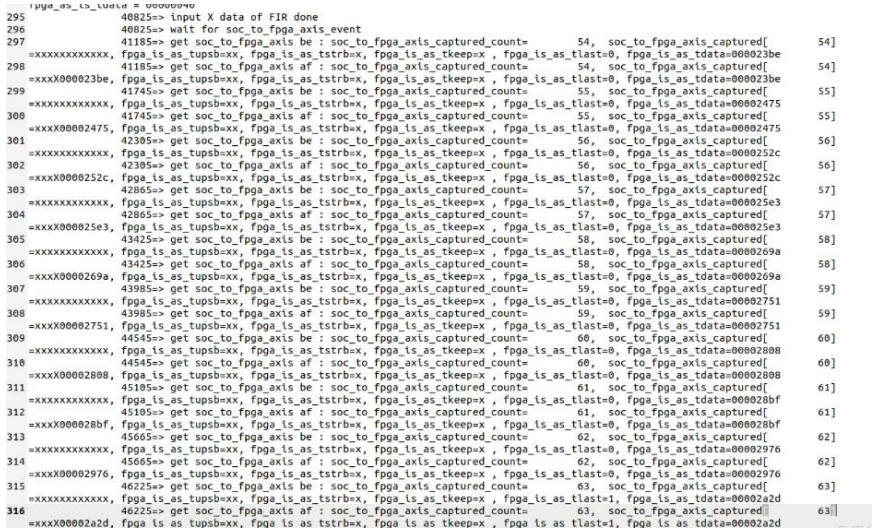
7. Screenshot simulation results printed on screen, to show that your Test#1 Test#2 complete successfully

Test#1:



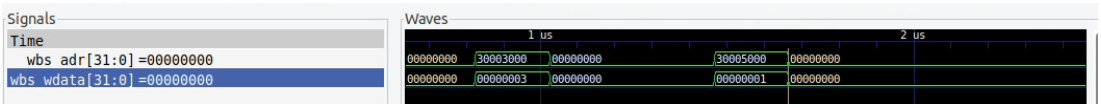


Test#2



```
             99265=> [PASS] expect_value[         34] = 00001572, captured[         34] = 00001572
             99265=> [PASS] expect_value[         35] = 00001629, captured[         35] = 00001629
             99265=> [PASS] expect_value[         36] = 000016e0, captured[         36] = 000016e0
             99265=> [PASS] expect_value[         37] = 00001797, captured[         37] = 00001797
             99265=> [PASS] expect_value[         38] = 0000184e, captured[         38] = 0000184e
             99265=> [PASS] expect_value[         39] = 00001905, captured[         39] = 00001905
             99265=> [PASS] expect_value[         40] = 000019bc, captured[         40] = 000019bc
             99265=> [PASS] expect_value[         41] = 00001a73, captured[         41] = 00001a73
             99265=> [PASS] expect_value[         42] = 00001b2a, captured[         42] = 00001b2a
             99265=> [PASS] expect_value[         43] = 00001be1, captured[         43] = 00001be1
             99265=> [PASS] expect_value[         44] = 00001c98, captured[         44] = 00001c98
             99265=> [PASS] expect_value[         45] = 00001d4f, captured[         45] = 00001d4f
             99265=> [PASS] expect_value[         46] = 00001e06, captured[         46] = 00001e06
             99265=> [PASS] expect_value[         47] = 00001ebd, captured[         47] = 00001ebd
             99265=> [PASS] expect_value[         48] = 00001f74, captured[         48] = 00001f74
             99265=> [PASS] expect_value[         49] = 0000202b, captured[         49] = 0000202b
             99265=> [PASS] expect_value[         50] = 000020e2, captured[         50] = 000020e2
             99265=> [PASS] expect_value[         51] = 00002199, captured[         51] = 00002199
             99265=> [PASS] expect_value[         52] = 00002250, captured[         52] = 00002250
             99265=> [PASS] expect_value[         53] = 00002307, captured[         53] = 00002307
             99265=> [PASS] expect_value[         54] = 000023be, captured[         54] = 000023be
             99265=> [PASS] expect_value[         55] = 00002475, captured[         55] = 00002475
             99265=> [PASS] expect_value[         56] = 0000252c, captured[         56] = 0000252c
             99265=> [PASS] expect_value[         57] = 000025e3, captured[         57] = 000025e3
             99265=> [PASS] expect_value[         58] = 0000269a, captured[         58] = 0000269a
             99265=> [PASS] expect_value[         59] = 00002751, captured[         59] = 00002751
             99265=> [PASS] expect_value[         60] = 00002808, captured[         60] = 00002808
             99265=> [PASS] expect_value[         61] = 000028bf, captured[         61] = 000028bf
             99265=> [PASS] expect_value[         62] = 00002976, captured[         62] = 00002976
             99265=> [PASS] expect_value[         63] = 00002a2d, captured[         63] = 00002a2d
test FIR end
================================================================================
             99365=> [PASS], check_cnt = 0064, error_cnt = 0000
================================================================================
================================================================================
================================================================================
================================================================================
             99765=> Final result [PASS], check_cnt = 0064, error_cnt = 0000
================================================================================
================================================================================
================================================================================
```

8. Screenshot simulation waveform:
   - Configuration cycle (when we program [‘h3000_5000] = 32’h01, signal user_prj_sel changes accordingly)

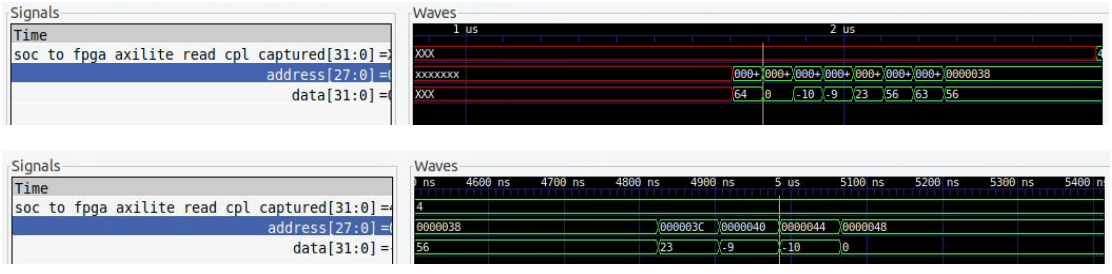


   - AXI-Lite transaction cycles (feed in tap parameters, data_length)

     Write the data length and tap parameters

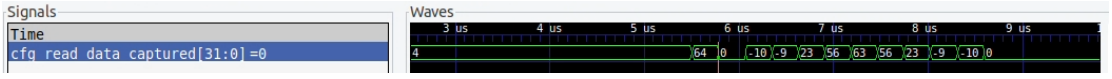
     From SOC side:

     From FPGA side:

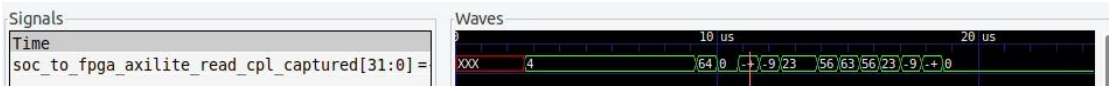


     Read back the data length and tap parameters

     From SOC side:

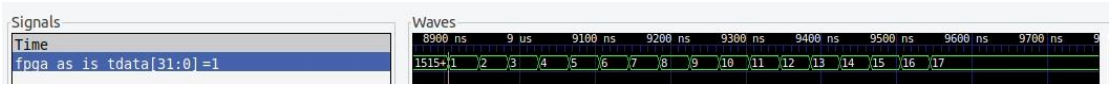


From FPGA side:



- Stream-in, Stream-out

  There are several threads like the waveform below, until stream-in data reach 64.



  The stream-out waveform shows the FIR will need several cycle to load new data and reset the previous output data.