

Modul 6

Pemrograman Lanjutan PS

1. Tujuan

- a. Membuat IP core berbasis AXI4-Lite pada PL.
- b. Membuat program untuk mengakses IP core.

2. Materi

a. AXI4 Bus

Advanced eXtensible Interface, atau AXI, merupakan bagian dari ARM's AMBA. Bus ini digunakan untuk menghubungkan antara IP core pada System-on-Chip.

3. Tutorial

Pada tutorial ini, kita akan melakukan beberapa hal sebagai berikut:

- Membuat modul multiplier pada PL.
- Membuat interface AXI-Lite untuk mengendalikan multiplier.
- Membuat program untuk mengendalikan multiplier dari PS.

Membuat modul multiplier, interface AXI-Lite, dan program PS.

1. Buat project baru pada Vivado, kemudian buat source baru untuk modul **multiplier.v**.

```
`timescale 1ns / 1ps

module multiplier
(
    input wire [31:0] a,
    input wire [31:0] b,
    output wire [31:0] c
);

    assign c = a * b;

endmodule
```

2. Buatlah kode Verilog untuk interface AXI-Lite **axi_lite.v**.

```
`timescale 1ns / 1ps

module axi_lite
(
    // ### Clock and reset signals
    #####
    input wire      aclk,
    input wire      aresetn,
    // ### AXI4-lite slave signals
    #####
    // *** Write address signals ***
    output wire      s_axi_awready,
    input wire [31:0] s_axi_awaddr,
    input wire      s_axi_awvalid,
    // *** Write data signals ***
    output wire      s_axi_wready,
    input wire [31:0] s_axi_wdata,
    input wire [3:0]  s_axi_wstrb,
    input wire      s_axi_wvalid,
    // *** Write response signals ***
    input wire      s_axi_bready,
    output wire [1:0] s_axi_bresp,
    output wire      s_axi_bvalid,
    // *** Read address signals ***
    output wire      s_axi_arready,
    input wire [31:0] s_axi_araddr,
    input wire      s_axi_arvalid,
    // *** Read data signals ***
    input wire      s_axi_rready,
    output wire [31:0] s_axi_rdata,
    output wire [1:0] s_axi_rresp,
    output wire      s_axi_rvalid
    // ### User signals
    #####
);

// ### Register map
#####
localparam C_ADDR_BITS = 8;
// *** Address ***
localparam C_ADDR_REG0 = 8'h00,
           C_ADDR_REG1 = 8'h04,
           C_ADDR_REG2 = 8'h08,
           C_ADDR_REG3 = 8'h0c;

// *** AXI write FSM ***
localparam S_WRIDLE = 2'd0,
           S_WRDATA = 2'd1,
           S_WRESP = 2'd2;

// *** AXI read FSM ***
localparam S_RDIDLE = 2'd0,
           S_RDDATA = 2'd1;

// *** AXI write ***
reg [1:0] wstate_cs, wstate_ns;
reg [C_ADDR_BITS-1:0] waddr;
```

```

wire [31:0] wmask;
wire aw_hs, w_hs;
// *** AXI read ***
reg [1:0] rstate_cs, rstate_ns;
wire [C_ADDR_BITS-1:0] raddr;
reg [31:0] rdata;
wire ar_hs;
// *** Registers ***
reg [31:0] reg0;
reg [31:0] reg1;
reg [31:0] reg2;
reg [31:0] reg3;
wire [31:0] c_w;

// ### AXI write
#####
assign s_axi_awready = (wstate_cs == S_WRIDLE);
assign s_axi_wready = (wstate_cs == S_WRDATA);
assign s_axi_bresp = 2'b00; // OKAY
assign s_axi_bvalid = (wstate_cs == S_WRRESP);
assign wmask = {{8{s_axi_wstrb[3]}}, {8{s_axi_wstrb[2]}},
{8{s_axi_wstrb[1]}}, {8{s_axi_wstrb[0]}}}};
assign aw_hs = s_axi_awvalid & s_axi_awready;
assign w_hs = s_axi_wvalid & s_axi_wready;

// *** Write state register ***
always @(posedge aclk)
begin
    if (!aresetn)
        wstate_cs <= S_WRIDLE;
    else
        wstate_cs <= wstate_ns;
end

// *** Write state next ***
always @(*)
begin
    case (wstate_cs)
        S_WRIDLE:
            if (s_axi_awvalid)
                wstate_ns = S_WRDATA;
            else
                wstate_ns = S_WRIDLE;
        S_WRDATA:
            if (s_axi_wvalid)
                wstate_ns = S_WRRESP;
            else
                wstate_ns = S_WRDATA;
        S_WRRESP:
            if (s_axi_bready)
                wstate_ns = S_WRIDLE;
            else
                wstate_ns = S_WRRESP;
        default:
            wstate_ns = S_WRIDLE;
    endcase
end

```

```

// *** Write address register ***
always @(posedge aclk)
begin
    if (aw_hs)
        waddr <= s_axi_awaddr[C_ADDR_BITS-1:0];
    end

// ### AXI read
#####
assign s_axi_arready = (rstate_cs == S_RDIDLE);
assign s_axi_rdata = rdata;
assign s_axi_rresp = 2'b00; // OKAY
assign s_axi_rvalid = (rstate_cs == S_RDDATA);
assign ar_hs = s_axi_arvalid & s_axi_arready;
assign raddr = s_axi_araddr[C_ADDR_BITS-1:0];

// *** Read state register ***
always @(posedge aclk)
begin
    if (!aresetn)
        rstate_cs <= S_RDIDLE;
    else
        rstate_cs <= rstate_ns;
    end

// *** Read state next ***
always @(*)
begin
    case (rstate_cs)
        S_RDIDLE:
            if (s_axi_arvalid)
                rstate_ns = S_RDDATA;
            else
                rstate_ns = S_RDIDLE;
        S_RDDATA:
            if (s_axi_rready)
                rstate_ns = S_RDIDLE;
            else
                rstate_ns = S_RDDATA;
        default:
            rstate_ns = S_RDIDLE;
    endcase
end

// *** Read data register ***
always @(posedge aclk)
begin
    if (!aresetn)
        rdata <= 0;
    else if (ar_hs)
        case (raddr)
            C_ADDR_REG0:
                rdata <= reg0[31:0];
            C_ADDR_REG1:
                rdata <= reg1[31:0];
            C_ADDR_REG2:

```

```

        rdata <= c_w[31:0];
        C_ADDR_REG3:
        rdata <= reg3[31:0];

    endcase

end

// ### Registers
#####
always @(posedge aclk)
begin
    if (!aresetn)
    begin
        reg0[31:0] <= 0;
        reg1[31:0] <= 0;
        reg2[31:0] <= 0;
        reg3[31:0] <= 0;
    end
    else if (w_hs && waddr == C_ADDR_REG0)
    begin
        reg0[31:0] <= (s_axi_wdata[31:0] & wmask) | (reg0[31:0] &
~wmask);
    end
    else if (w_hs && waddr == C_ADDR_REG1)
    begin
        reg1[31:0] <= (s_axi_wdata[31:0] & wmask) | (reg1[31:0] &
~wmask);
    end
    else if (w_hs && waddr == C_ADDR_REG2)
    begin
        reg2[31:0] <= (s_axi_wdata[31:0] & wmask) | (reg2[31:0] &
~wmask);
    end
    else if (w_hs && waddr == C_ADDR_REG3)
    begin
        reg3[31:0] <= (s_axi_wdata[31:0] & wmask) | (reg3[31:0] &
~wmask);
    end
    end

    multiplier multiplier_0
    (
        .a(reg0),
        .b(reg1),
        .c(c_w)
    );

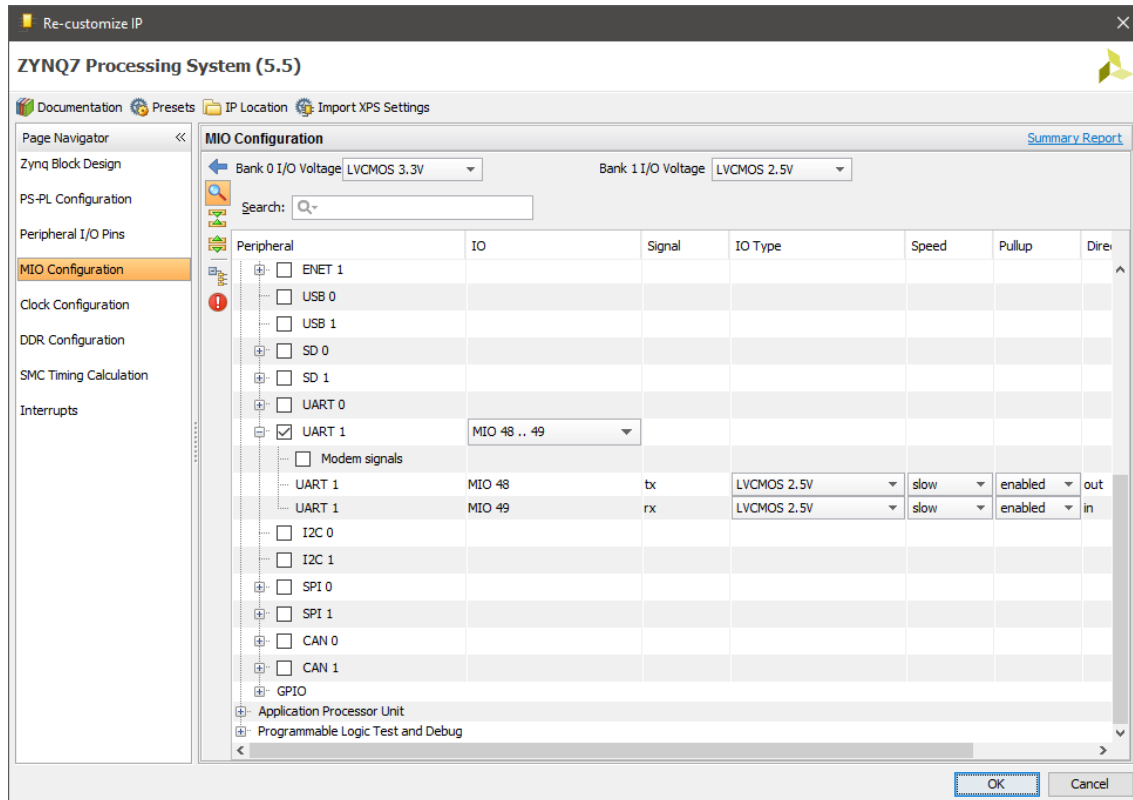
endmodule

```

3. Buat block design, dan tambahkan IP **ZYNQ7 Processing System** (detail step-by-step seperti pada modul 5).

4. Aktifkan **UART1** seperti gambar di bawah ini.

Enable UART1:

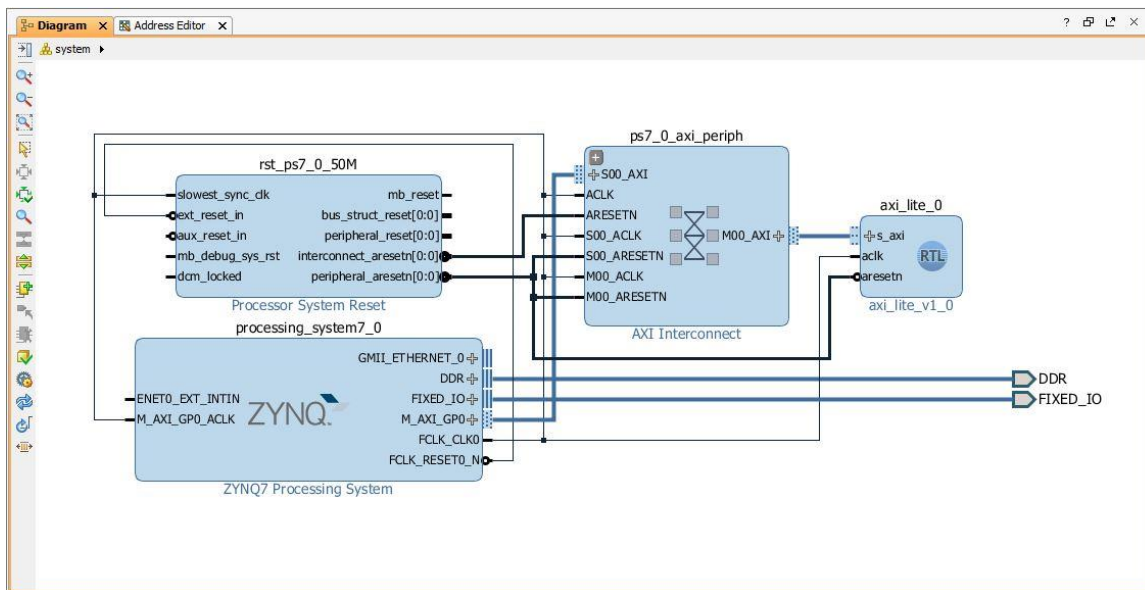


5. **Run Block Automation** untuk menghubungkan DDR dan fixed IO (detail step-by-step seperti pada modul 5).

6. Klik kanan pada block design kemudian pilih **Add Module**, pilih **axi_lite** module.

7. **Run Connection Automation** untuk menghubungkan IP axi_lite dengan PS (detail step-by-step seperti pada modul 5).

8. Hasil akhir seperti gambar berikut ini.



9. Buat **HDL wrapper**, kemudian **generate output products** (detail step-by-step seperti pada modul 5).

10. Run **synthesis**, **implementation**, dan **generate bitstream**.

11. **Export hardware**, kemudian **launch SDK**.

12. Buat project baru pada SDK dengan template helloworld, kemudian buat kode seperti ini:

```
#include <stdio.h>
#include <stdint.h>

uint32_t *multiplier_p = (uint32_t *)0x40000000;

int main()
{
    *(multiplier_p+0) = 25;
    *(multiplier_p+1) = 5;
    printf("%d\n", (unsigned int)*(multiplier_p+2));

    return 0;
}
```


13. Build project, kemudian upload bitsream dan elf file ke board. Hasil akhir seperti ini.

