PS-controlled Accelerator for FFT (Lab 3)

Chester Sungchung Park (박성정)
SoC Design Lab, Konkuk University
Webpage: http://soclab.konkuk.ac.kr

Outline

- Objectives
- Lab 3: PS-controlled accelerator

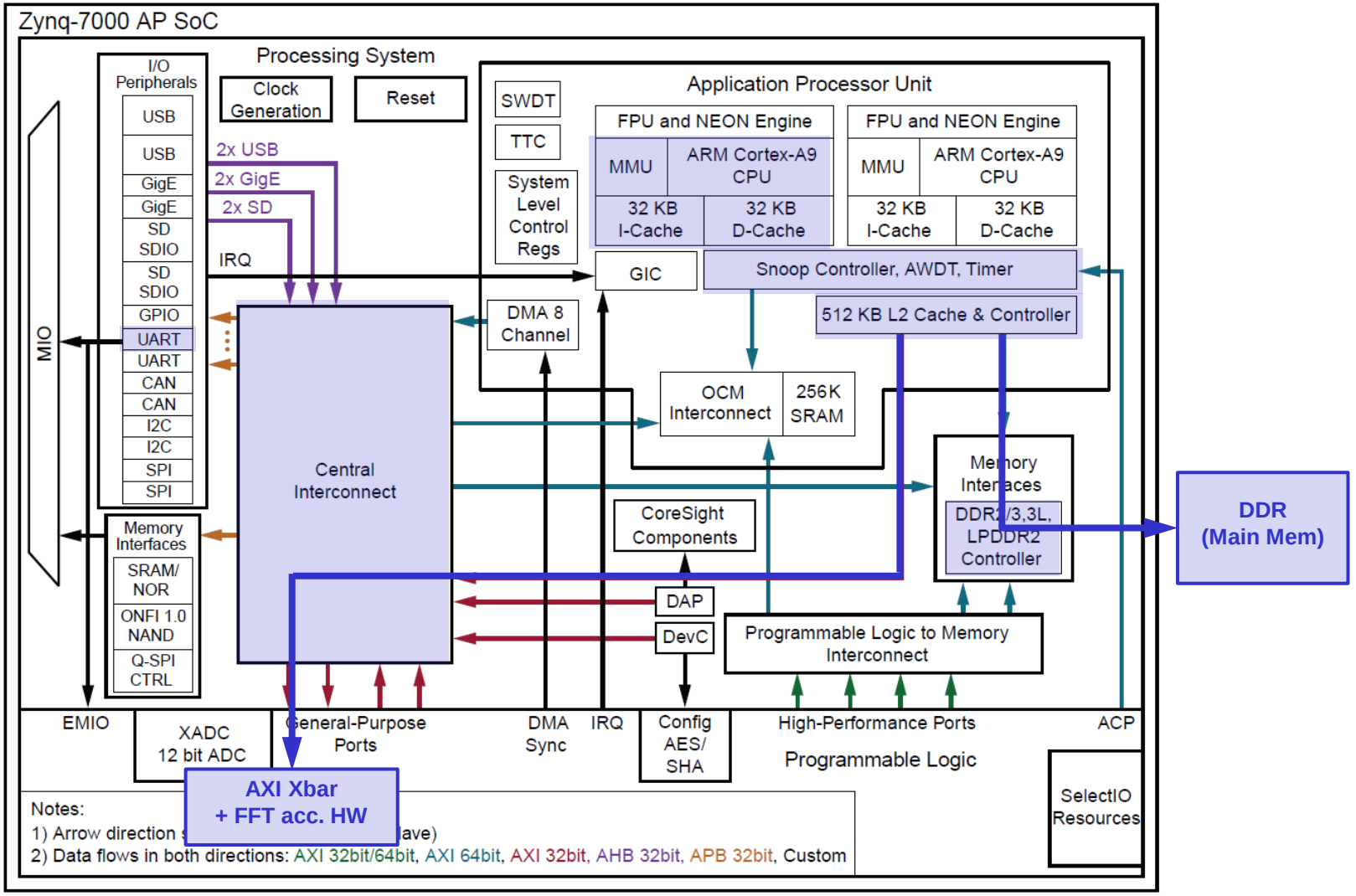
Objectives

- After completing this lab, you will be able to:
  - Create IP projects
  - Create block designs
  - Add additional IPs to block design from IP repository
  - Generate bitstream of the Vivado project
  - Run SDK applications with programmed FPGA
  - Measure hardware signals with Integrated Logic Analyzer (ILA)

Lab 3: PS-Controlled Accelerator

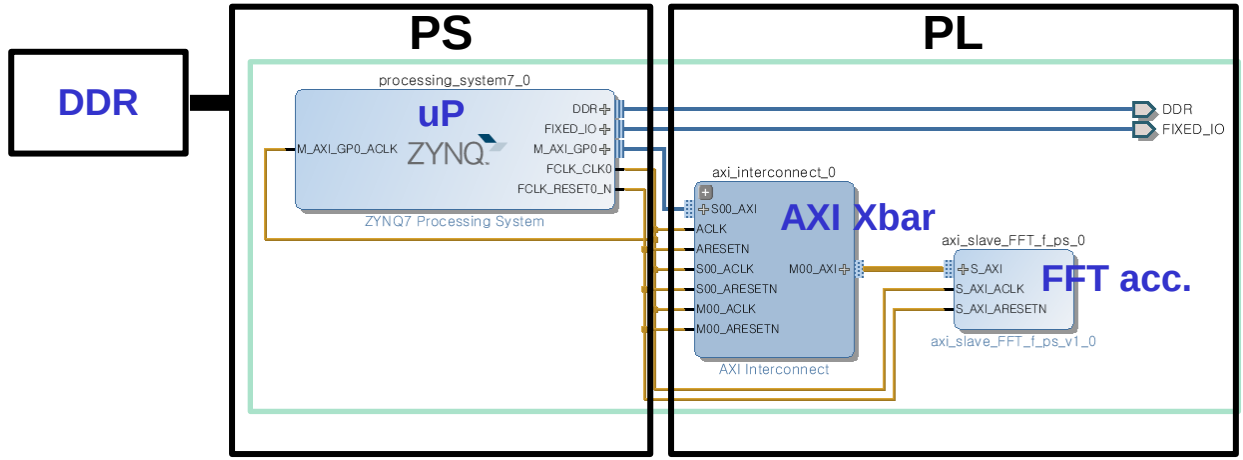
- Creating IP projects
- Creating block designs
- Generating bitstream
- Running C applications
- Debugging designs in Integrated Logic Analyzer (ILA)

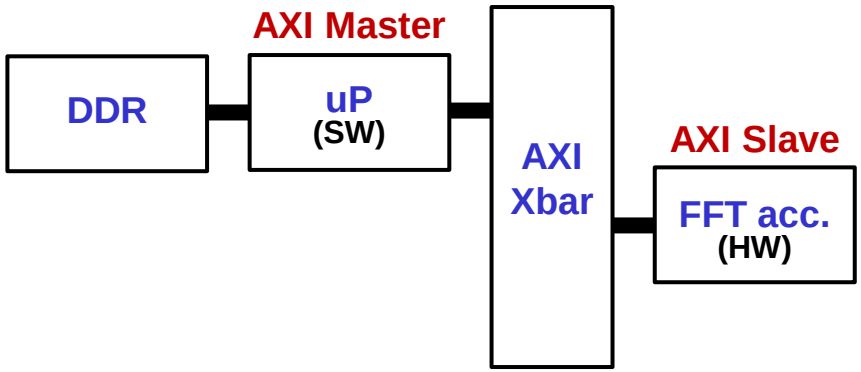
Block Diagram

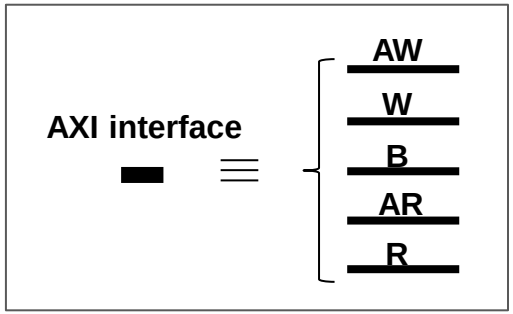


Block Diagram

- SoC integration

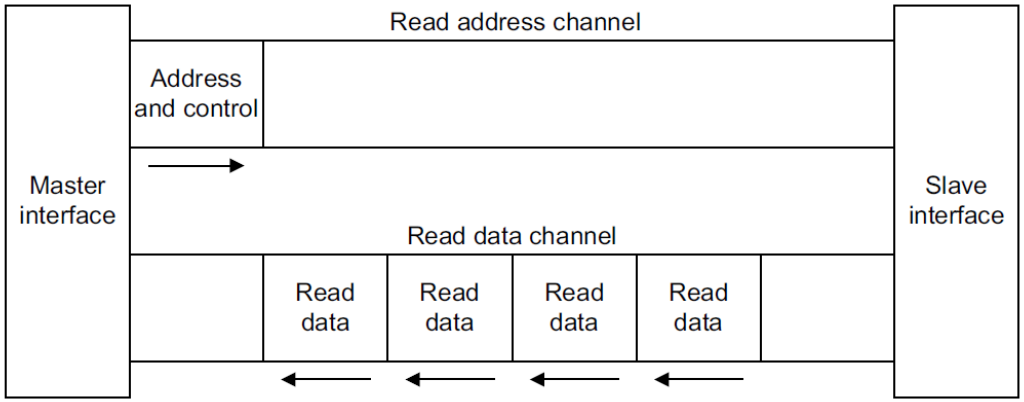


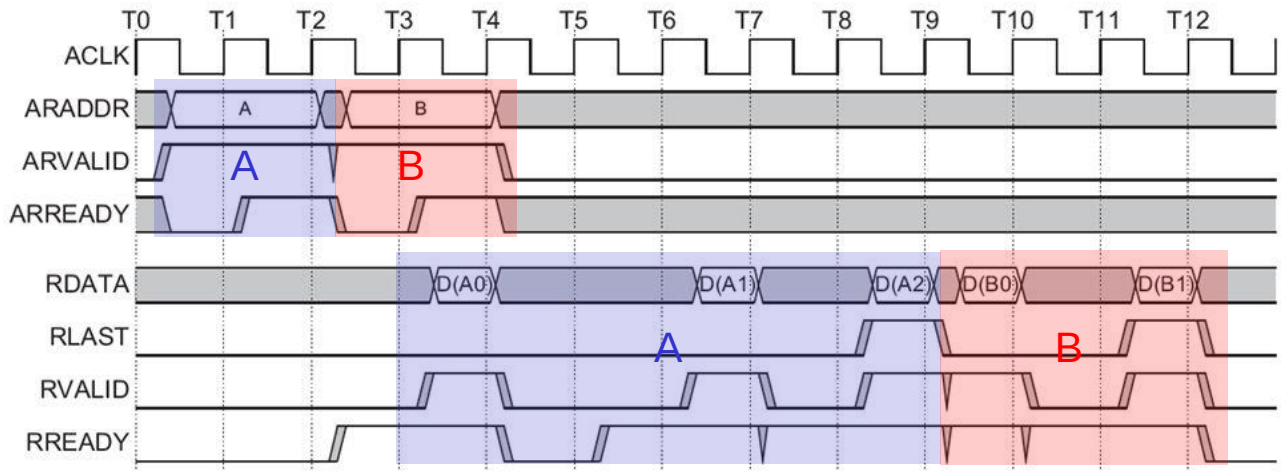




AMBA AXI

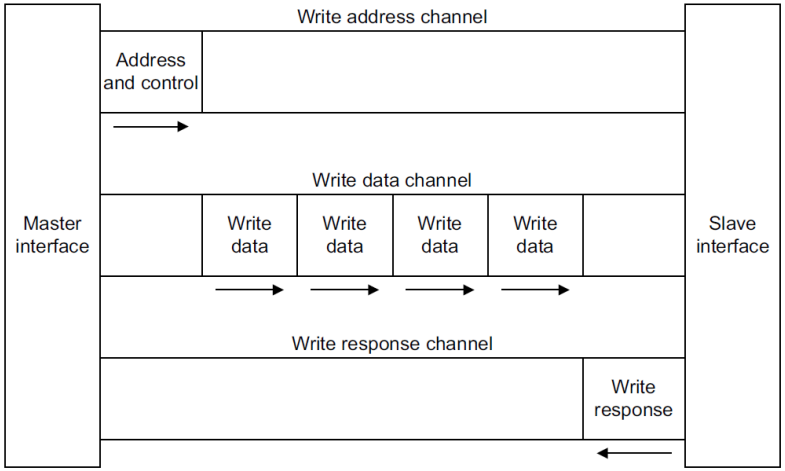
- AXI protocol: read

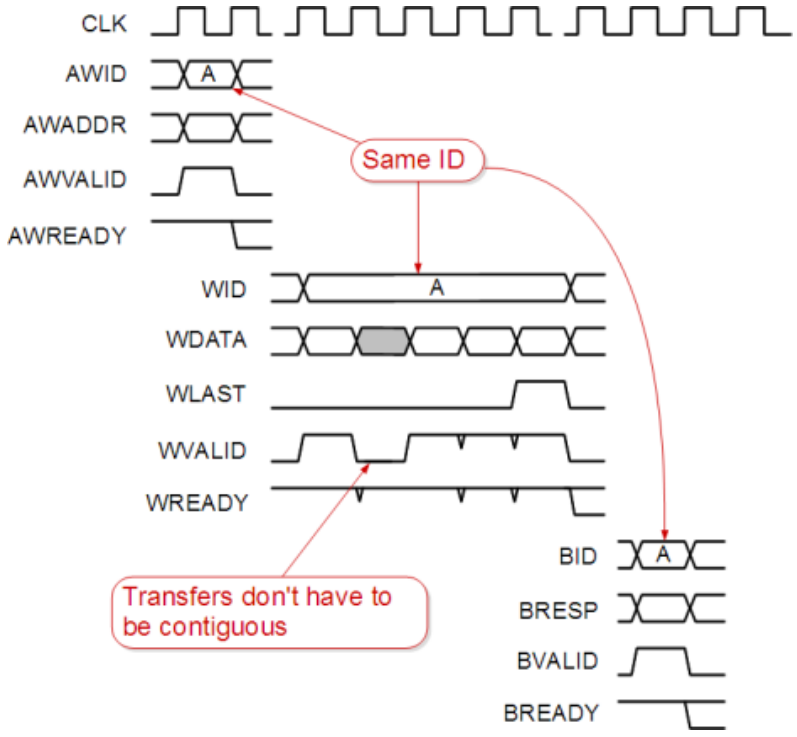




AMBA AXI

- AXI protocol: write





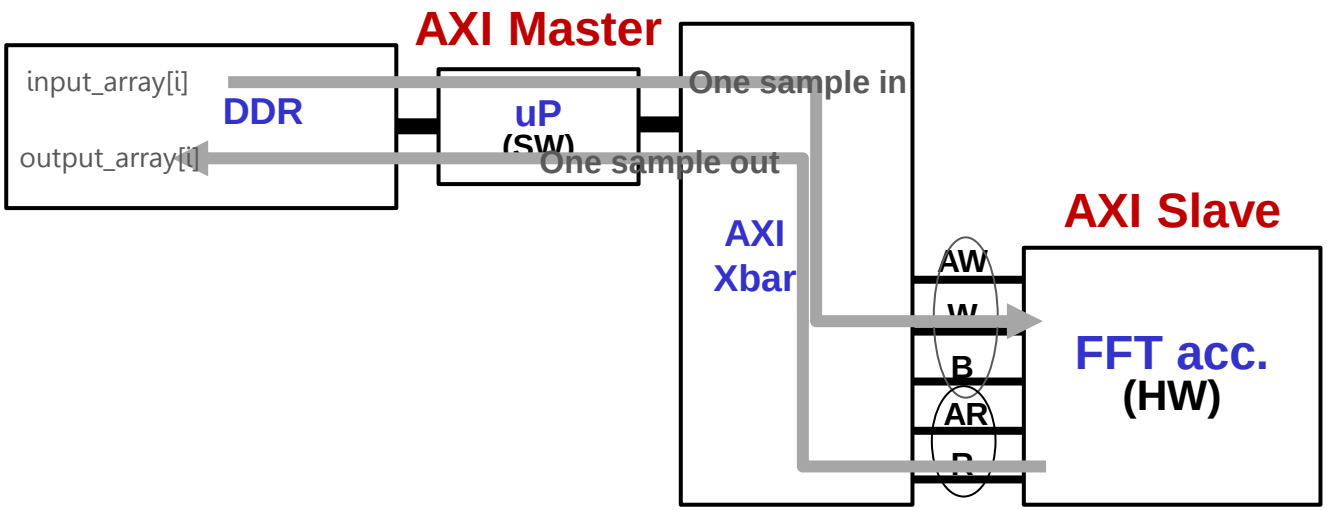
FFT Accelerator

- Dataflow

```
for(i = 0; i < 64; i++)
{
    Xil_Out32(IP_FOR_PS_BASE, input_array[i]);
    output_array[i] = Xil_In32(IP_FOR_PS_BASE + 4*i);
}
```

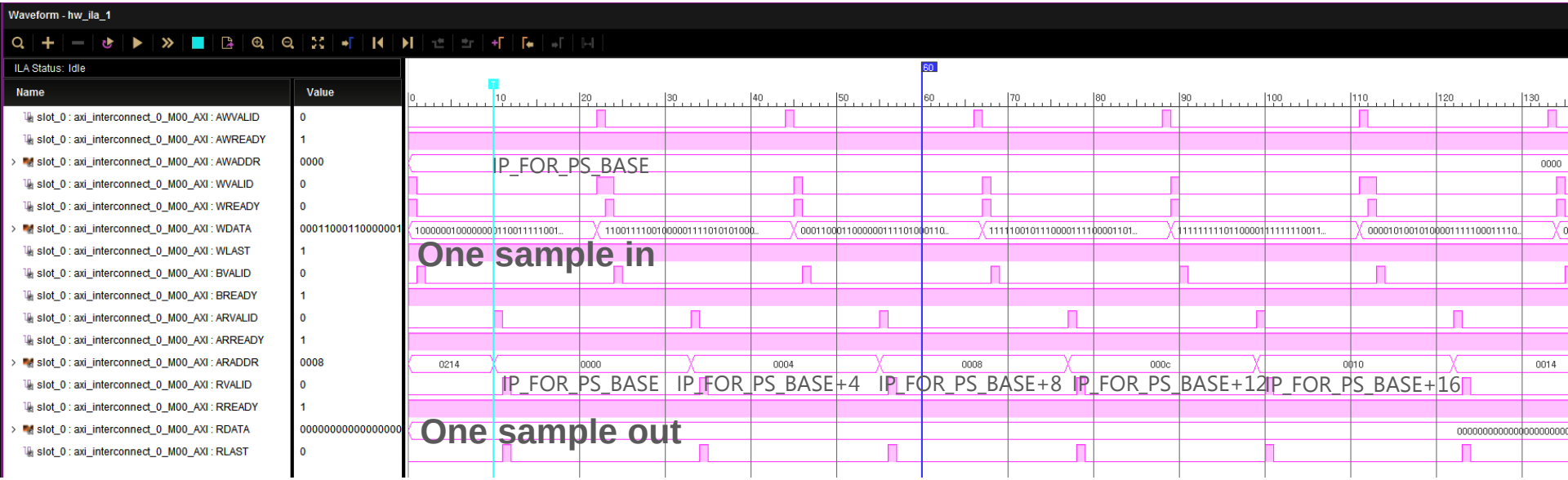
One sample in

One sample out



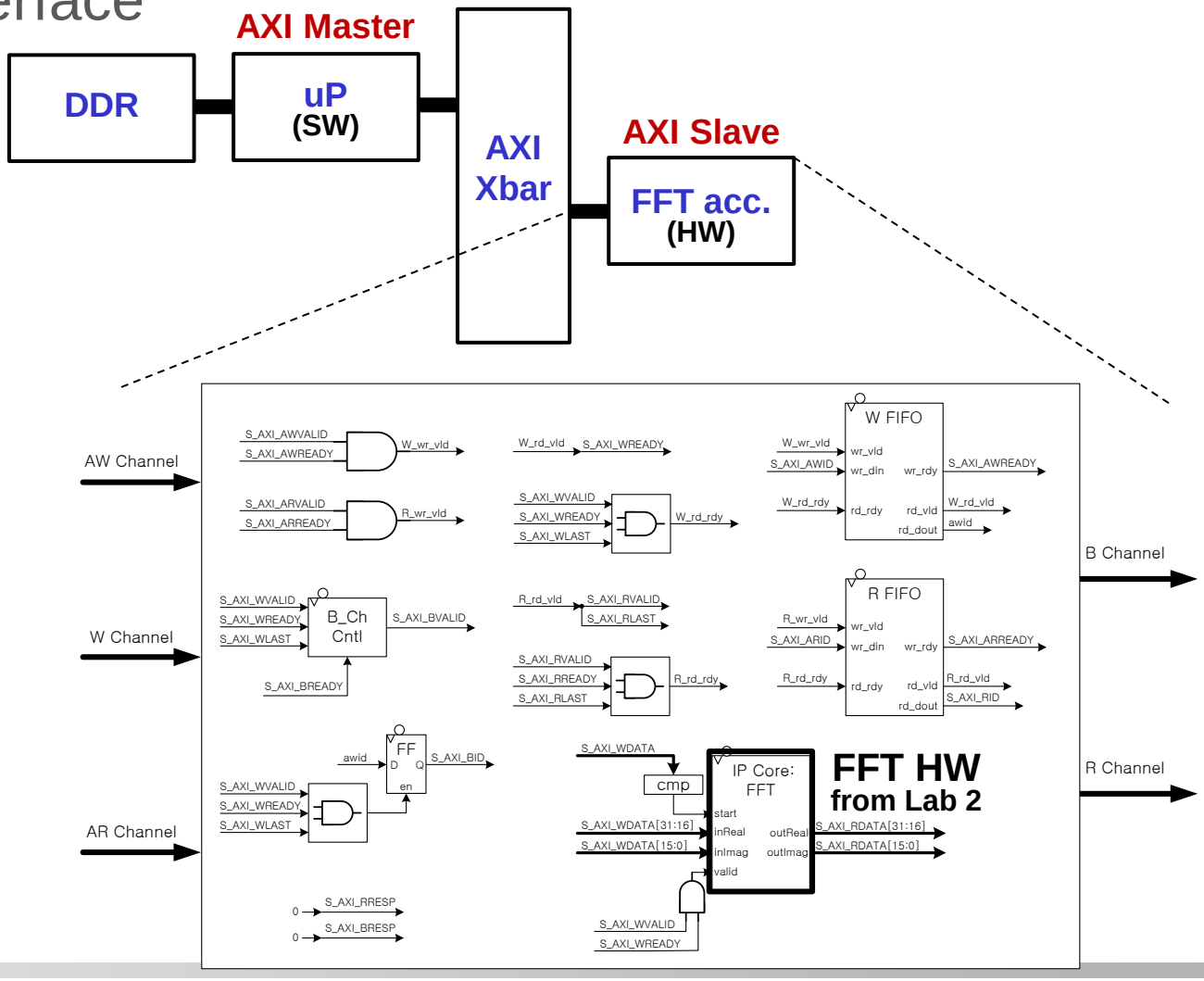
FFT Accelerator

- Timing diagram



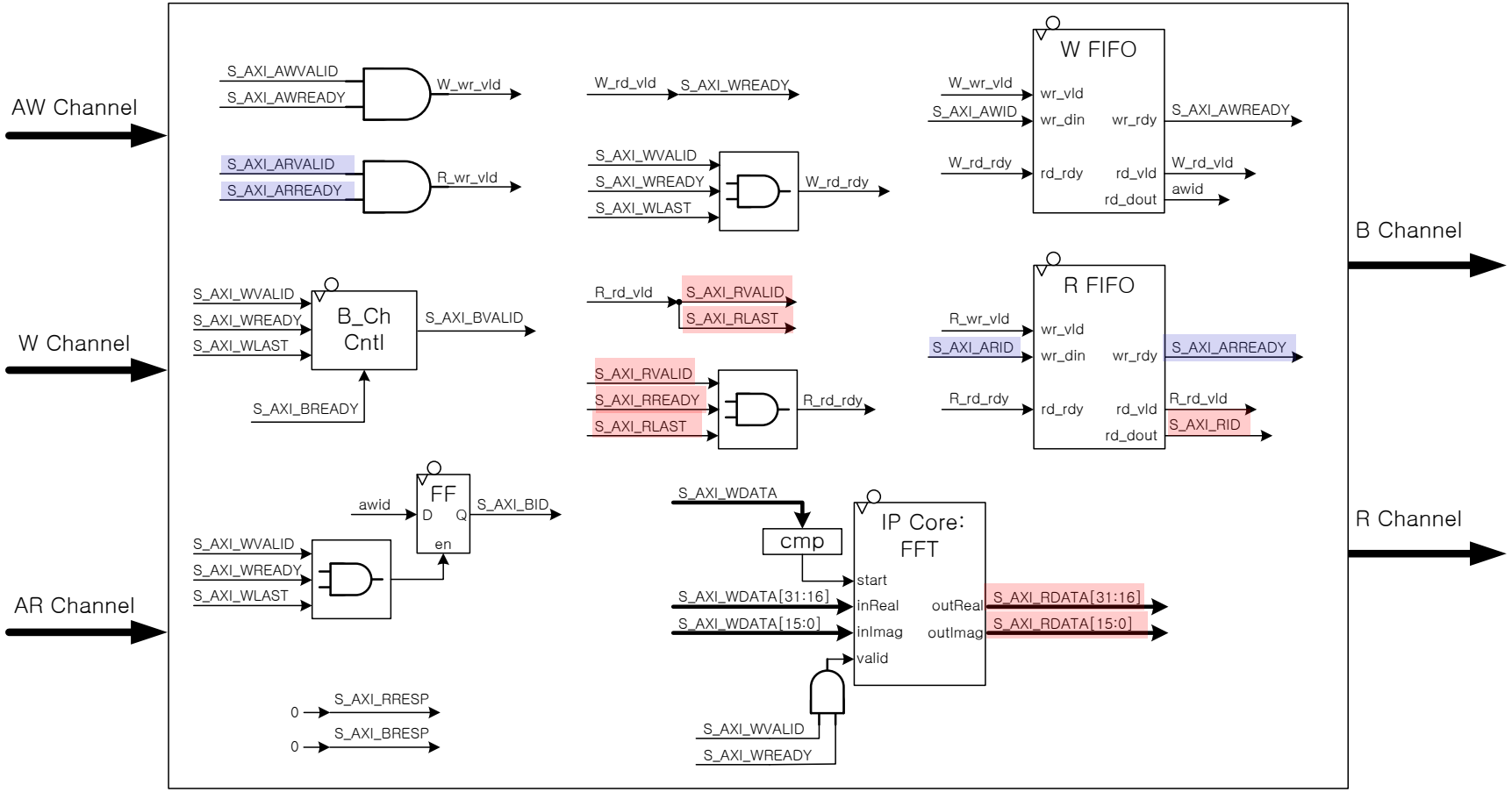
FFT Accelerator

- AXI interface



FFT Accelerator

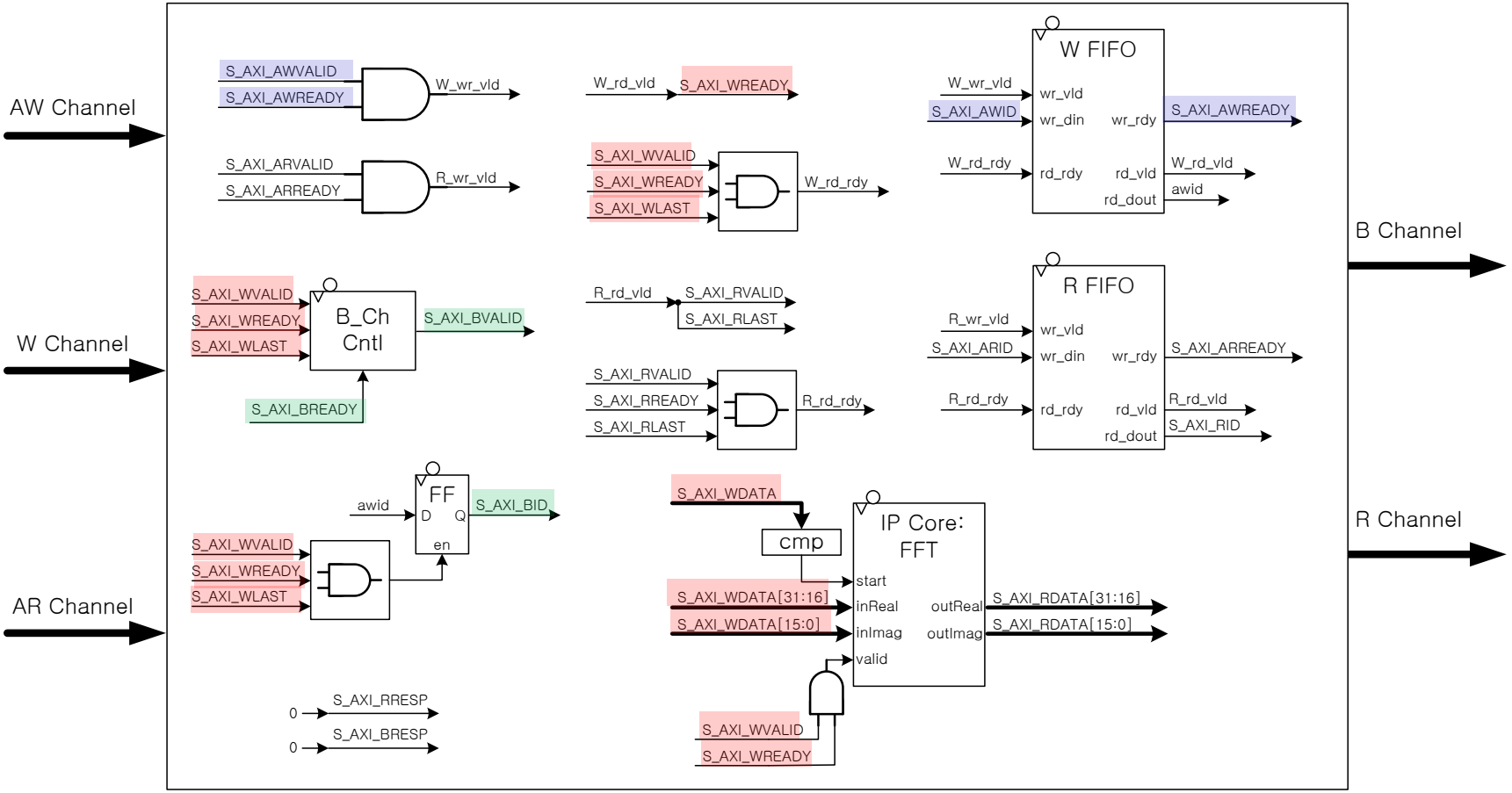
- AXI interface: read logic



Source: Sunwoo Kim

FFT Accelerator

- AXI interface: write logic



Source: Sunwoo Kim

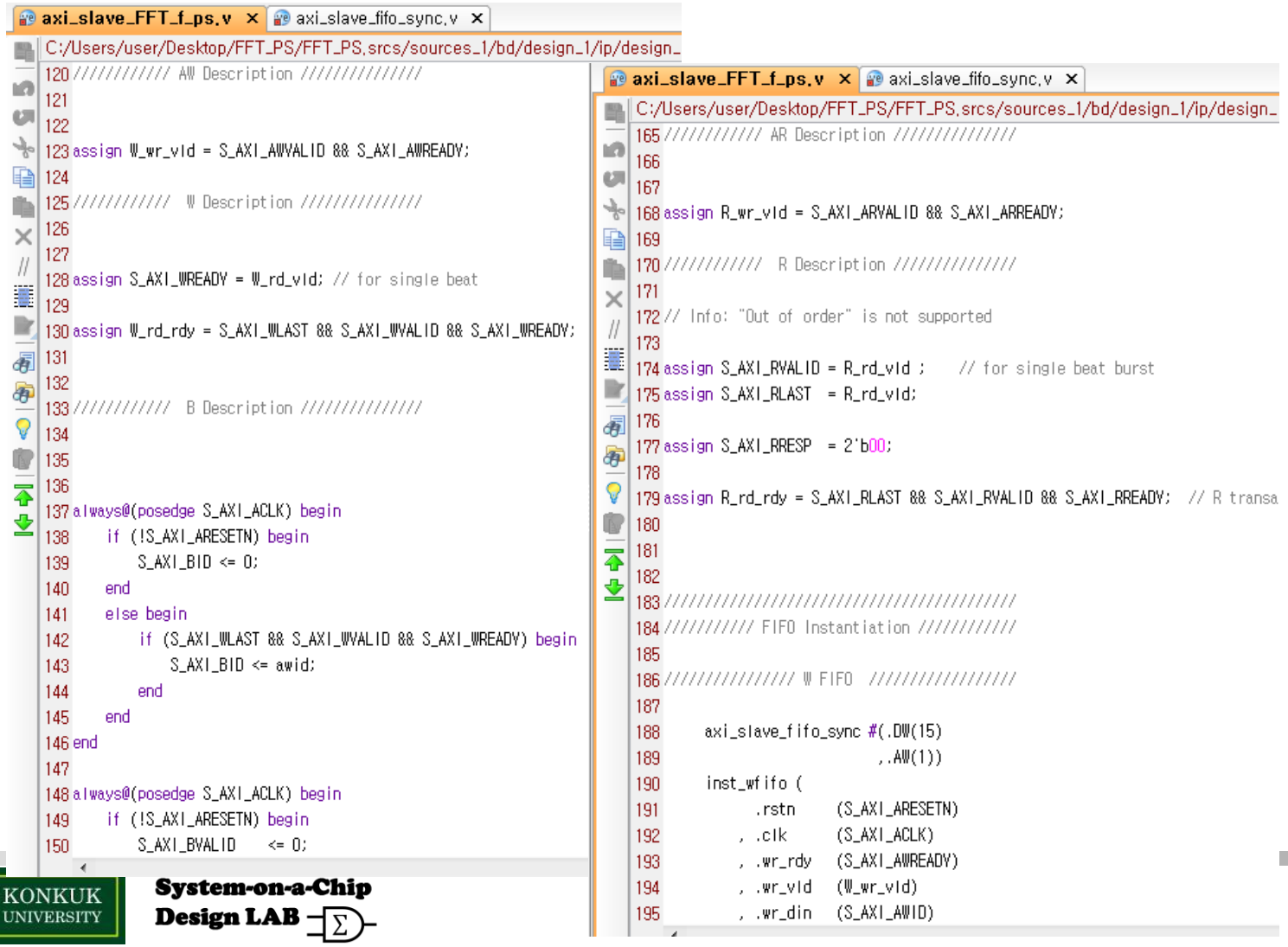
Source Codes

- main
  - ① Assigns input data into an array
  - ② Sends input data to and get output data from a HW block
  - ③ Converts output data to binary
  - ④ Prints binary output data

```
Xil_Out32(IP_FOR_PS_BASE,0x7FFFFFFF);

for(i = 0; i < 134; i++)                                    ①
{
    tmp  = inReal[i]<<16;
    tmp1 = (0x0000FFFF & inImag[i]);
    input_array[i] = tmp + tmp1;
}

for(i = 0; i < 64; i++)                                     ②
{
    Xil_Out32(IP_FOR_PS_BASE, input_array[i]);
    output_array[i] = Xil_In32(IP_FOR_PS_BASE + 4*i);
}

for(i = 0; i < 134; i++)                                    ③
{
    for(j=0;j<32;j++)
        if ((output_array[i]>>(31-j))&0x00000001)
            o[i][j] = '1';
        else
            o[i][j] = '0';
}

for(i = 0; i < 134; i++)                                    ④
{
    xil_printf("%3d: ",i);
    for(j=0;j<16;j++)
        xil_printf("%c",o[i][j]);
    xil_printf(" ");
    for(j=16;j<32;j++)
        xil_printf("%c",o[i][j]);
    xil_printf("\n");
}
```

Source Codes

- axi_slave_FFT_f_ps.v



axi_slave_FFT_f_ps.v | axi_slave_fifo_sync.v

C:/Users/user/Desktop/FFT_PS/FFT_PS.srcs/sources_1/bd/design_1/ip/design_

```
120 //////////// AW Description ///////////////
121
122
123 assign W_wr_vld = S_AXI_AWVALID && S_AXI_AWREADY;
124
125 ////////////  W Description ///////////////
126
127
128 assign S_AXI_WREADY = W_rd_vld; // for single beat
129
130 assign W_rd_rdy = S_AXI_WLAST && S_AXI_WVALID && S_AXI_WREADY;
131
132
133 ////////////  B Description ///////////////
134
135
136
137 always@(posedge S_AXI_ACLK) begin
138     if (!S_AXI_ARESETN) begin
139         S_AXI_BID <= 0;
140     end
141     else begin
142         if (S_AXI_WLAST && S_AXI_WVALID && S_AXI_WREADY) begin
143             S_AXI_BID <= awid;
144         end
145     end
146 end
147
148 always@(posedge S_AXI_ACLK) begin
149     if (!S_AXI_ARESETN) begin
150         S_AXI_BVALID    <= 0;
```

axi_slave_FFT_f_ps.v | axi_slave_fifo_sync.v

C:/Users/user/Desktop/FFT_PS/FFT_PS.srcs/sources_1/bd/design_1/ip/design_

```
165 //////////// AR Description ///////////////
166
167
168 assign R_wr_vld = S_AXI_ARVALID && S_AXI_ARREADY;
169
170 ////////////  R Description ///////////////
171
172 // Info: "Out of order" is not supported
173
174 assign S_AXI_RVALID = R_rd_vld ;    // for single beat burst
175 assign S_AXI_RLAST  = R_rd_vld;
176
177 assign S_AXI_RRESP  = 2'b00;
178
179 assign R_rd_rdy = S_AXI_RLAST && S_AXI_RVALID && S_AXI_RREADY;  // R transa
180
181
182
183 ////////////////////////////////////////////
184 ////////// FIFO Instantiation ////////////
185
186 ///////////////// W FIFO  /////////////////
187
188     axi_slave_fifo_sync #(.DW(15)
189                           ,.AW(1))
190     inst_wfifo (
191           .rstn     (S_AXI_ARESETN)
192         , .clk      (S_AXI_ACLK)
193         , .wr_rdy   (S_AXI_AWREADY)
194         , .wr_vld   (W_wr_vld)
195         , .wr_din   (S_AXI_AWID)
```

Source Codes

- axi_slave_fifo_sync.v

axi_slave_FFT_f_ps.v | axi_slave_fifo_sync.v

C:/Users/user/Desktop/FFT_PS/FFT_PS.srcs/sources_1/bd/design_1/ip/design_1

```
97   always @(posedge clk or negedge rstn) begin
98       if (rstn==1'b0) begin
99           item_cnt <= 0;
100      end else begin
101          if (wr_vld&&!full&&(!rd_rdy||(rd_rdy&&empty))) begin
102              item_cnt <= item_cnt + 1;
103          end else
104          if (rd_rdy&&!empty&&(!wr_vld||(wr_vld&&full))) begin
105              item_cnt <= item_cnt - 1;
106          end
107      end
108  end
109
110  //------------------------------------------------
111  assign rd_vld = ~empty;
112  assign wr_rdy = ~full;
113  assign empty  = (fifo_head == fifo_tail);
114  assign full   = (item_cnt>=DT);
115
116  reg [DW-1:0] Mem [0:DT-1];
117  assign rd_dout  = Mem[fifo_head[AW-1:0]];
118  always @(posedge clk) begin
119      if (!full && wr_vld) begin
120          Mem[fifo_tail[AW-1:0]] <= wr_din;
121      end
122  end
123  //------------------------------------------------
124 endmodule
125 //-------------------------------------------------------------
```

axi_slave_FFT_f_ps.v | axi_slave_fifo_sync.v

C:/Users/user/Desktop/FFT_PS/FFT_PS.srcs/sources_1/bd/desig

```
46 //-------------------------------------------------------------
47 `timescale 1ns/1ns
48
49 module axi_slave_fifo_sync #(parameter DW =42, AW =4 )
50 (
51       input  wire          rstn
52     , input  wire          clk
53     , output wire          wr_rdy
54     , input  wire          wr_vld
55     , input  wire [DW-1:0] wr_din
56     , input  wire          rd_rdy
57     , output wire          rd_vld
58     , output wire [DW-1:0] rd_dout
59 );
60   //------------------------------------------------
61   localparam DT = 1<<AW;
62
63   reg  [AW:0]   fifo_head; // where data to be read
64   reg  [AW:0]   fifo_tail; // where data to be written
65   reg  [AW:0]   next_tail;
66   reg  [AW:0]   next_head;
67   wire [AW-1:0] read_addr = (rd_vld&rd_rdy) ? next_head[AW-
68
```