Project Multiplier dan Block Memory BRAM

Erwin Setiawan

Created: 8 Nov 2018, Revision 1: 29 Nov 2022

Daftar Isi

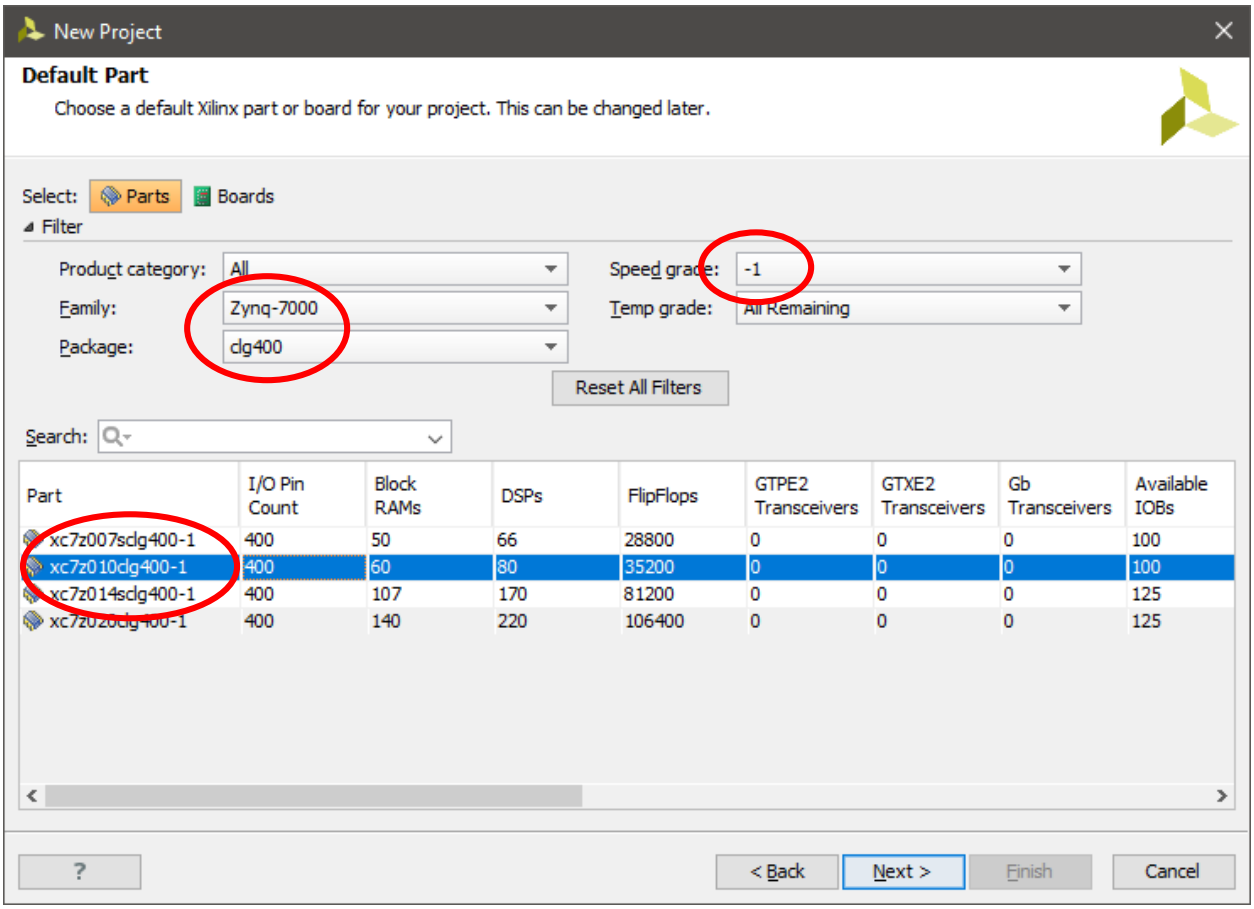
I. Simulasi Block Memory

Pada tutorial ini, akan dilakukan simulasi RTL dari IP Block Memory Generator.

1. Buat project baru dari **File→New Project**.

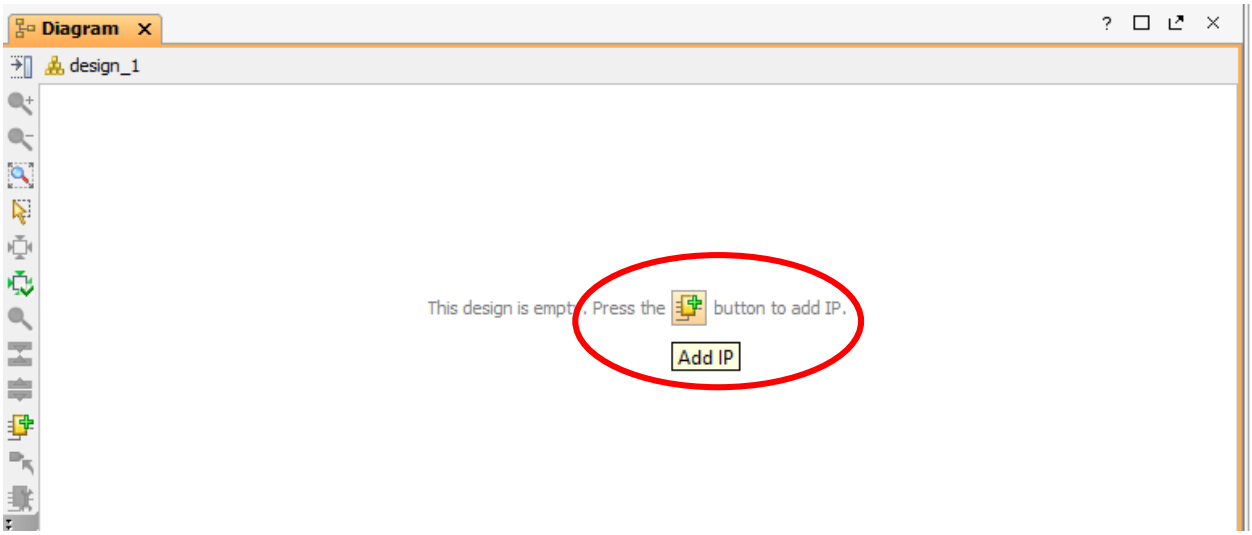
2. Masukan **Project name** dan **Project location**.

3. Pilih **RTL Project** dan enable **Do not specify sources at this time**.

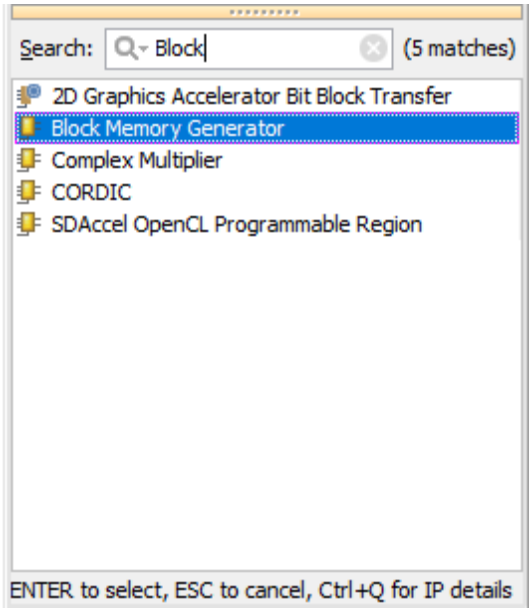


4. Pada **Flow Navigator**, pilih menu **Create Block Design** untuk membuat block design dengan nama default **design_1**.

5. Untuk menambahkan IP block dapat dilakukan dari tab **Diagram**, button **Add IP**.

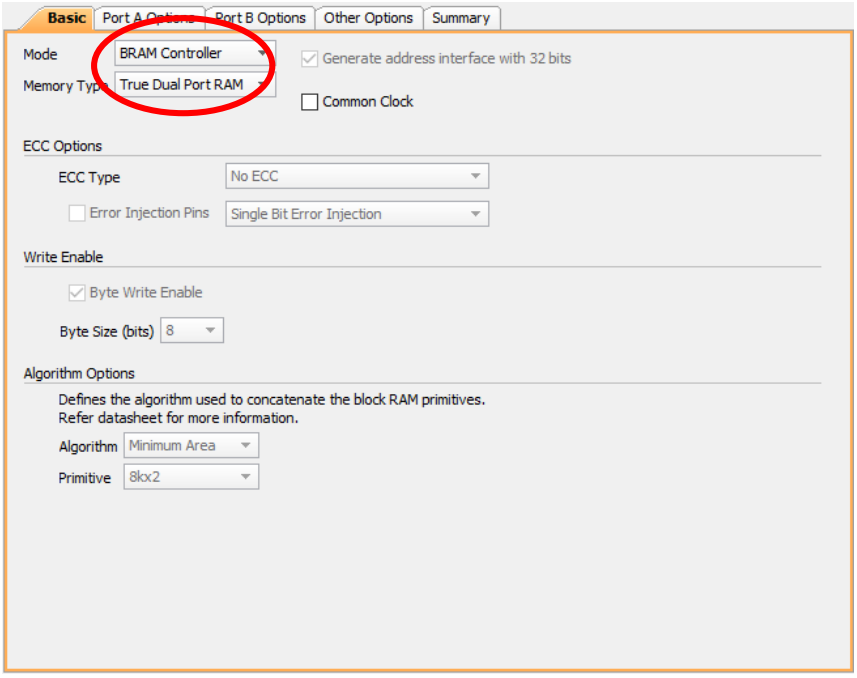


6. Pilih IP **Block Memory Generator**. Datasheet IP ini dapat didownload dari: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf

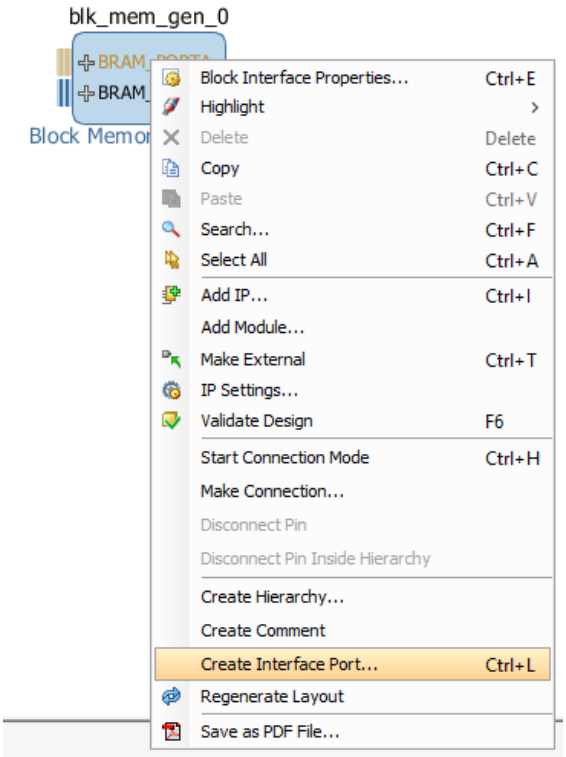


7. Double-click IP **Block Memory Generator** untuk melakukan konfigurasi.

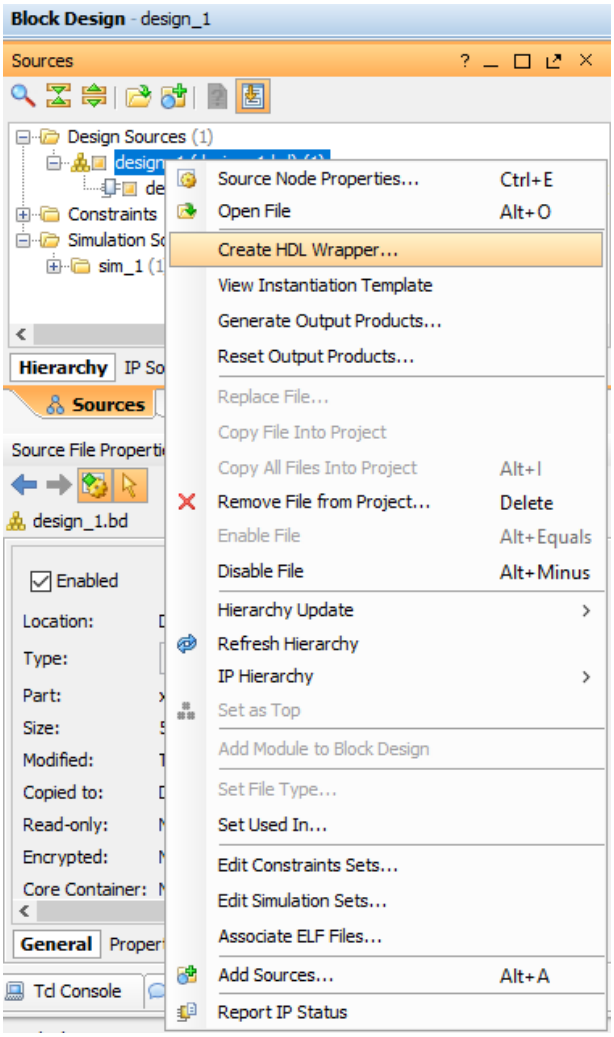
8. Pada tab **Basic**, ubah **Memory Type** menjadi **True Dual Port RAM**.



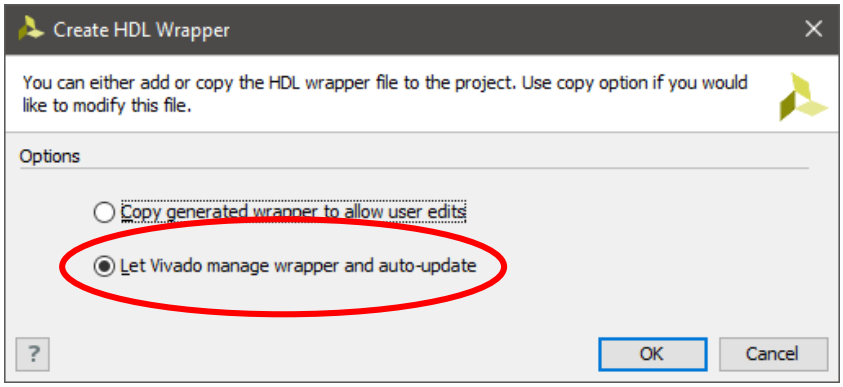
9. Right-click pada port **BRAM_PORTA** dan **BRAM_PORTB**, kemudian pilih **Create Interface Port**. Langkah ini berfungsi untuk membuat port agar dapat diakses dari top module.



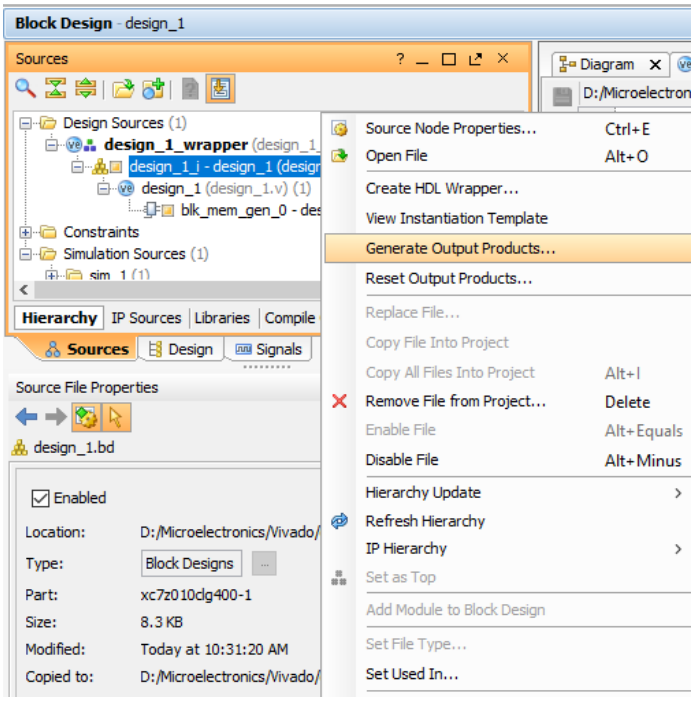
10. Buat wrapper HDL dari block design_1 dengan cara **right-click pada design_1** kemudian pilih **Create HDL Wrapper**.

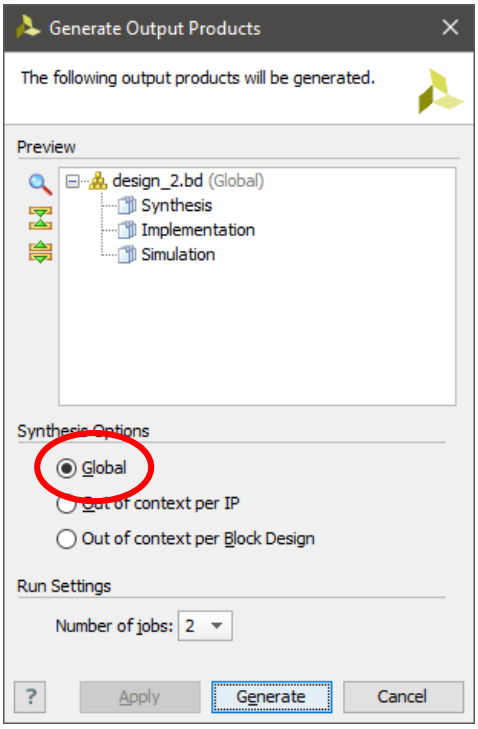


11. Pilih **Let Vivado manage wrapper and auto-update**, dengan pilihan ini, maka wrapper **design_1_wrapper.v** akan diupdate secara otomatis.

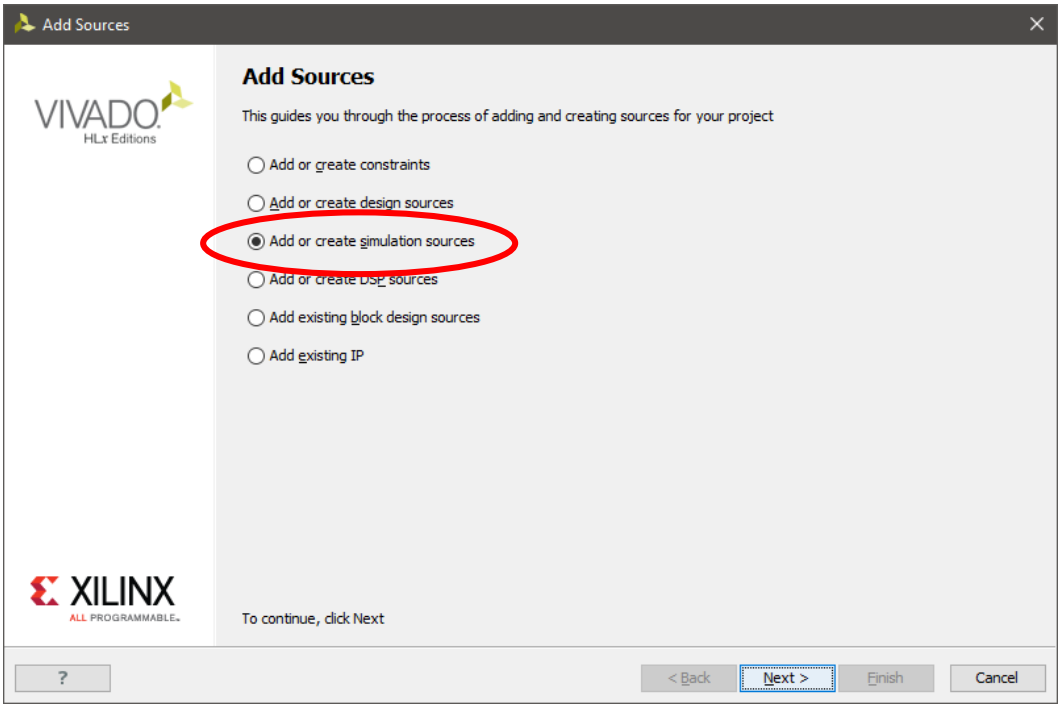


12. Generate code HDL dari block design_1 dengan cara **right-click pada design_1** kemudian pilih **Generate Output Products**. Pilih options **Global**. Generate Output Product *HARUS* dilakukan setiap kali melakukan perbahan pada block design.

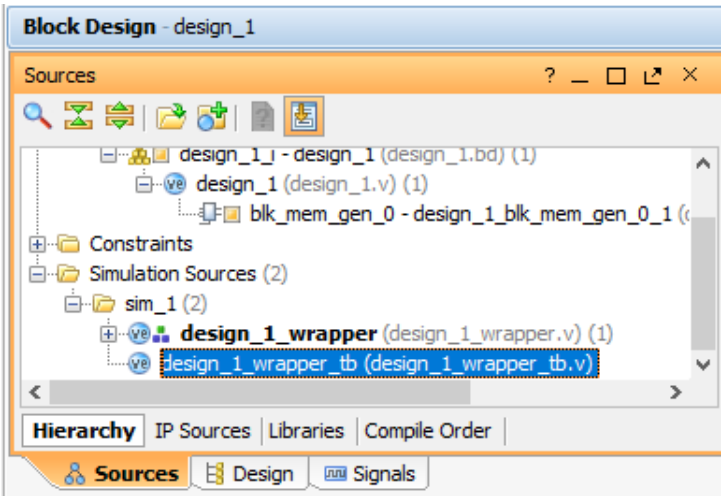




13. Pada **Flow Navigator**, pilih menu **Add Sources**, kemudian pada dialog Add Sources pilih **Add or create simulation sources**. Lagkah ini bertujuan untuk membuat file testbench untuk simulasi design_1_wrapper.v.



14. Click button **Create File**, untuk membuat file testbench dengan nama **design_1_wrapper_tb**. File ini dapat berada pada folder **Simulation Sources**, double-click untuk membuka file.



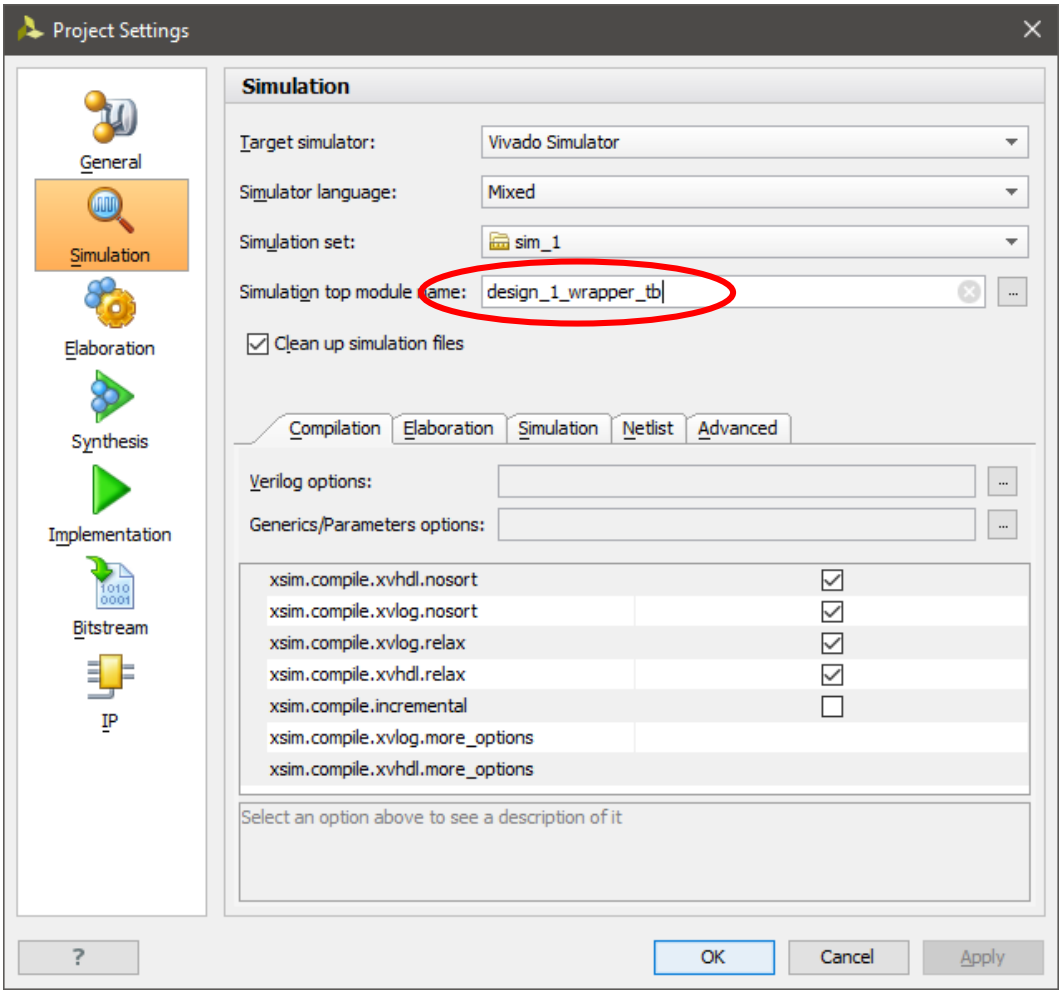
15. Buat testbench sebagai berikut:

```
`timescale 1ns / 1ps

module design_1_wrapper_tb();
    localparam T = 10;
```

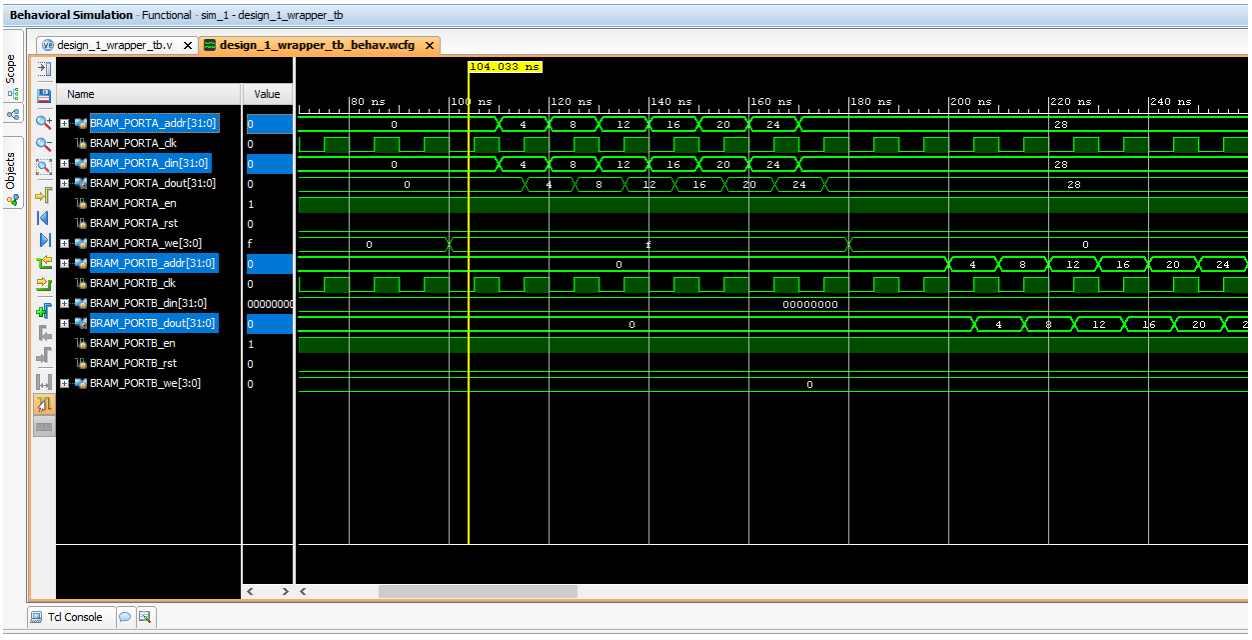
```
    reg [31:0]BRAM_PORTA_addr;
    reg BRAM_PORTA_clk;
    reg [31:0]BRAM_PORTA_din;
    wire [31:0]BRAM_PORTA_dout;
    reg BRAM_PORTA_en;
    reg BRAM_PORTA_rst;
    reg [3:0]BRAM_PORTA_we;
    reg [31:0]BRAM_PORTB_addr;
    reg BRAM_PORTB_clk;
    reg [31:0]BRAM_PORTB_din;
    wire [31:0]BRAM_PORTB_dout;
    reg BRAM_PORTB_en;
    reg BRAM_PORTB_rst;
    reg [3:0]BRAM_PORTB_we;

    integer i;

    design_1_wrapper uut
    (
        .BRAM_PORTA_addr(BRAM_PORTA_addr),
        .BRAM_PORTA_clk(BRAM_PORTA_clk),
        .BRAM_PORTA_din(BRAM_PORTA_din),
        .BRAM_PORTA_dout(BRAM_PORTA_dout),
        .BRAM_PORTA_en(BRAM_PORTA_en),
        .BRAM_PORTA_rst(BRAM_PORTA_rst),
        .BRAM_PORTA_we(BRAM_PORTA_we),
        .BRAM_PORTB_addr(BRAM_PORTB_addr),
        .BRAM_PORTB_clk(BRAM_PORTB_clk),
        .BRAM_PORTB_din(BRAM_PORTB_din),
        .BRAM_PORTB_dout(BRAM_PORTB_dout),
        .BRAM_PORTB_en(BRAM_PORTB_en),
        .BRAM_PORTB_rst(BRAM_PORTB_rst),
        .BRAM_PORTB_we(BRAM_PORTB_we)
    );

    always
    begin
        // Clock
        BRAM_PORTA_clk = 0;
        BRAM_PORTB_clk = 0;
        #(T/2);
        BRAM_PORTA_clk = 1;
        BRAM_PORTB_clk = 1;
        #(T/2);
    end

    initial
    begin
        // Init
        BRAM_PORTA_addr = 0;
        BRAM_PORTA_din = 0;
        BRAM_PORTA_en = 1;
        BRAM_PORTA_we = 0;
        BRAM_PORTB_addr = 0;
        BRAM_PORTB_din = 0;
        BRAM_PORTB_en = 1;
        BRAM_PORTB_we = 0;
```

```
        // Reset
        BRAM_PORTA_rst = 1;
        BRAM_PORTB_rst = 1;
        #(T*5);
        BRAM_PORTA_rst = 0;
        BRAM_PORTB_rst = 0;
        #(T*5);

        // Write port A
        BRAM_PORTA_we = 4'hf;
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTA_addr = i;
            BRAM_PORTA_din = i;
            #T;
        end
        BRAM_PORTA_we = 0;
        #T;

        // Read port B
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTB_addr = i;
            #T;
        end
        #T;
    end

endmodule
```

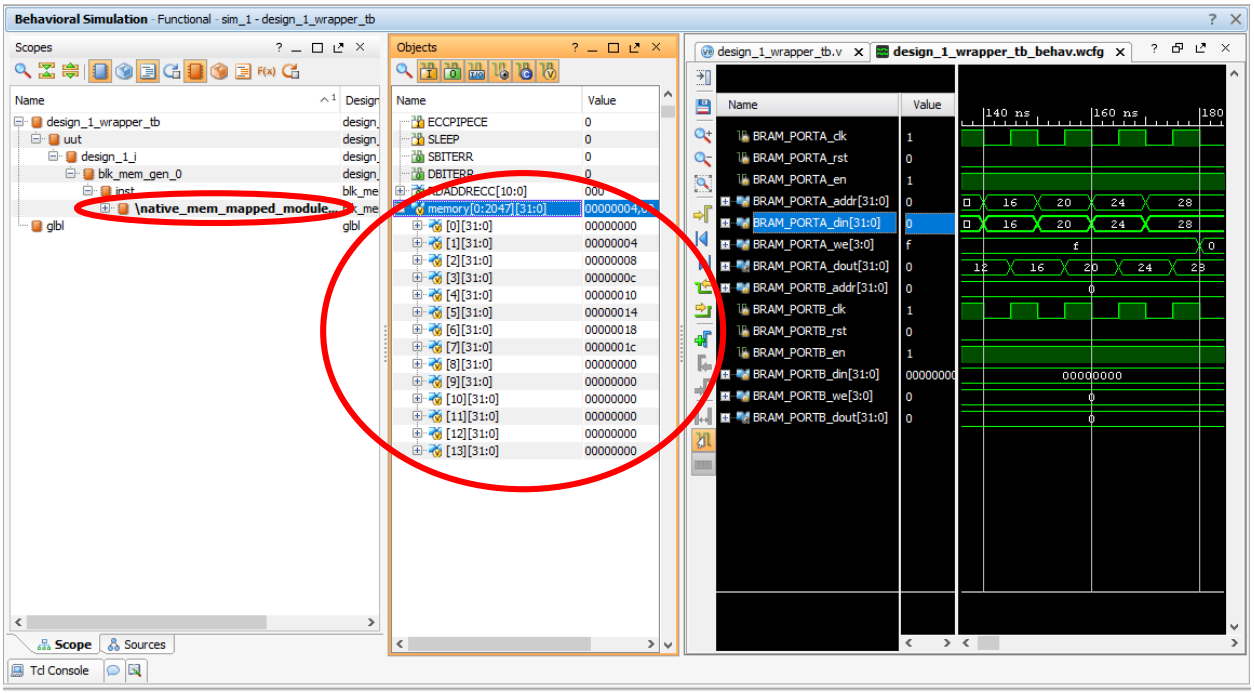
16. Pada **Flow Navigator**, pilih menu **Simulation Settings**, pastikan file testbench yang digunakan yaitu **design_1_wrapper_tb.**



17. Pada **Flow Navigator**, pilih menu **Run Simulation** untuk menjalankan simulasi.



18. Melihat isi block memory pada simulasi yaitu pada **Scopes**, pilih **\native_mem_mapped_module** kemudian cari **Objects** yang bernama **memory**.



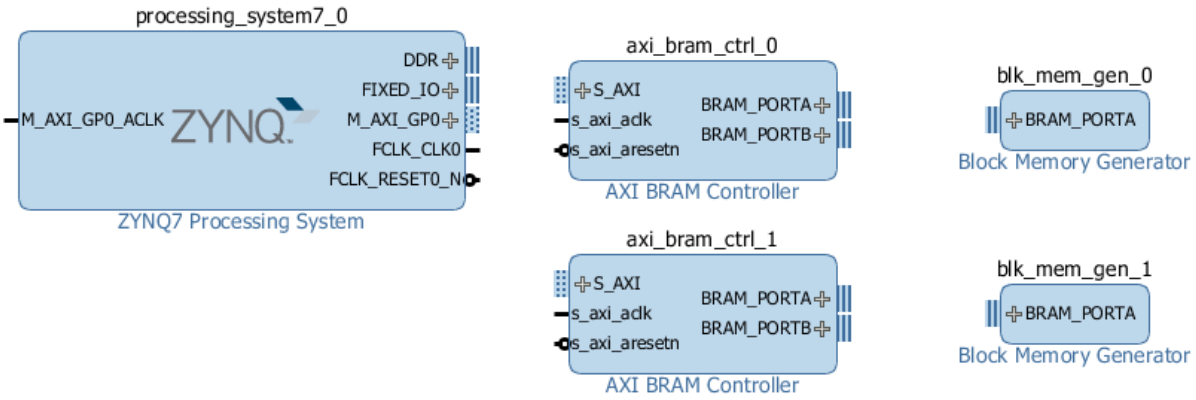
II. Implementasi Block Memory

Pada tutorial ini akan dilakukan integrasi antara IP Block memory Generator dengan ZYNQ7000. Integrasi ini membutuhkan IP AXI BRAM Controller yang berfungsi sebagai interface AXI4 dari Block Memory ke processor ARM.

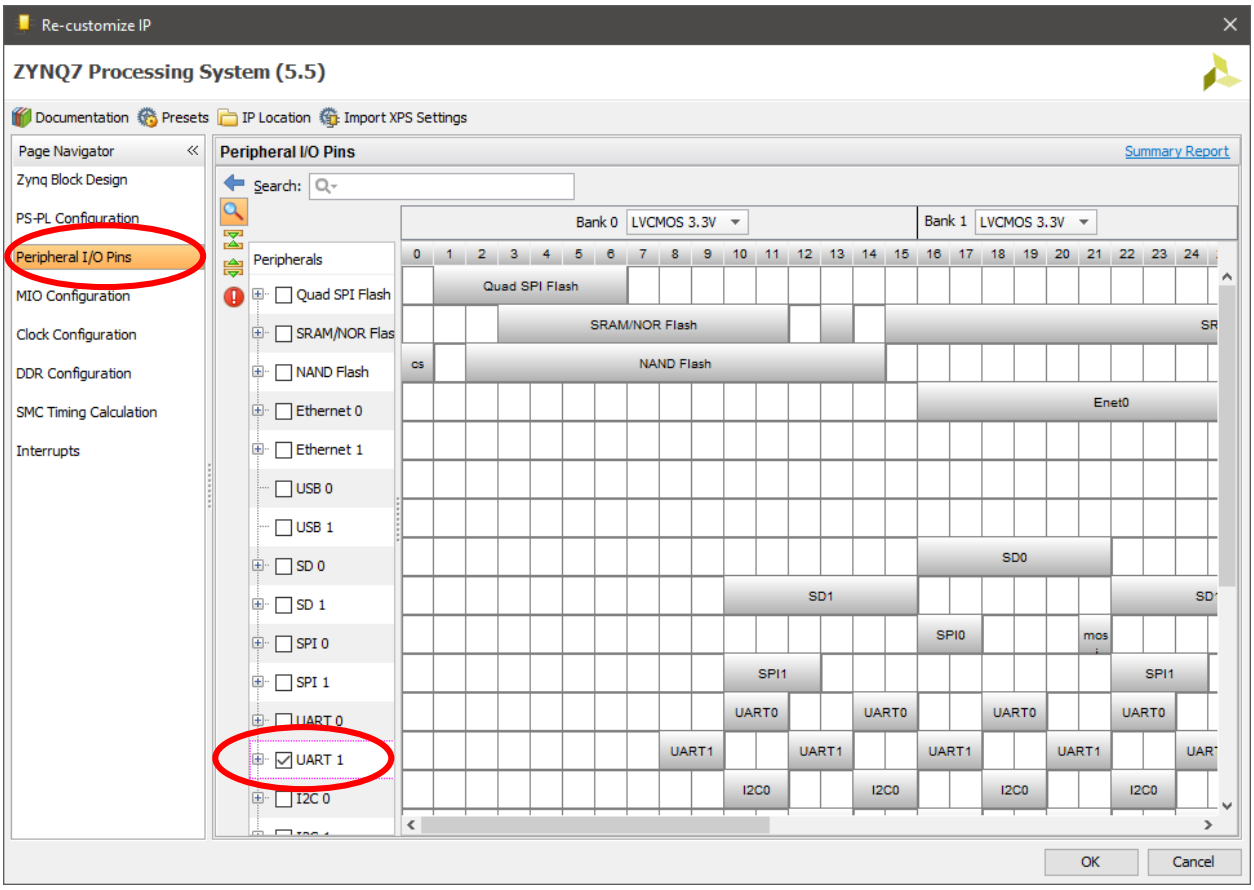
1. Pada project yang sama, buatlah block design baru **design_2**, dari menu **Flow Navigator**, **Create Block Design**.

2. Tambahkan IP pada block design:

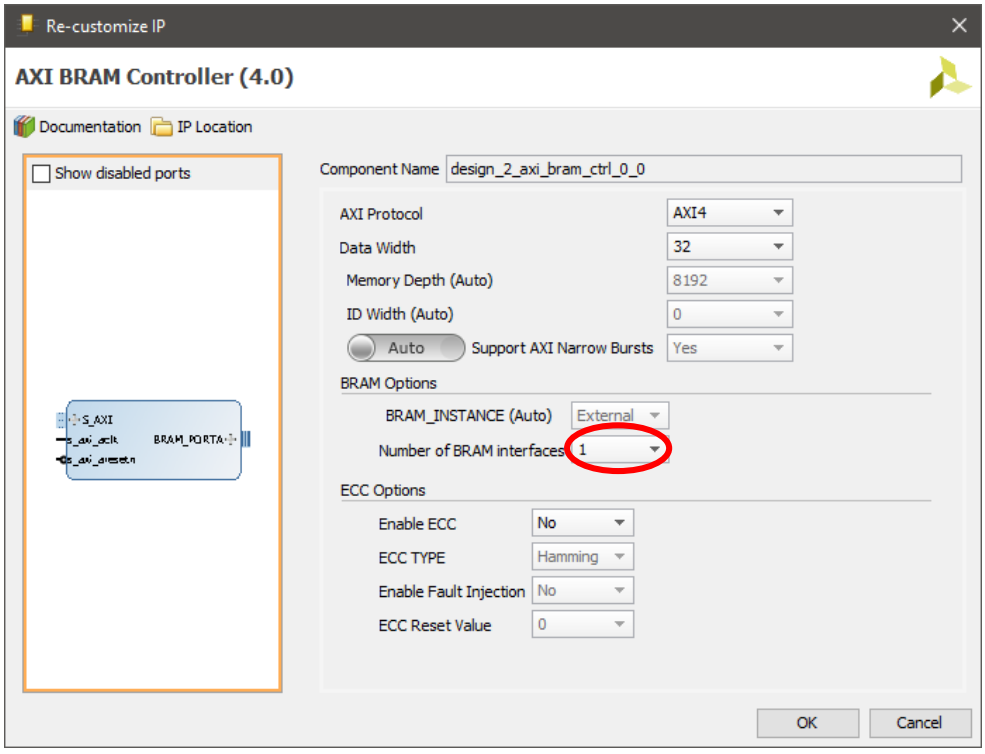
- **1x ZYNQ7 Processing System**
- **2x AXI BRAM Controller**
- **2x Block Memory Generator**



3. **Double-click** block **ZYNQ7 Processing System** untuk melakukan konfigurasi. Pada tutorial ini dibutuhkan UART1 untuk debug data pada memory. Aktifkan UART1 pada **Peripheral I/O Pins.**

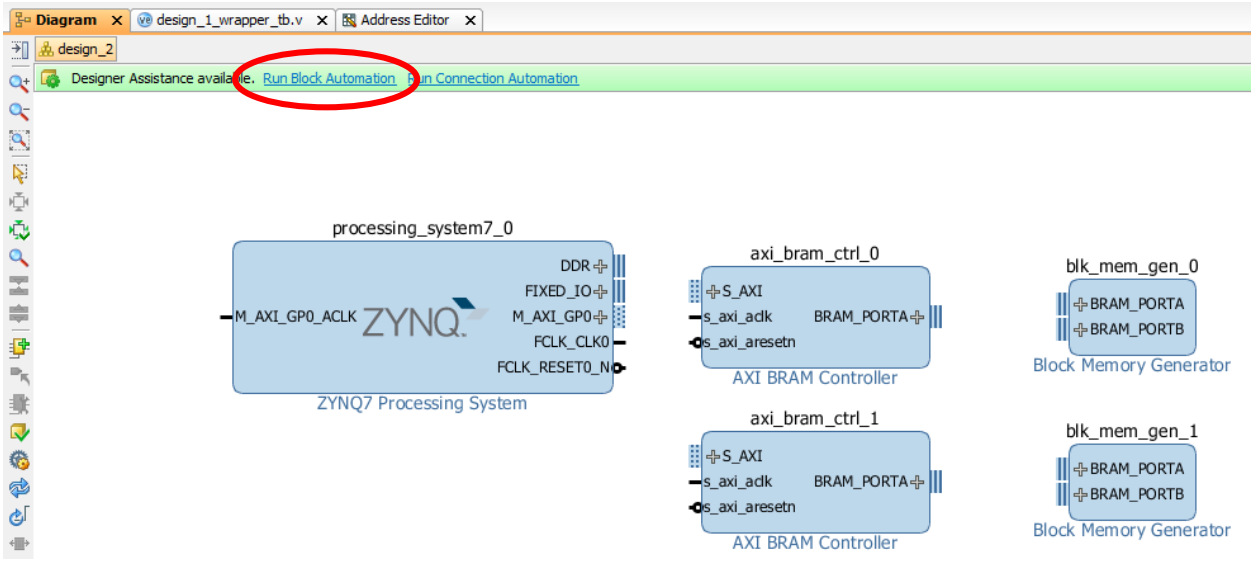


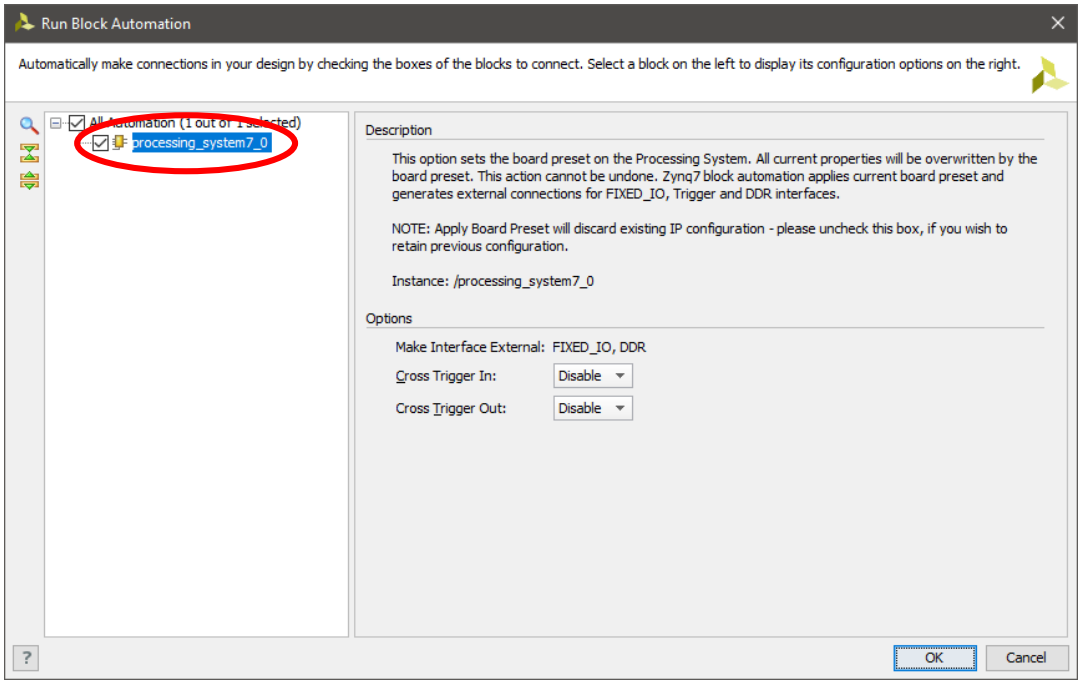
4. **Double-click** kedua block **BRAM Controller**, untuk mengubah interface menjadi 1 port.



5. **Double-click** kedua block **Block Memory Generator**, kemudian ubah **Memory Type** menjadi **True Dual Port RAM**.

6. Click **Run Block Automation** untuk membuat port yang menghubungkan ZYNQ7 dengan memory DDR dan FIXED IO.





7. Click **Run Connection Automation** untuk menghubungkan ZYNQ7 dengan AXI BRAM Controller melalui bus AXI4.

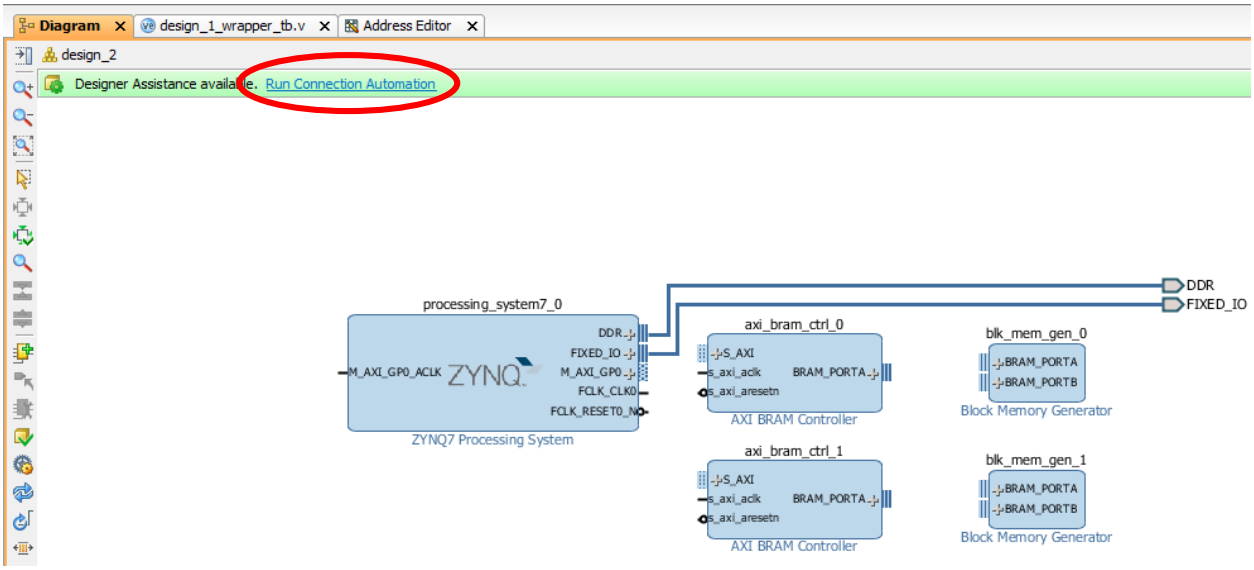
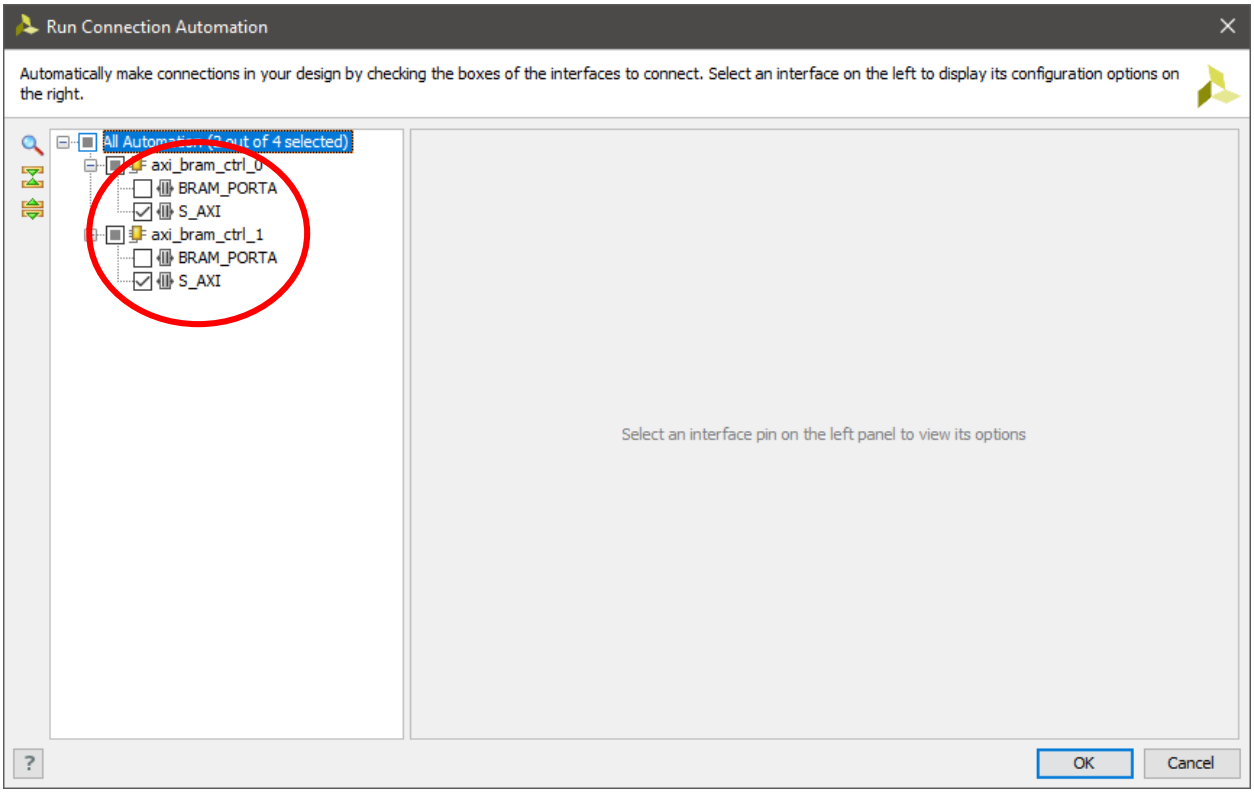
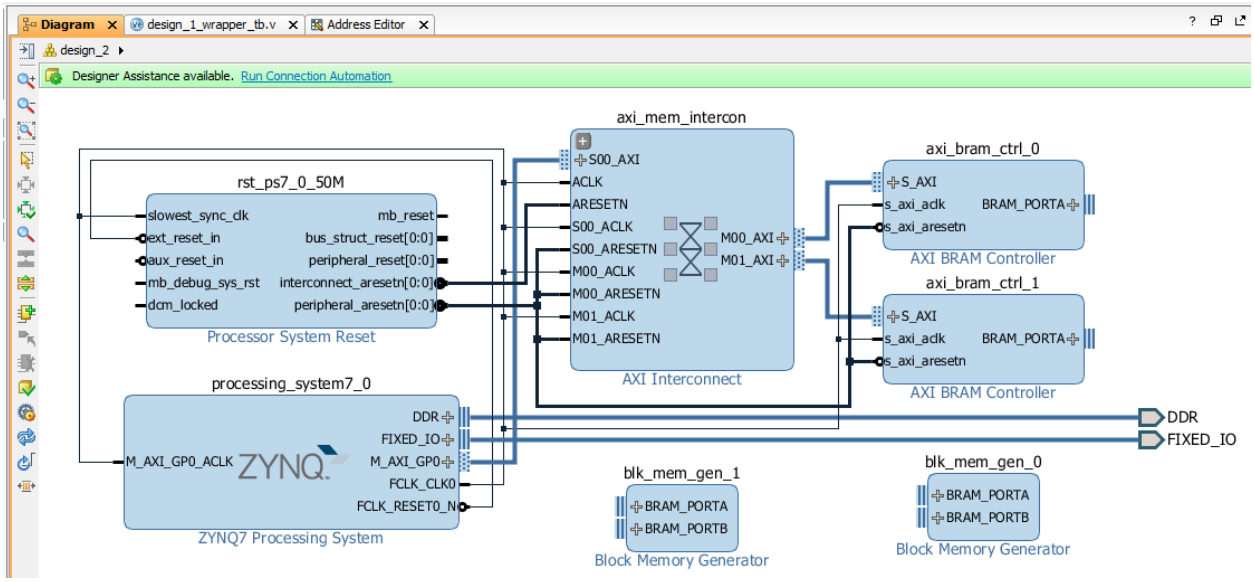
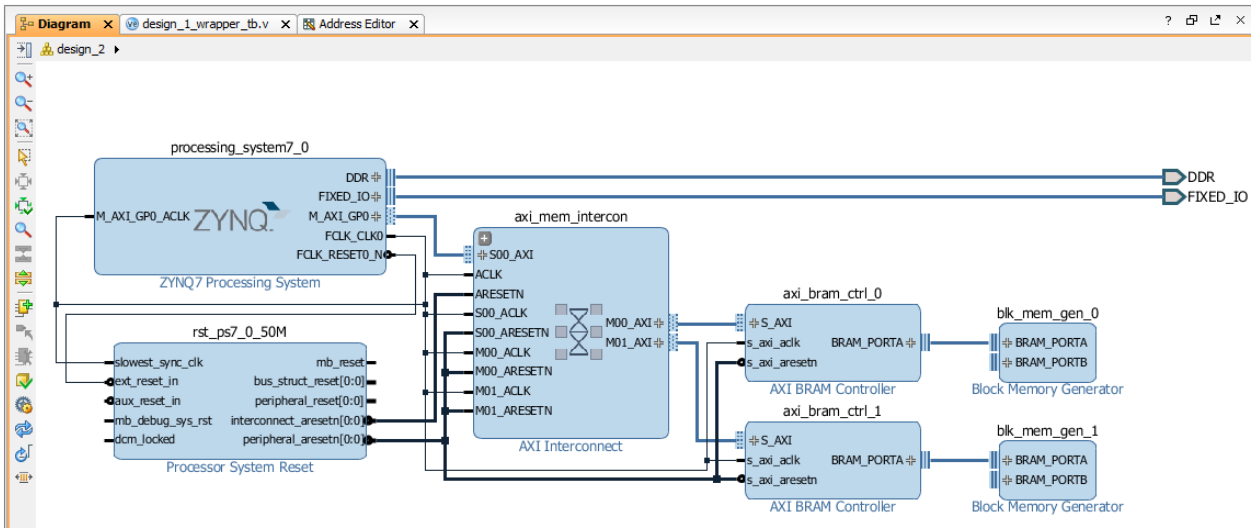


Diagram
design_1_wrapper_tb.v
Address Editor
design_2
Designer Assistance available. Run Connection Automation
processing_system7_0
DDR
FIXED_IO
M_AXI_GP0_ACLK
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
axi_bram_ctrl_0
S_AXI
s_axi_aclk
s_axi_aresetn
BRAM_PORTA
AXI BRAM Controller
blk_mem_gen_0
BRAM_PORTA
BRAM_PORTB
Block Memory Generator
axi_bram_ctrl_1
AXI BRAM Controller
blk_mem_gen_1
Block Memory Generator

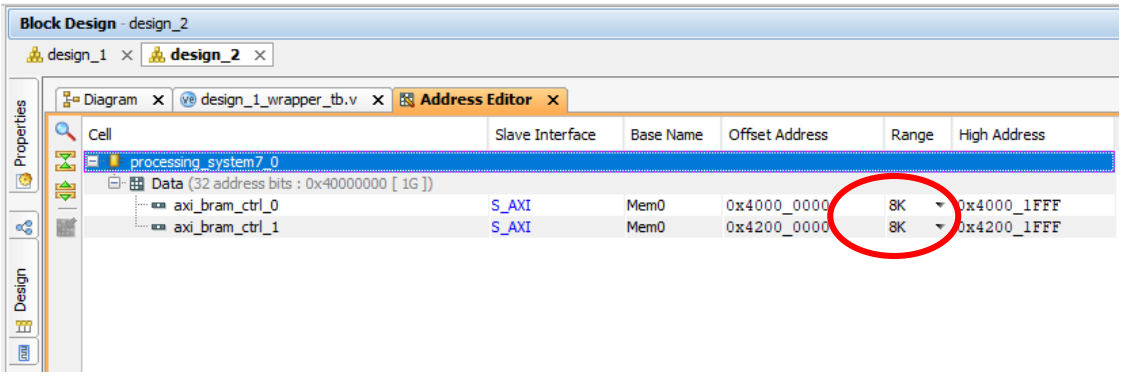

Run Connection Automation
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.
All Automation (3 out of 4 selected)
axi_bram_ctrl_0
BRAM_PORTA
S_AXI
axi_bram_ctrl_1
BRAM_PORTA
S_AXI
Select an interface pin on the left panel to view its options
OK
Cancel

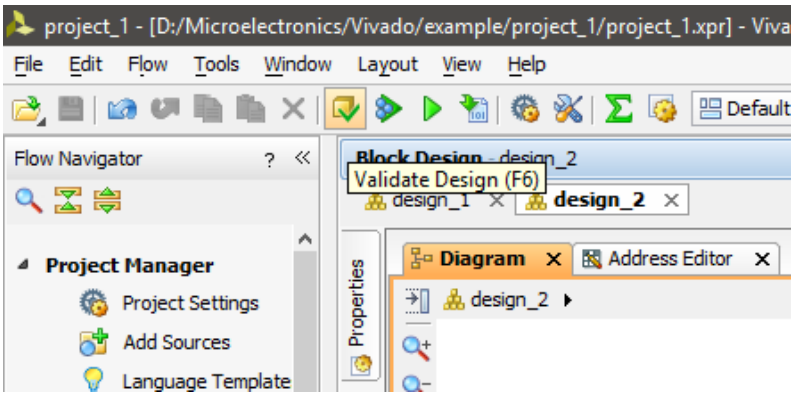


8. Hubungkan **BRAM_PORTA** dari **Block Memory Generator** ke **AXI BRAM Controller** seperti pada gambar di bawah ini:



9. Ukuran memory dapat diubah pada tab **Address Editor** dengan mengubah **range**. Lakukan **Validate Design** setelah merubah range.



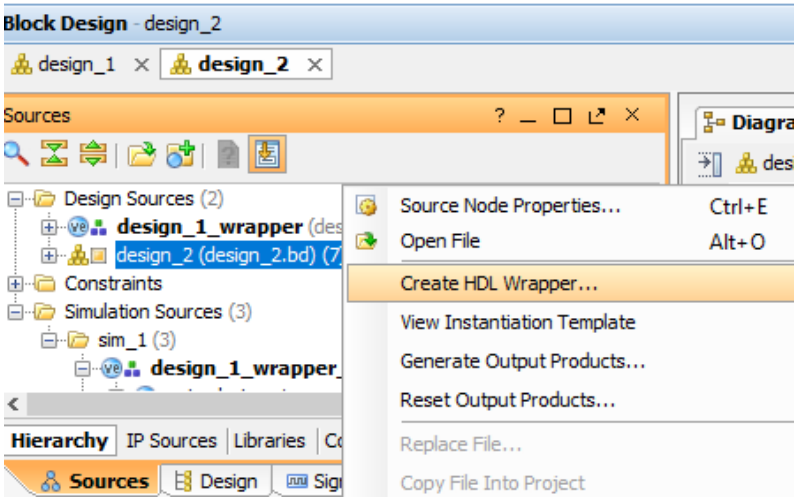


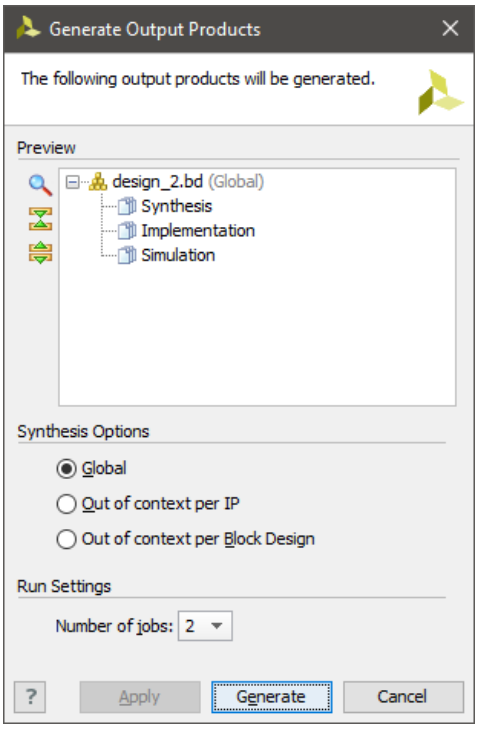
Range akan menentukan size/depth dari BRAM seperti berikut ini:

- Range 4K = Depth 2048 x 32 bit*
- Range 8K = Depth 2048 x 32 bit
- Range 16K = Depth 4096 x 32 bit
- Range 32K = Depth 8192 x 32 bit
- Range 128K = Depth 32768 x 32 bit

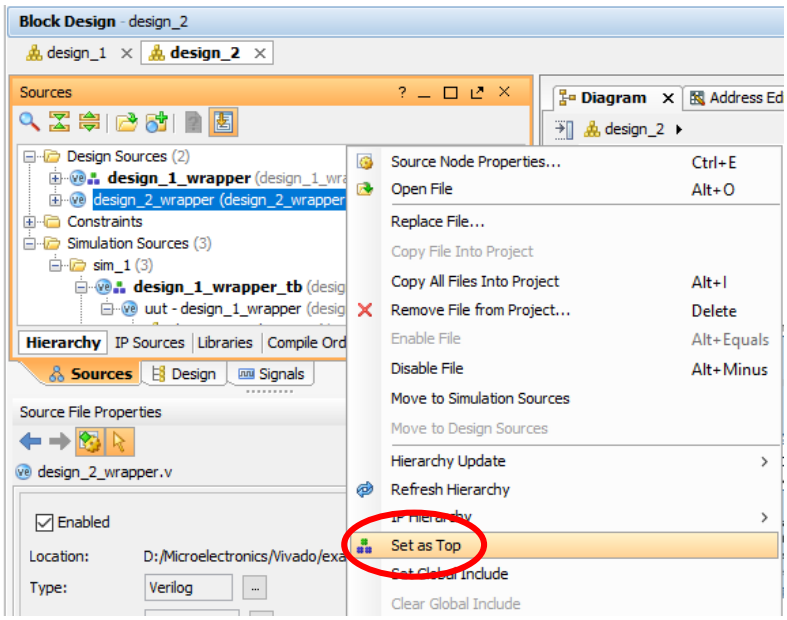
***range 4K menghasilkan depth yang sama dengan 8K yaitu 2048. Not sure why.**

10. Buat **HDL wrapper** dan **generate output product** dari **design_2**.



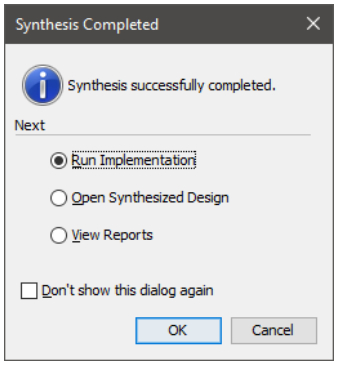


11. Sebelum melakukan Synthesis, **design_2_wrapper** harus diset mejadi top module yang akan disintesis dengan cara **right-click**, kemudian pilih **Set as Top**.

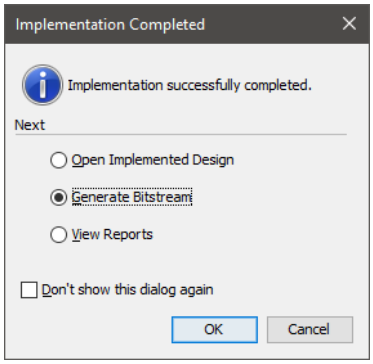


12. Pada **Flow Navigator**, pilih menu **Run Synthesis**.

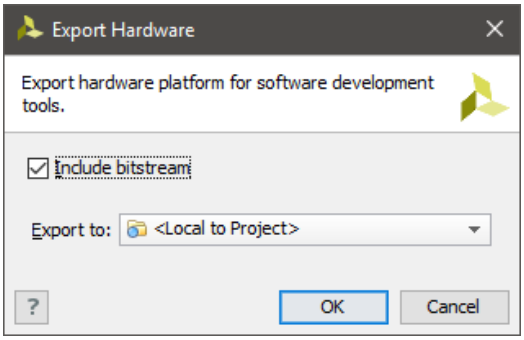
13. Setelah synthesis selesai, pilih **Run Implementation**.



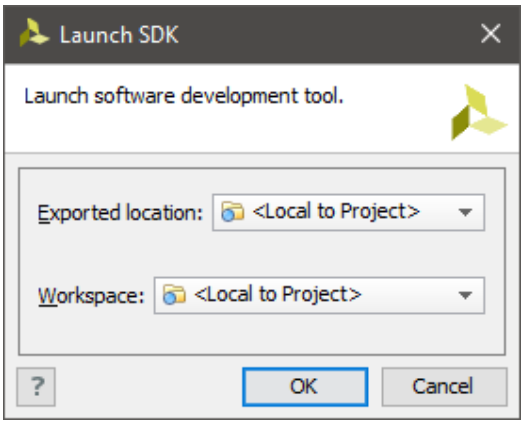
14. Setelah implementation selesai, pilih **Generate Bitstream**.



15. Export file bitstream dengan cara pilih menu **File→Export→Export Hardware**, kemudian aktifkan **Include bitstream**.

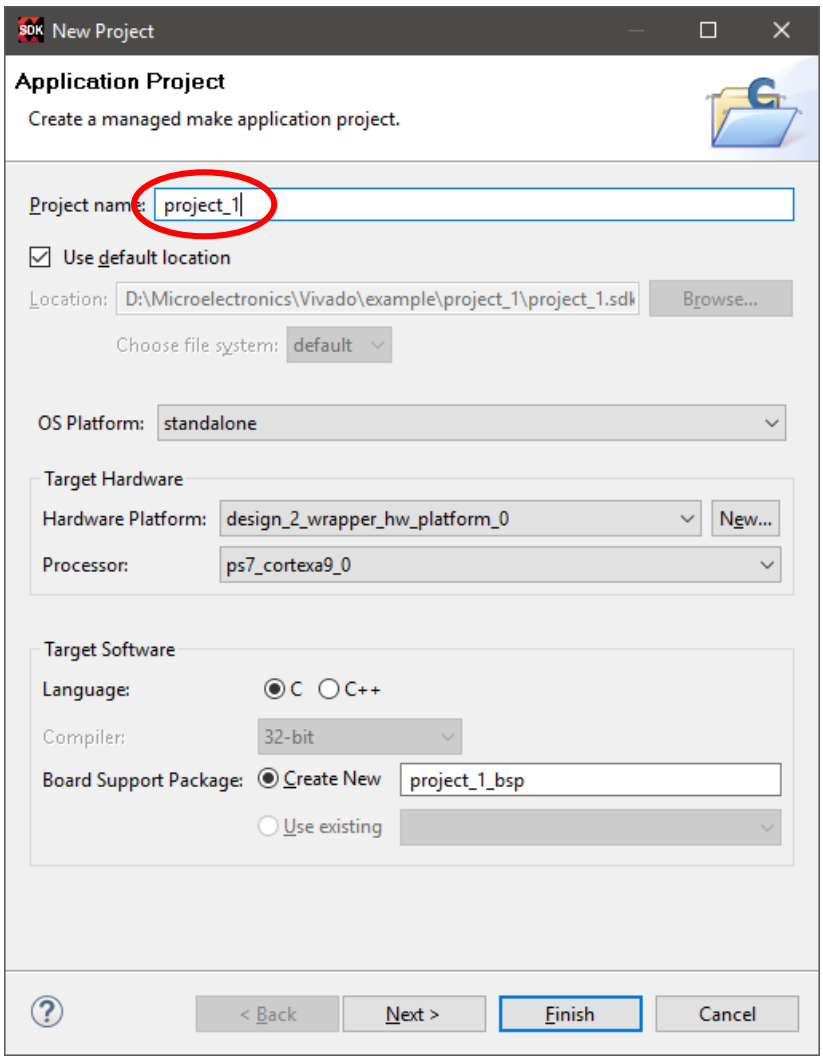


16. Pilih menu **File→Launch SDK.**

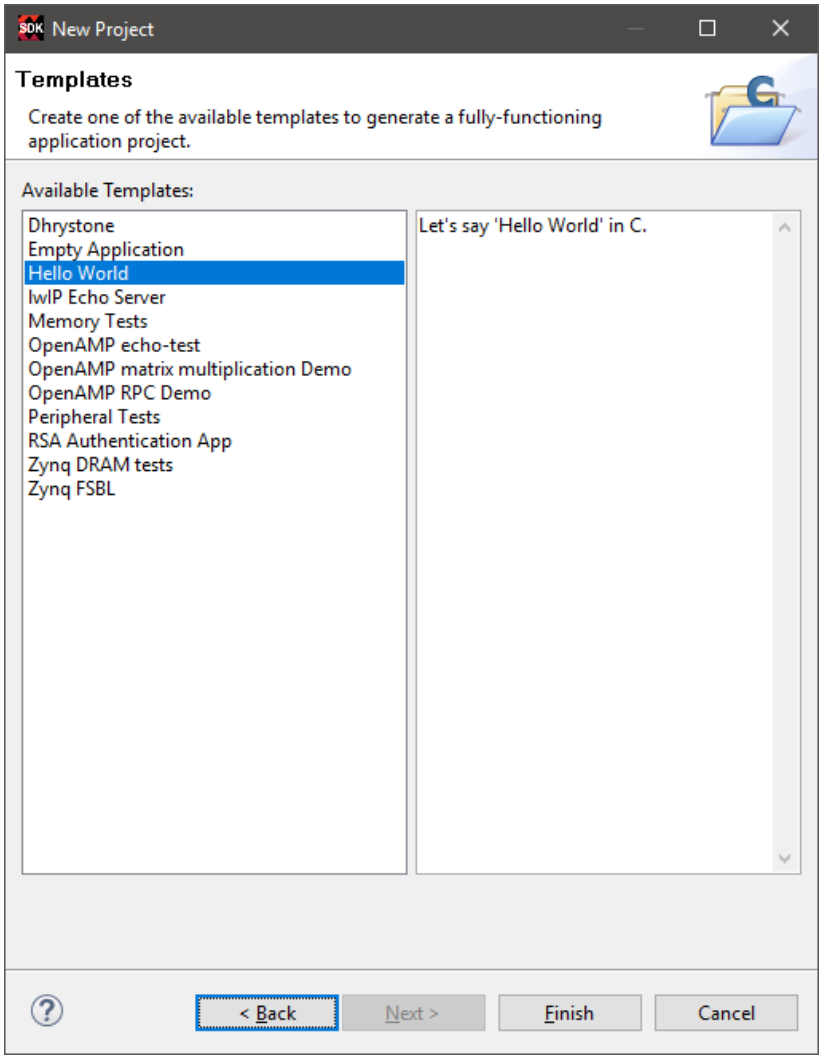


17. Pada SDK pilih menu **File→New→Application Project.**

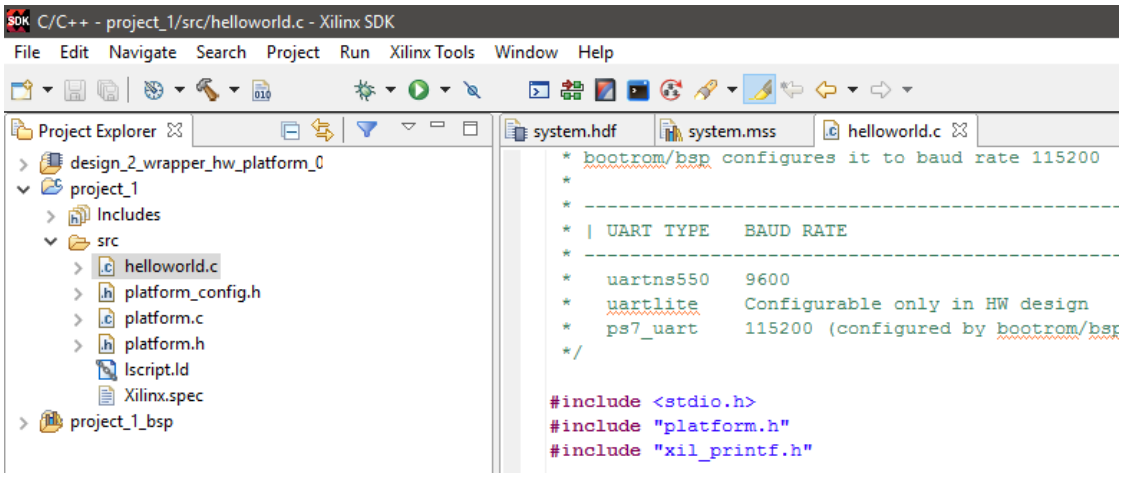
18. Masukan project name.



19. Click **Next**, pilih template **Hello World**, kemudian **Finish**.



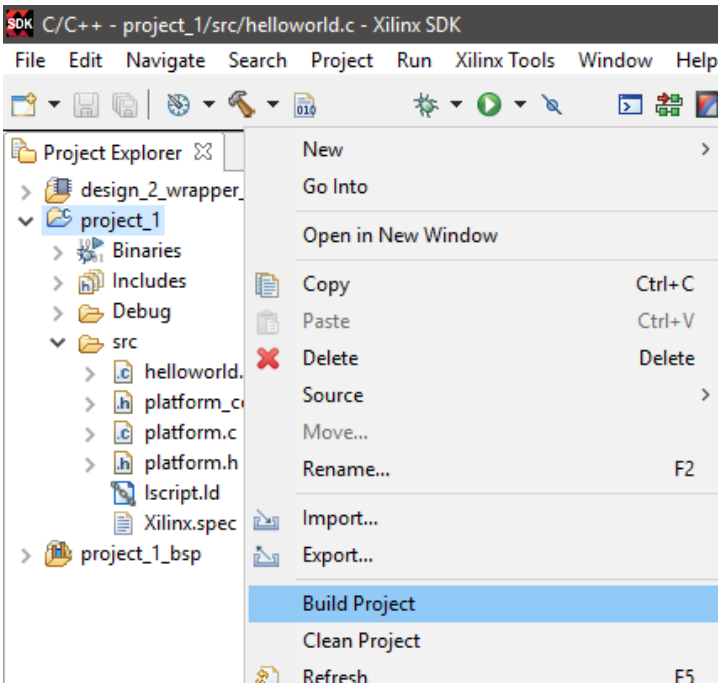
20. Buka file **helloworld.c**, kemudian buat program untuk membaca memory.



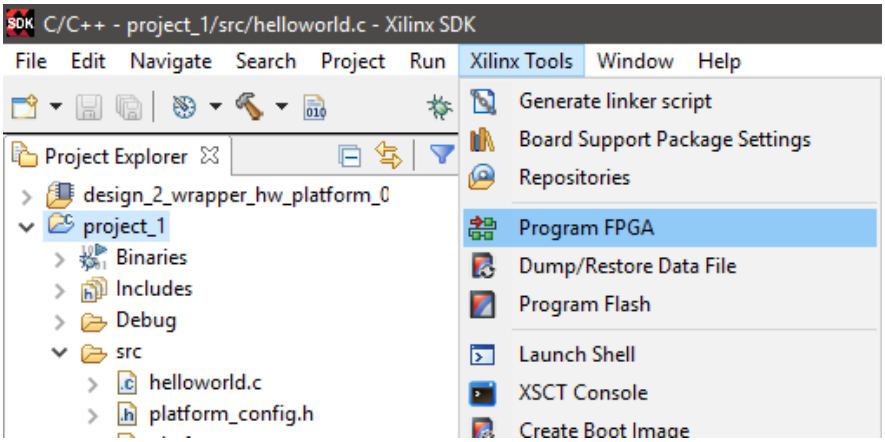

```
#include <stdio.h>
#include <stdint.h>
#include <sleep.h>
```

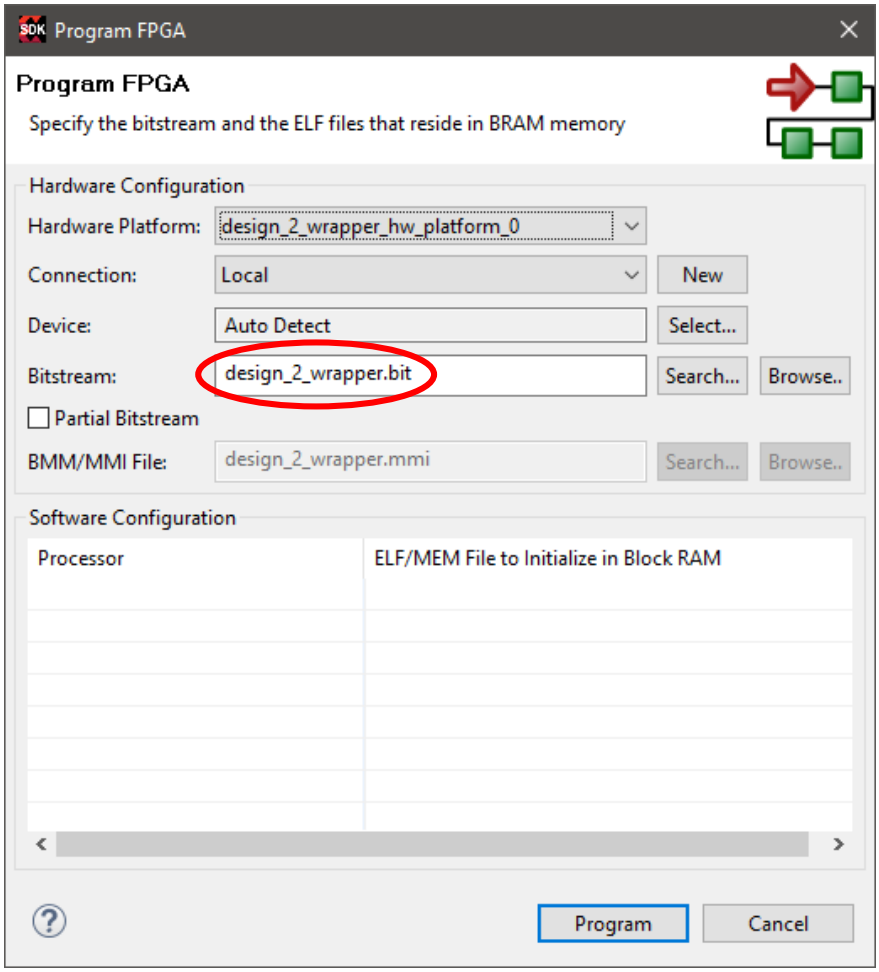
```c
#define MEM_INP_BASE 0x40000000 // Sesuaikan dengan address editor di Vivado
#define MEM_OUT_BASE 0x42000000 // Sesuaikan dengan address editor di Vivado

uint32_t *meminp_p, *memout_p;

int main()
{
    while (1)
    {
        // *** Initialize pointer ***
        meminp_p = (uint32_t *)MEM_INP_BASE;
        memout_p = (uint32_t *)MEM_OUT_BASE;

        // *** Write to block memory input ***
        for (int i = 0; i <= 15; i++)
            *(meminp_p+i) = i;

        // *** Read from block memory input ***
        printf("Block Memory Input: ");
        for (int i = 0; i <= 15; i++)
            printf("%d, ", (unsigned int)*(meminp_p+i));

        // *** Write to block memory output ***
        for (int i = 0; i <= 15; i++)
            *(memout_p+i) = 15-i;

        // *** Read from block memory output ***
        printf("\nBlock Memory Output: ");
        for (int i = 0; i <= 15; i++)
            printf("%d, ", (unsigned int)*(memout_p+i));
        printf("\n\n");

        sleep(1);
    }

    return 0;
}
```

21. Build project dengan cara **right-click** pada project folder, kemudian pilih **Build Project**.

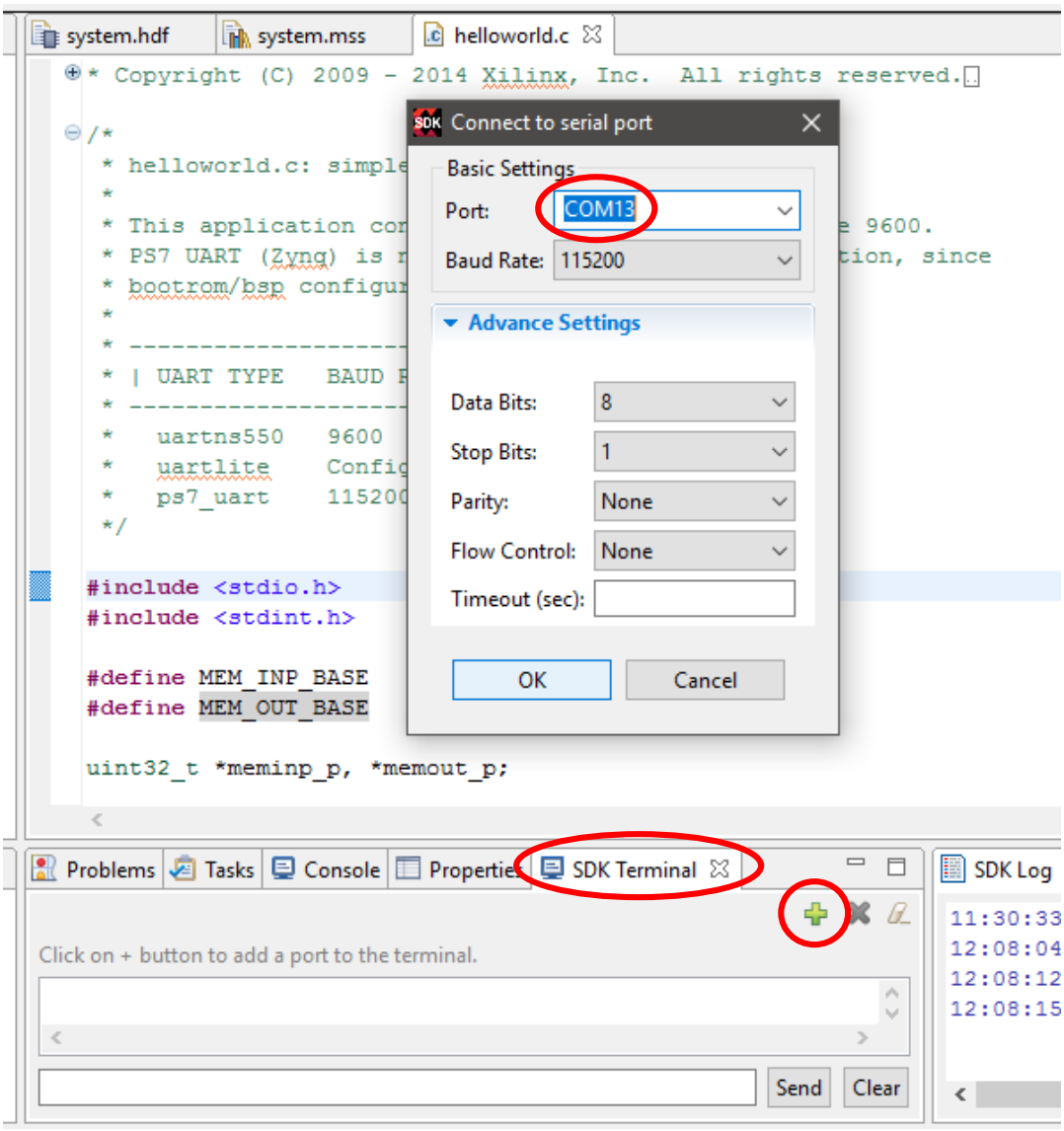


22. Program FPGA dengan cara pilih menu **Xilinx Tools→Program FPGA**.

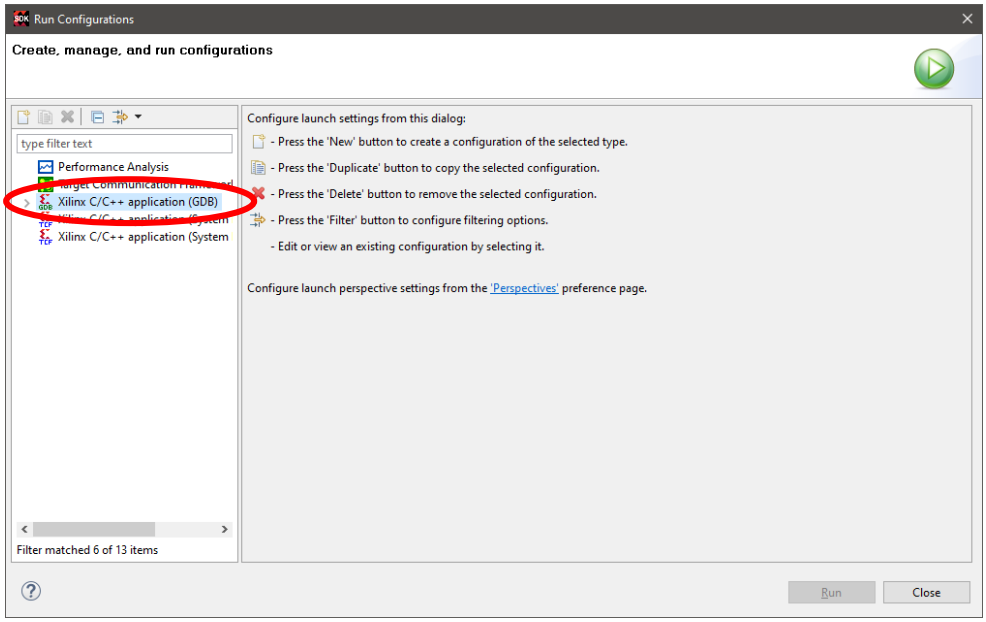




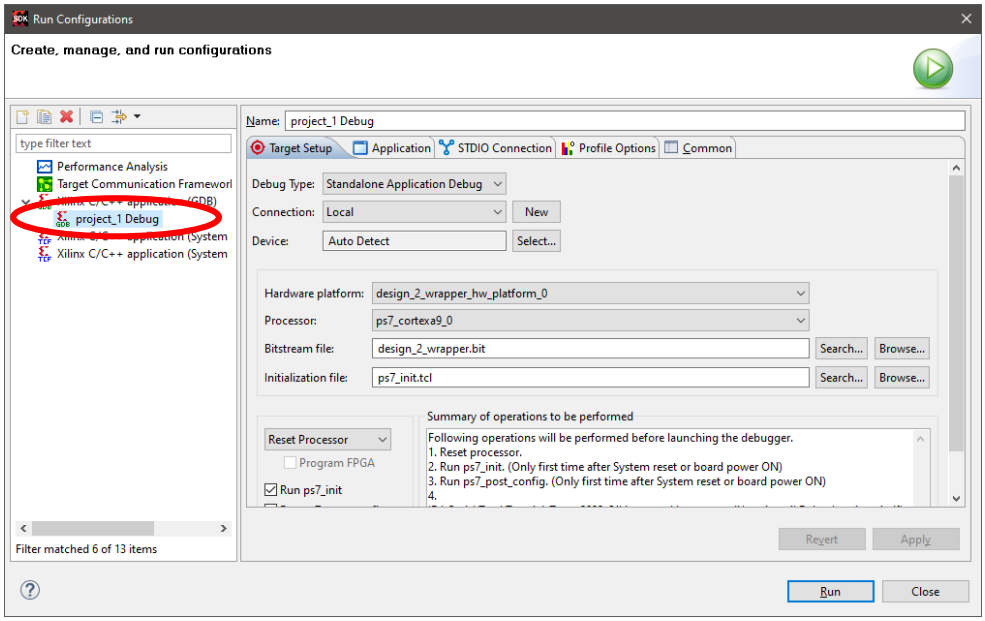
23. Buka **SDK terminal** untuk melihat log akses memory, pilih **COM Port** yang sesuai.



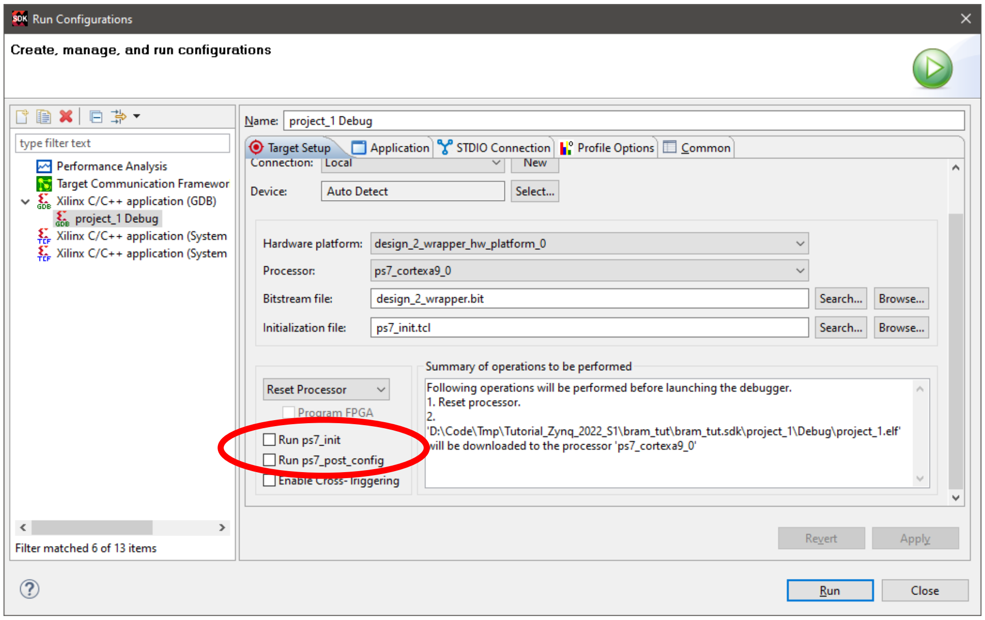
24. Pilih menu **Run, Run configurations.** Double click **Xilinx C/C++ application (GDB).**



25. Setelah double click, maka akan otomatis terbuat setting bernama **project_1 Debug**.

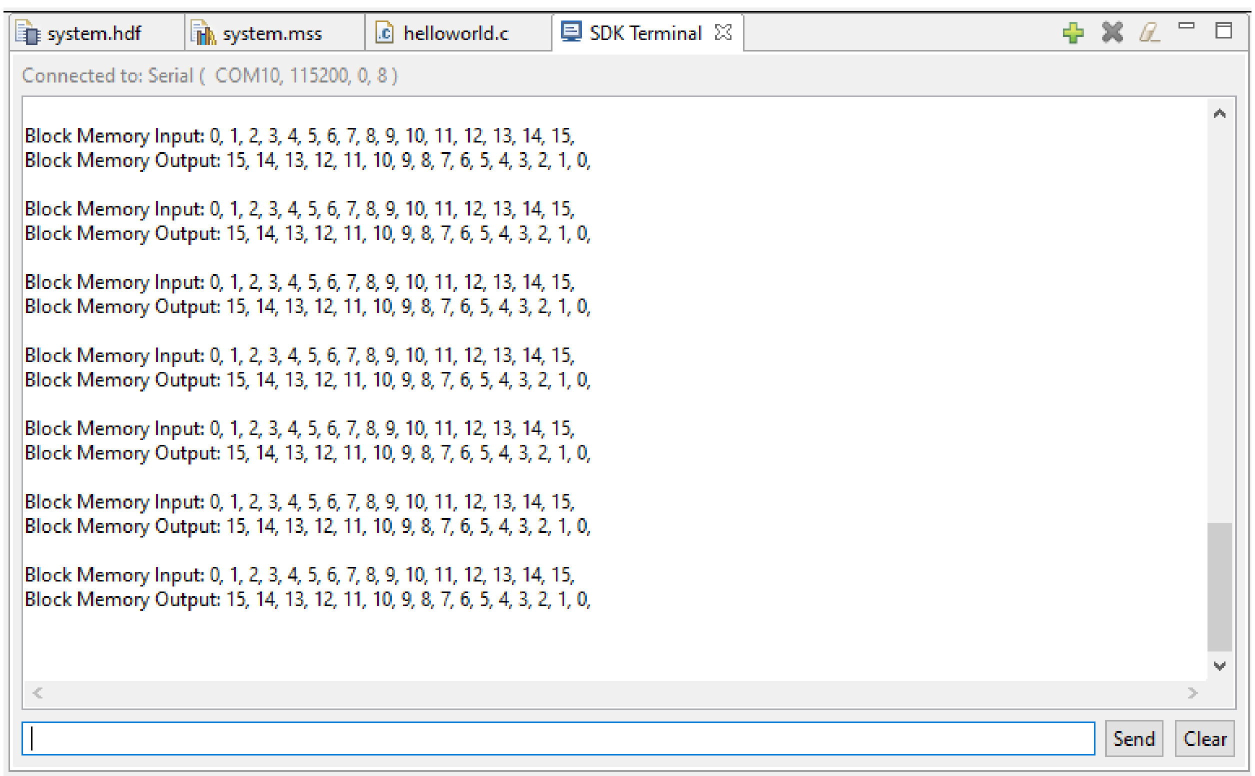


26. Atur konfigurasi dengan melakukan **un-check Run ps7_init** dan **Run ps7_post_config**, kemudian **Apply**.

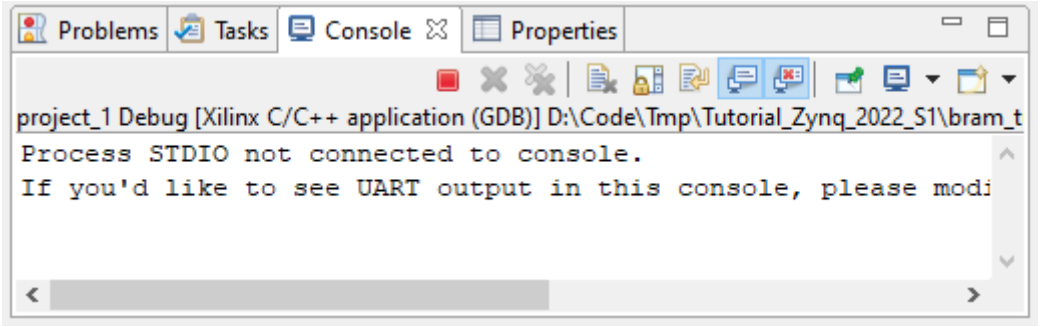


27. Run program dengan klik tombol **Run**.

28. Pada **SDK terminal**, data pada memory akan ditampilkan.

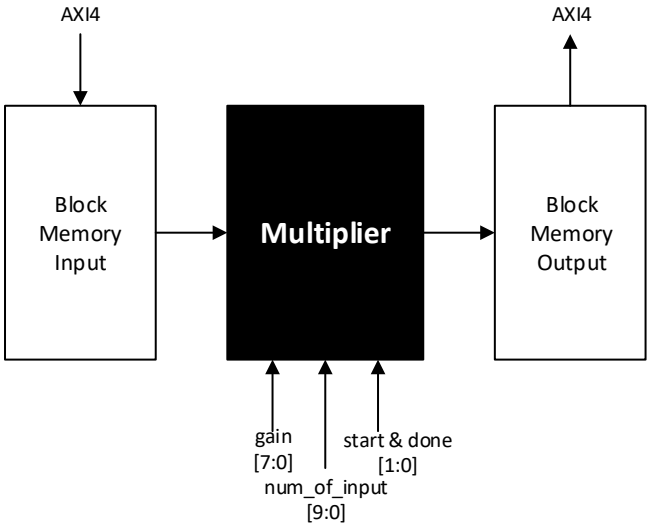


29. Gunakan tombol **stop** untuk menghentikan program sebelum mengupload program baru.
Catatan: tombol stop akan otomatis berhenti (tidak bisa di-klik) jika pada progam C tidak membuat loop while(1).



III. Pembuatan IP Core Multiplier

Pada bagian ini akan dibuat IP core multiplier. IP ini dapat membaca data input dari memory input, kemudian menulis hasil perkaliannya ke memori output. Block diagram IP multiplier ditunjukan pada gambar berikut ini.

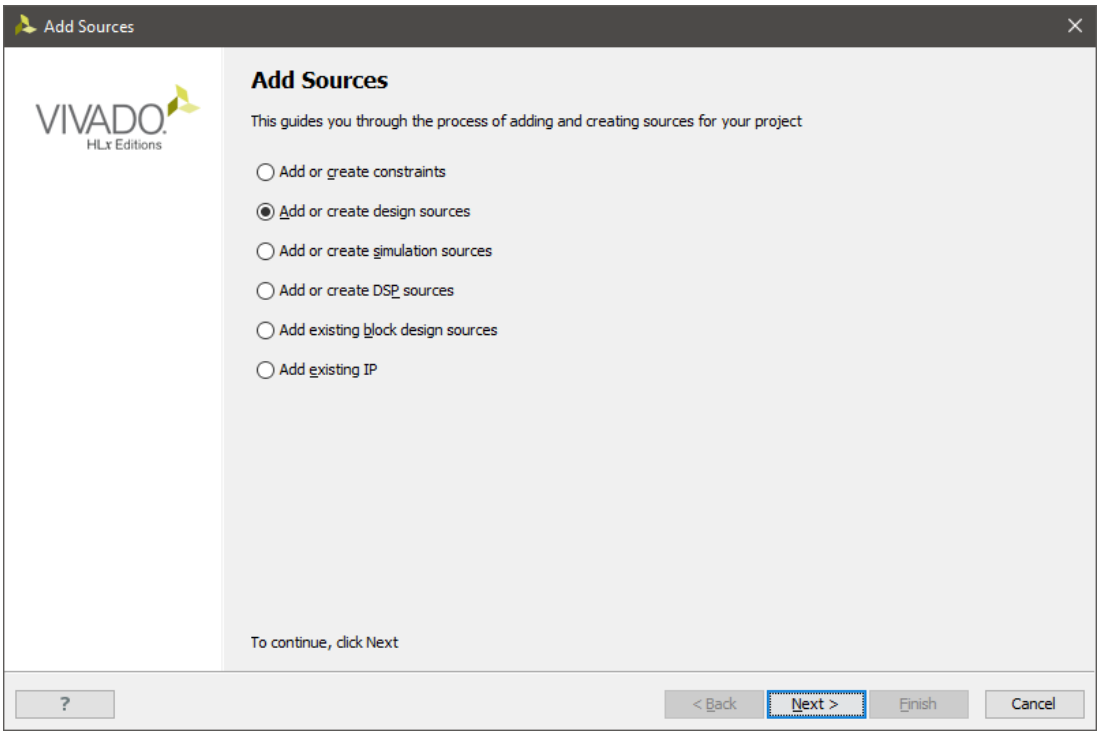


Input dan output data terhubung ke memory input dan output. Port gain berfungsi sebagai konstanta pengali dari multiplier tersebut, sehingga

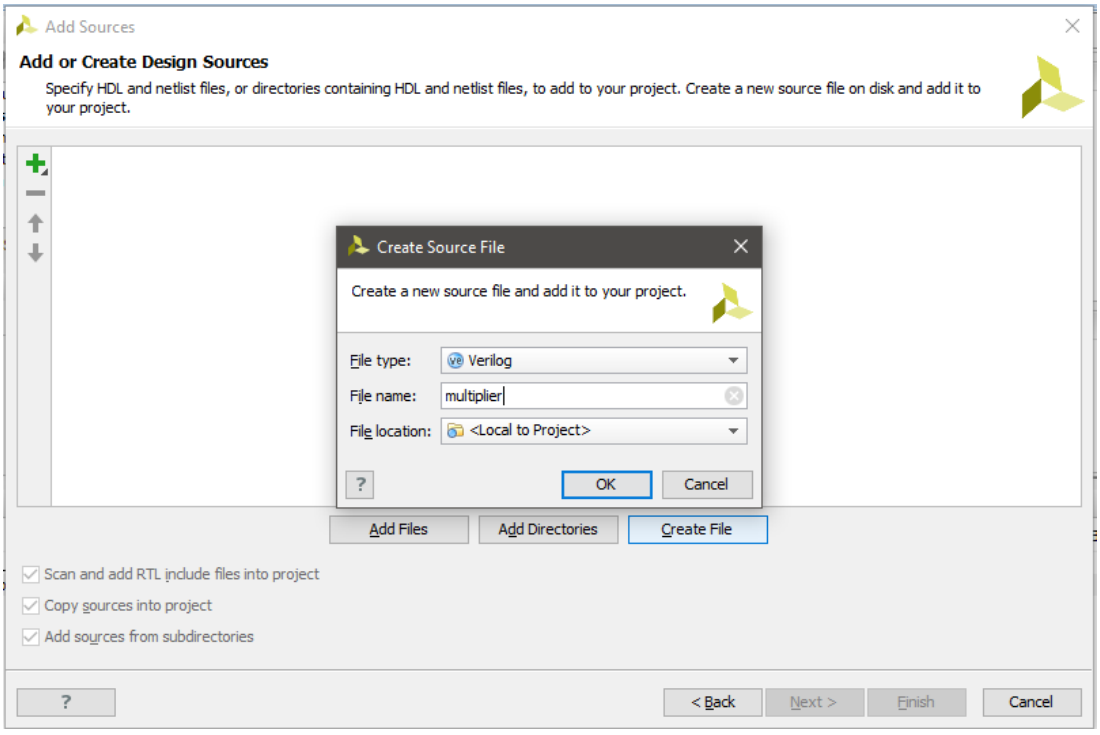
$$output(n) = input(n) * gain$$

Number of input merupakan jumlah data input yang akan diproses oleh multiplier. Sinyal start berfungsi untuk memulai proses perkalian oleh IP multiplier. Sinyal done berfungsi untuk memberi tahu processor bahwa proses multiplication telah selesai dan output sudah berada di memory output.

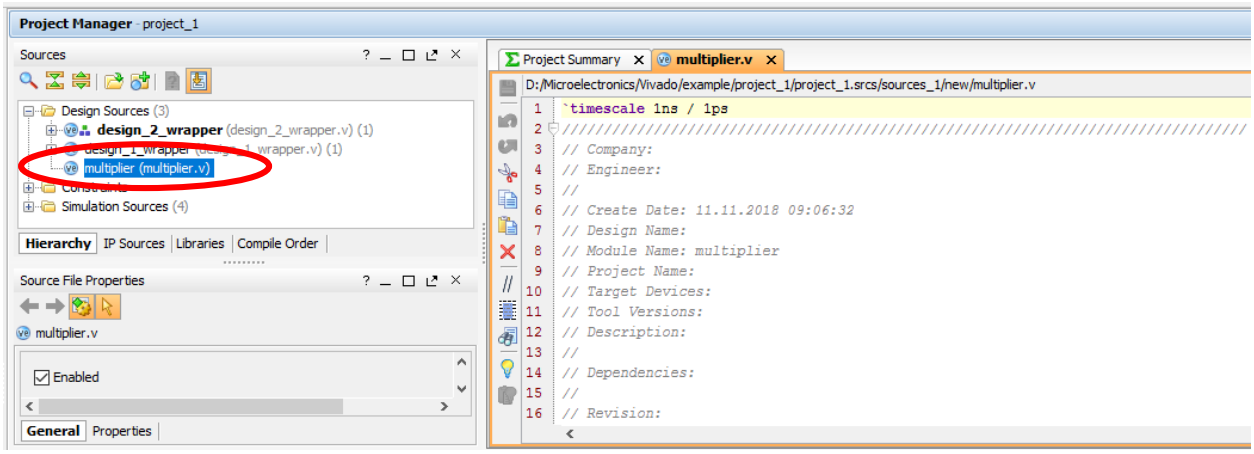
1. Pada **Flow Navigator**, pilih menu **Add Sources**, kemudian pilih **Add or create design sources**.



2. Buat file Verilog baru dengan click button **Create File**, kemudian beri nama file **multiplier**.



3. Buka file **multiplier.v** kemudian tambahkan code Verilog berikut ini.



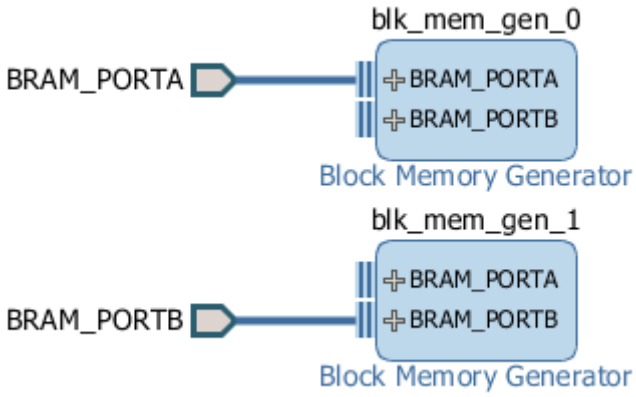

```
`timescale 1ns / 1ps

module multiplier
    (
        // *** Clock and reset ***
        input wire clk,
        input wire rst_n,
        output wire rst,
        // *** RAM input ***
        output wire [31:0] rd_addr,
        input wire [31:0] rd_data,
        // *** RAM output ***
        output wire [31:0] wr_addr,
        output wire [31:0] wr_data,
        output wire [3:0] wr_en,
        // *** Control signal ***
        input wire [9:0] num_of_inp,
        input wire [7:0] gain,
        input wire start,
        output wire done
    );

    localparam [1:0] S_IDLE = 4'h0,
                     S_READ = 4'h1,
                     S_WRITE = 4'h2,
                     S_DONE = 4'h3;

    reg [1:0] _cs, _ns;
    reg [31:0] rd_addr_cv, rd_addr_nv;
    reg [31:0] wr_addr_cv, wr_addr_nv;
    reg [31:0] wr_data_cv, wr_data_nv;
    reg [3:0] wr_en_cv, wr_en_nv;
    reg done_cv, done_nv;

    // *** Register ***
    always @(posedge clk)
    begin
        if (!rst_n)
        begin
            _cs <= S_IDLE;
            rd_addr_cv <= 0;
```

```
            wr_addr_cv <= 0;
            wr_data_cv <= 0;
            wr_en_cv <= 0;
            done_cv <= 0;
        end
        else
        begin
            _cs <= _ns;
            rd_addr_cv <= rd_addr_nv;
            wr_addr_cv <= wr_addr_nv;
            wr_data_cv <= wr_data_nv;
            wr_en_cv <= wr_en_nv;
            done_cv <= done_nv;
        end
    end

    // *** Next state and data path ***
    always @(*)
    begin
        _ns = _cs;
        rd_addr_nv = rd_addr_cv;
        wr_addr_nv = wr_addr_cv;
        wr_data_nv = wr_data_cv;
        wr_en_nv = wr_en_cv;
        done_nv = done_cv;
        case (_cs)
            S_IDLE:
            begin
                if (start)
                begin
                    if (num_of_inp == 0)
                    begin
                        _ns = S_DONE;
                    end
                    else
                    begin
                        done_nv = 0;
                        rd_addr_nv = 0;
                        _ns = S_READ;
                    end
                end
            end
            S_READ:
            begin
                rd_addr_nv = rd_addr_cv + 4;
                wr_data_nv = rd_data * gain;
                wr_en_nv = 4'hf;
                _ns = S_WRITE;
            end
            S_WRITE:
            begin
                wr_addr_nv = wr_addr_cv + 4;
                wr_en_nv = 4'h0;
                if (rd_addr_cv == num_of_inp)
                begin
                    _ns = S_DONE;
                end
```

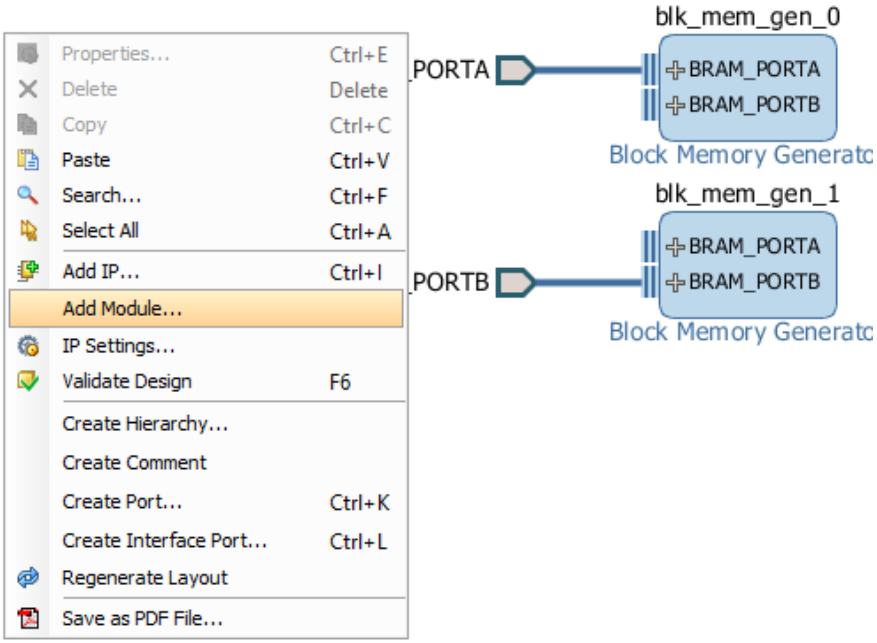
```
                else
                begin
                    _ns = S_READ;
                end
            end
            S_DONE:
            begin
                rd_addr_nv = 0;
                wr_addr_nv = 0;
                done_nv = 1;
                _ns = S_IDLE;
            end
        endcase
    end

    // *** Output ***
    assign rst = ~rst_n;
    assign rd_addr = rd_addr_cv;
    assign wr_addr = wr_addr_cv;
    assign wr_data = wr_data_cv;
    assign wr_en = wr_en_cv;
    assign done = done_cv;

endmodule
```

4. Save file tersebut, kemudian buat block design baru dengan nama **design_3**. Pada design ini kita akan mensimulasikan memory input, memory output, dan multiplier.

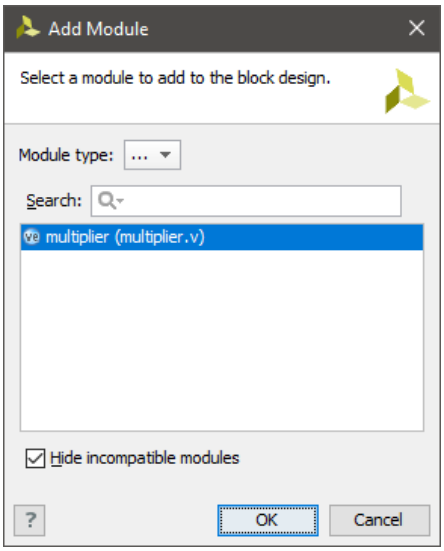
5. Pada block design tambahkan **dua Block Memory Generator,** dengan mode **BRAM Controller** dan tipe **true dual port RAM**.



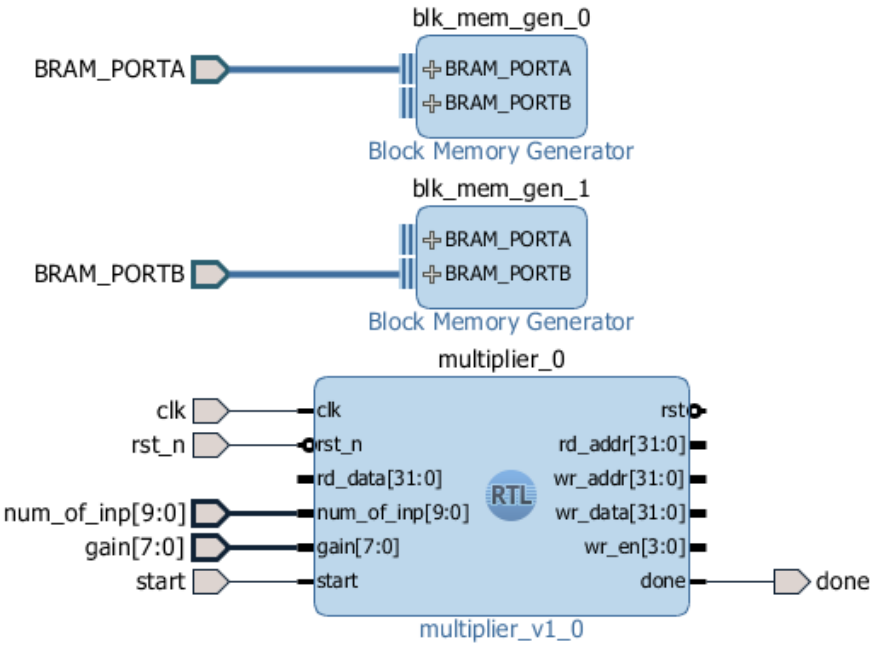
6. **Right-click** pada block design, kemudian pilih menu **Add Module**. Menu ini berfungsi untuk menambahkan file Verilog multiplier yang telah dibuat.



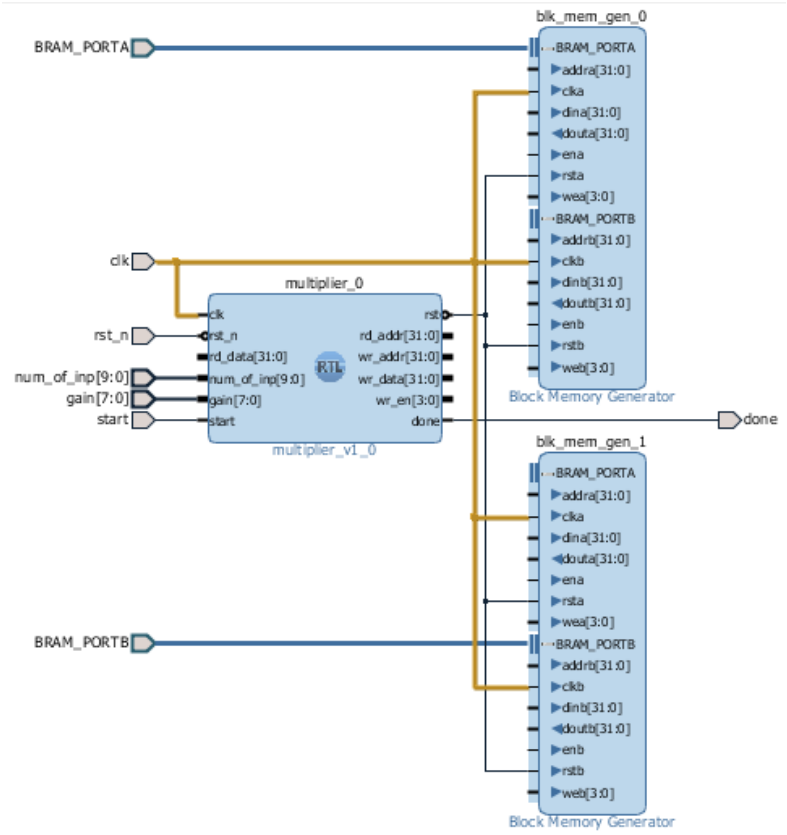
7. Pilih file **multiplier.v**.



8. Pada setiap port berikut ini: **clk, rst_n, num_of_inp, gain, start, done**, right-click kemudian pilih **Create Port**. Hal ini berfungsi untuk membuat port yang akan dihubungkan ke file testbench.

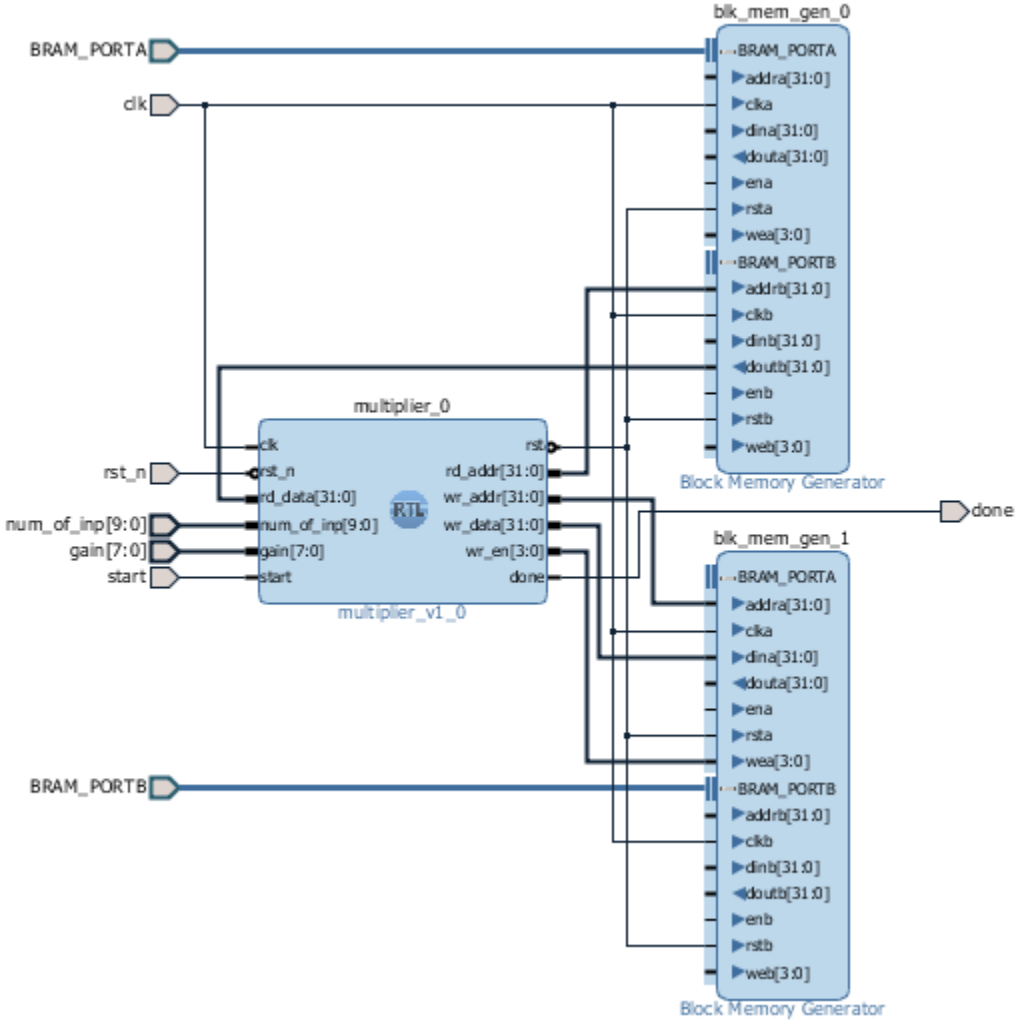


9. Hubungkan pin **clk** ke **clka** dan **clkb** dari kedua memory seperti pada gambar berikut ini. Kemudian hubungkan pin **rst** dari IP multiplier ke **rsta** dan **rstb** dari kedua memory. Sistem ZYNQ menggunakan reset **active-low** sedangkan memory menggunakan **active-high**.

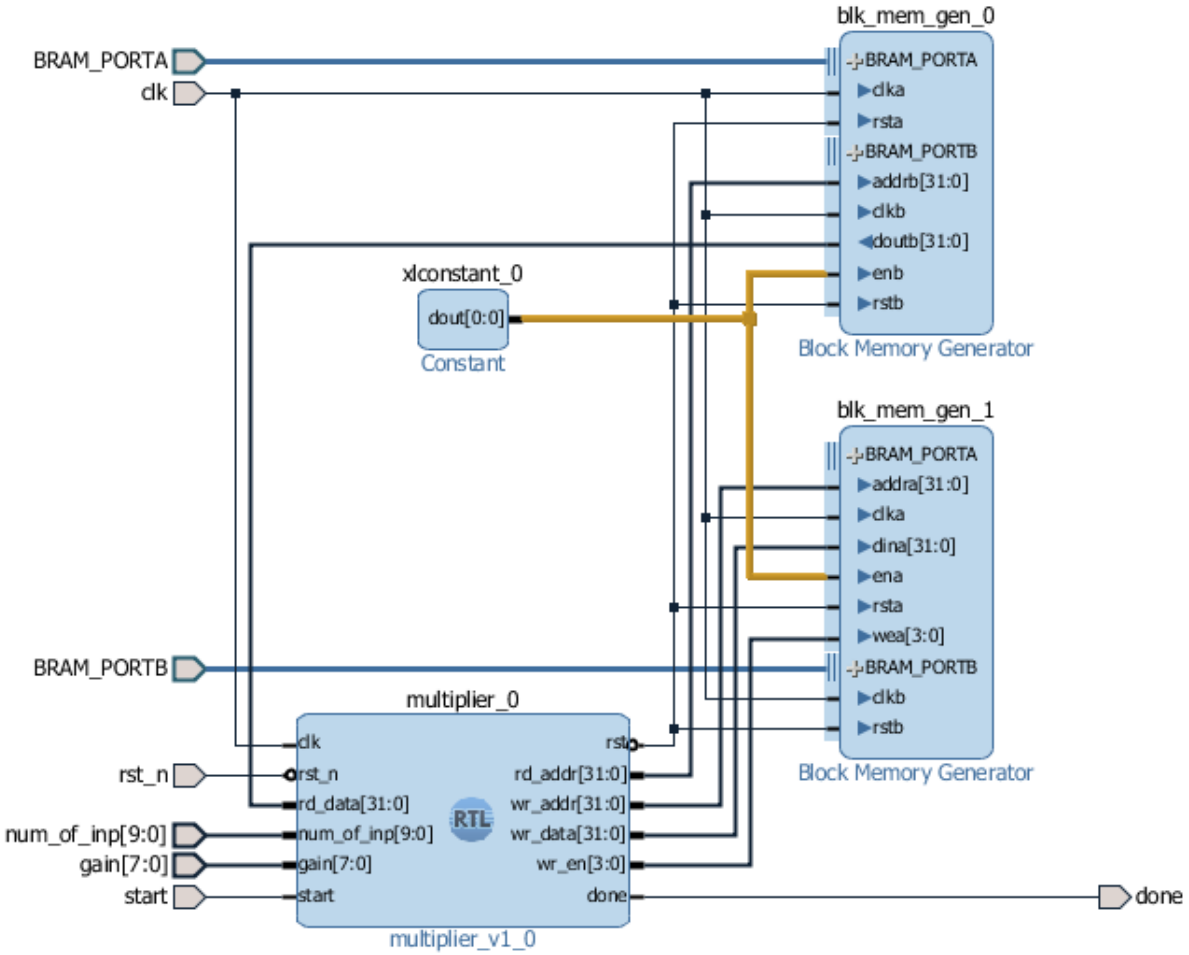


10. Hubungkan port address dan data input/output dari IP multiplier ke block memoru input dan output sesuai tabel berikut ini.

Multiplier	Memory Input	Memory Output
rd_addr	addrb	-
rd_data	doutb	-
wr_addr	-	addra
wr_data	-	dina
wr_en	-	wea



11. **Add IP**, tambahkan IP **Constant**, kemudian hubungkan ke **enb memory input** dan **ena memory output**.



12. Buat **HDL wrapper** kemudian **generate output product**.

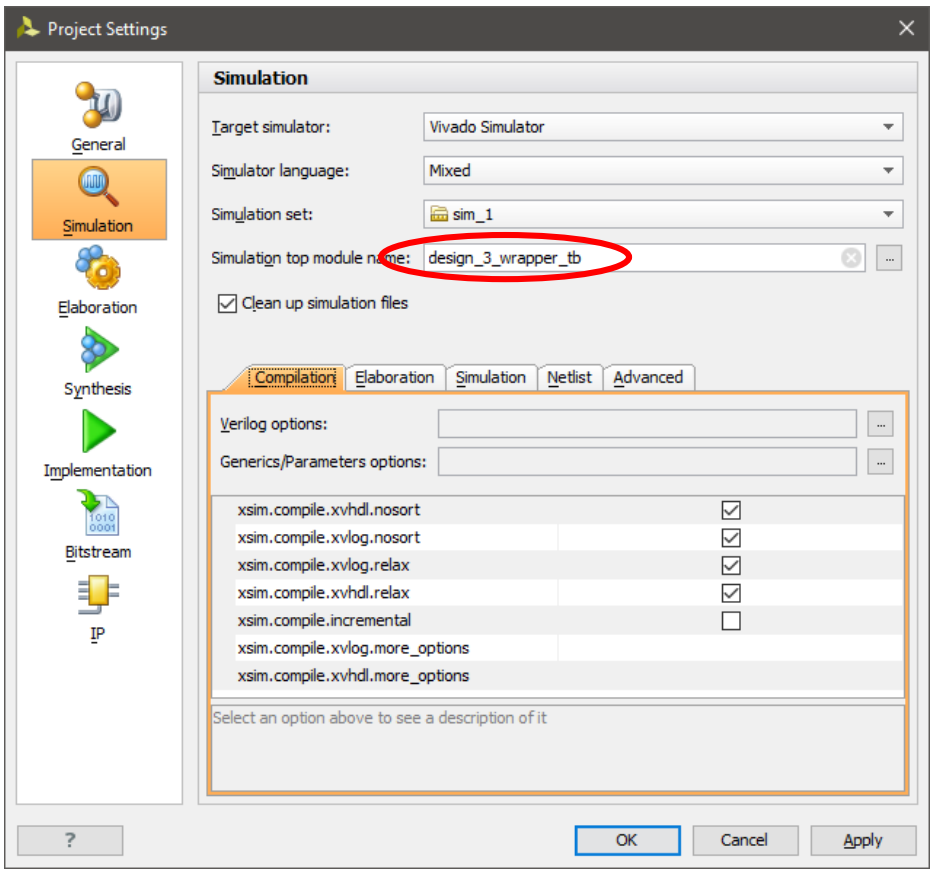
13. Buat file testbench **design_3_wrapper_tb** sebagai berikut.

```
`timescale 1ns / 1ps

module design_3_wrapper_tb();
    localparam T = 8;

    reg [31:0]BRAM_PORTA_addr;
    reg [31:0]BRAM_PORTA_din;
    wire [31:0]BRAM_PORTA_dout;
    reg BRAM_PORTA_en;
    reg [3:0]BRAM_PORTA_we;
    reg [31:0]BRAM_PORTB_addr;
    reg [31:0]BRAM_PORTB_din;
    wire [31:0]BRAM_PORTB_dout;
    reg BRAM_PORTB_en;
    reg [3:0]BRAM_PORTB_we;
    reg clk;
    wire done;
    reg [7:0]gain;
    reg [9:0]num_of_inp;
    reg rst_n;
```

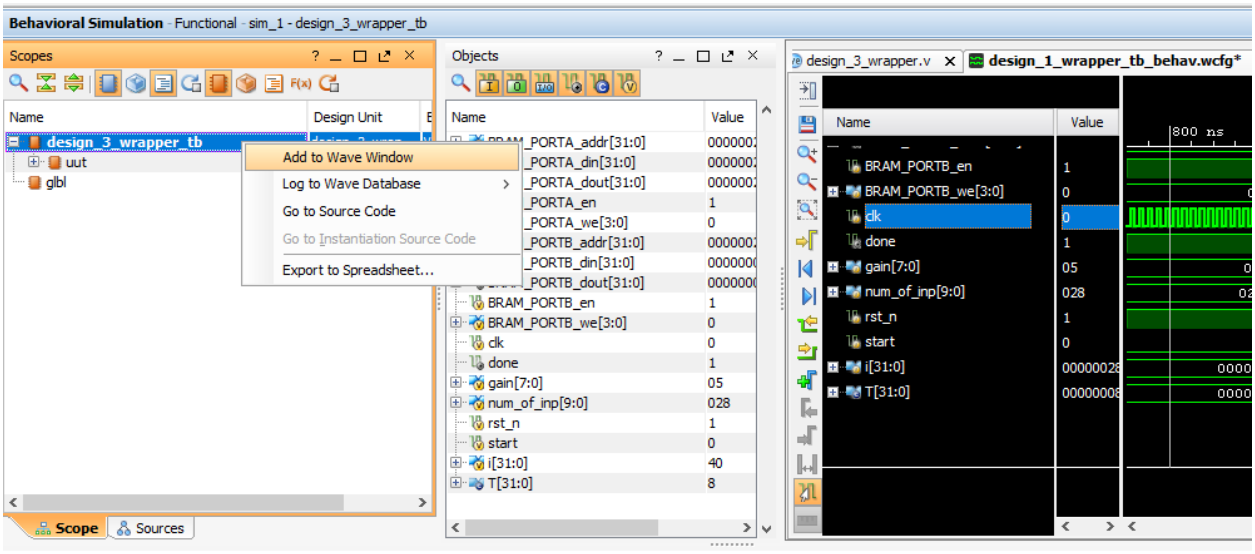
```
    reg start;

    integer i;

    design_3_wrapper uut
    (
        .BRAM_PORTA_addr(BRAM_PORTA_addr),
        .BRAM_PORTA_din(BRAM_PORTA_din),
        .BRAM_PORTA_dout(BRAM_PORTA_dout),
        .BRAM_PORTA_en(BRAM_PORTA_en),
        .BRAM_PORTA_we(BRAM_PORTA_we),
        .BRAM_PORTB_addr(BRAM_PORTB_addr),
        .BRAM_PORTB_din(BRAM_PORTB_din),
        .BRAM_PORTB_dout(BRAM_PORTB_dout),
        .BRAM_PORTB_en(BRAM_PORTB_en),
        .BRAM_PORTB_we(BRAM_PORTB_we),
        .clk(clk),
        .done(done),
        .gain(gain),
        .num_of_inp(num_of_inp),
        .rst_n(rst_n),
        .start(start)
    );

    always
    begin
        // *** Clock ***
        clk = 0;
        #(T/2);
        clk = 1;
        #(T/2);
    end

    initial
    begin
        // *** Init ***
        BRAM_PORTA_addr = 0;
        BRAM_PORTA_din = 0;
        BRAM_PORTA_en = 1;
        BRAM_PORTA_we = 0;
        BRAM_PORTB_addr = 0;
        BRAM_PORTB_din = 0;
        BRAM_PORTB_en = 1;
        BRAM_PORTB_we = 0;
        gain = 0;
        num_of_inp = 0;
        start = 0;

        // *** Reset ***
        rst_n = 0;
        #(T*5);
        rst_n = 1;
        #(T*5);

        // ### 1 ###
        // *** Configuration ***
        gain = 5;
```

```
        num_of_inp = 10*4;
        // *** Write input ***
        BRAM_PORTA_we = 1;
        for (i = 0; i < 10*4; i = i+4)
        begin
            BRAM_PORTA_addr = i;
            BRAM_PORTA_din = i+4;
            #T;
        end
        BRAM_PORTA_we = 0;
        #(T*10);
        // *** Start ***
        start = 1;
        #T;
        start = 0;
        // Wait until process is done
        #(T*50);
        // *** Read output ***
        for (i = 0; i < 10*4; i = i+4)
        begin
            BRAM_PORTB_addr = i;
            #T;
        end
        #(T*10);
    end

endmodule
```

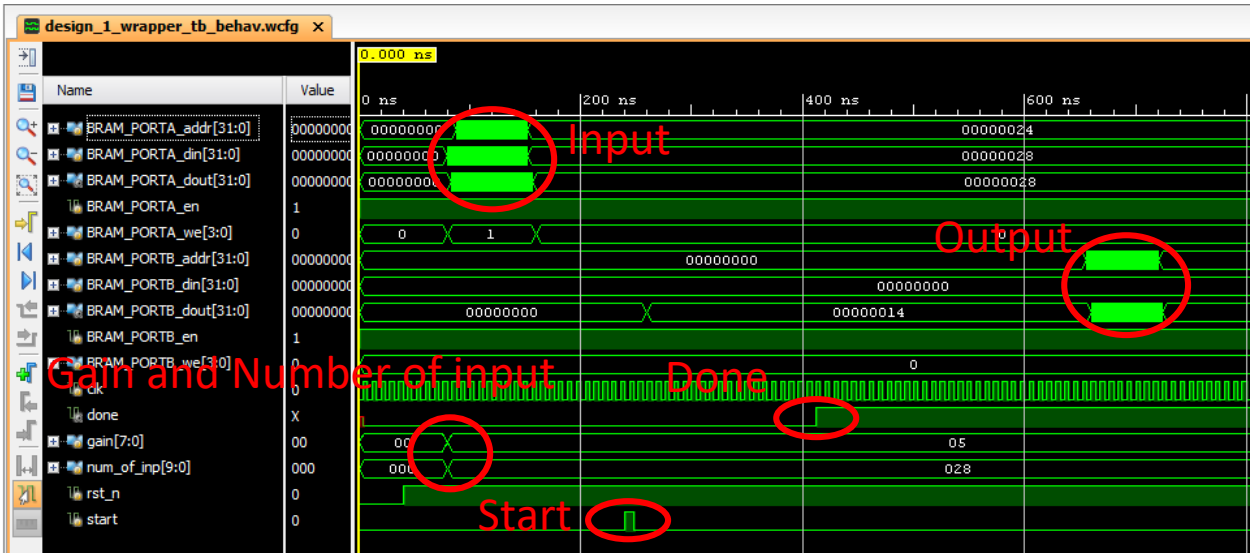
14. Ubah top module pada **Simulation Settings** menjadi **design_3_wrapper_tb**. kemudian **Run Simulation**.



15. Tambahkan sinyal yang diinginkan dengan cara **right-click** kemudian pilih **Add to Wave Window.**

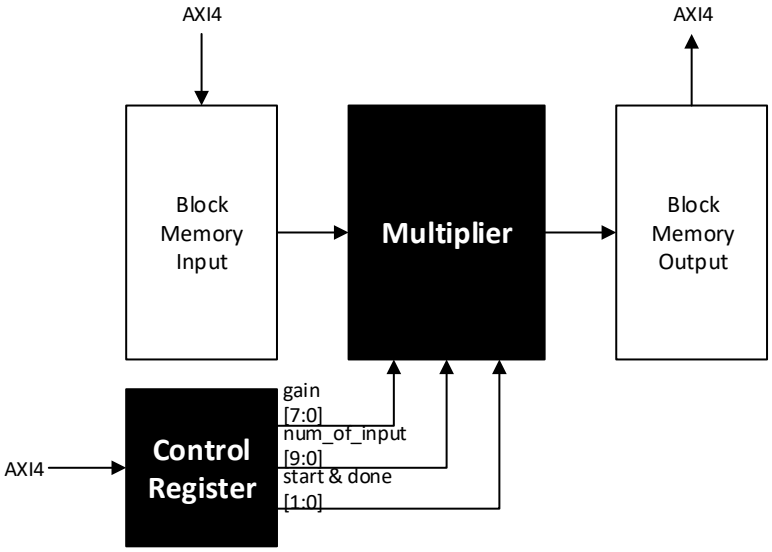


16. Jika tidak ada error, maka hasilnya seperti pada gambar berikut ini.


design_1_wrapper_tb_behav.wcfg
Name
Value
BRAM_PORTA_addr[31:0]
BRAM_PORTA_din[31:0]
BRAM_PORTA_dout[31:0]
BRAM_PORTA_en
BRAM_PORTA_we[3:0]
BRAM_PORTB_addr[31:0]
BRAM_PORTB_din[31:0]
BRAM_PORTB_dout[31:0]
BRAM_PORTB_en
BRAM_PORTB_we[3:0]
clk
done
gain[7:0]
num_of_inp[9:0]
rst_n
start
0.000 ns
0 ns
200 ns
400 ns
600 ns
Input
Output
Gain and Number of input
Done
Start

IV. Pembuatan IP Core Control Register

Pada design_3, sinyal gain, num_of_input, start, dan done dinkontrol melalui testbench. Pada design ini kita akan membuat control register seperti pada block diagram berikut ini.



Control register akan dihubungkan ke bus AXI4-Lite sehingga dapat diakses processor. Sehingga software dapat memprogram gain, num_of_input serta mengontrol IP multiplier dengan sinyal start dan done. Memory map dari control register ditampilkan pada tabel berikut ini.

Offset Address	Bit	Fungsi	Tipe
0x0	0-9	Number of Input	R/W
	10	Start	R/W
	11	Done	R
0x4	0-7	Gain	R/W

1. Pada **Flow Navigator**, pilih menu **Add Sources**, kemudian pilih **Add or create design sources**.

2. Buat file Verilog baru dengan click button **Create File**, kemudian beri nama file **axi_control**.

```
`timescale 1ns / 1ps

module axi_control
    #(
        C_ADDR_WIDTH      = 32,
        C_DATA_WIDTH      = 32
    )
    (
        // *** Clock and reset signals ***
        input  wire                           aclk,
        input  wire                           aresetn,
        // *** AXI4-lite slave signals ***
        output wire                           s_axi_awready,
        input  wire [C_ADDR_WIDTH-1:0]        s_axi_awaddr,
        input  wire                           s_axi_awvalid,
```

```
        output wire                             s_axi_wready,
        input  wire [C_DATA_WIDTH-1:0]          s_axi_wdata,
        input  wire [C_DATA_WIDTH/8-1:0]        s_axi_wstrb,
        input  wire                             s_axi_wvalid,
        input  wire                             s_axi_bready,
        output wire [1:0]                       s_axi_bresp,
        output wire                             s_axi_bvalid,
        output wire                             s_axi_arready,
        input  wire [C_ADDR_WIDTH-1:0]          s_axi_araddr,
        input  wire                             s_axi_arvalid,
        input  wire                             s_axi_rready,
        output wire [C_DATA_WIDTH-1:0]          s_axi_rdata,
        output wire [1:0]                       s_axi_rresp,
        output wire                             s_axi_rvalid,
        // *** Control signals ***
        output wire [9:0]                       num_of_inp,
        output wire [7:0]                       gain,
        output wire                             start,
        input wire                              done
    );

    // *** Register map ***
    // 0x00: done[11] (R) | start[10] (R/W) | num_of_inp[9:0] (R/W)
    // 0x04: gain[7:0] (R/W)
    localparam C_ADDR_BITS = 4;
    // *** Address ***
    localparam C_ADDR_CTRL = 4'h00,
               C_ADDR_GAIN = 4'h04;
    // *** AXI write FSM ***
    localparam S_WRIDLE = 2'd0,
               S_WRDATA = 2'd1,
               S_WRRESP = 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;
    // *** Internal registers ***
    reg [9:0] num_of_inp_reg;
    reg start_reg;
    reg done_reg;
    reg [7:0] gain_reg;
    // *** User signals ***

    // *** 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_CTRL:
                    rdata <= {done_reg, start_reg, num_of_inp_reg[9:0]};
                C_ADDR_GAIN:
                    rdata <= gain_reg[7:0];
            endcase
    end

    // *** Internal registers
************************************************
    // *** num_of_inp_reg[9:0], start_reg ***
    always @(posedge aclk)
    begin
        if (!aresetn)
        begin
            start_reg <= 0;
            num_of_inp_reg[9:0] <= 0;
        end
        else if (w_hs && waddr == C_ADDR_CTRL)
        begin
            if (s_axi_wdata[10] == 1)
                start_reg <= 1;
            num_of_inp_reg[9:0] <= (s_axi_wdata[31:0] & wmask) |
(num_of_inp_reg[9:0] & ~wmask);
```

```
        end
        else
        begin
            start_reg <= 0;
        end
    end

    // *** gain_reg[7:0] ***
    always @(posedge aclk)
    begin
        if (!aresetn)
            gain_reg[7:0] <= 0;
        else if (w_hs && waddr == C_ADDR_GAIN)
            gain_reg[7:0] <= (s_axi_wdata[31:0] & wmask) | (gain_reg[7:0] &
~wmask);
    end

    // *** done_reg ***
    always @(posedge aclk)
    begin
        if (!aresetn)
            done_reg <= 0;
        else
            done_reg <= done;
    end

    // *** Instantiation
************************************************
    assign num_of_inp = num_of_inp_reg;
    assign gain = gain_reg;
    assign start = start_reg;

endmodule
```

3. Buat block design baru dengan nama **design_4**.

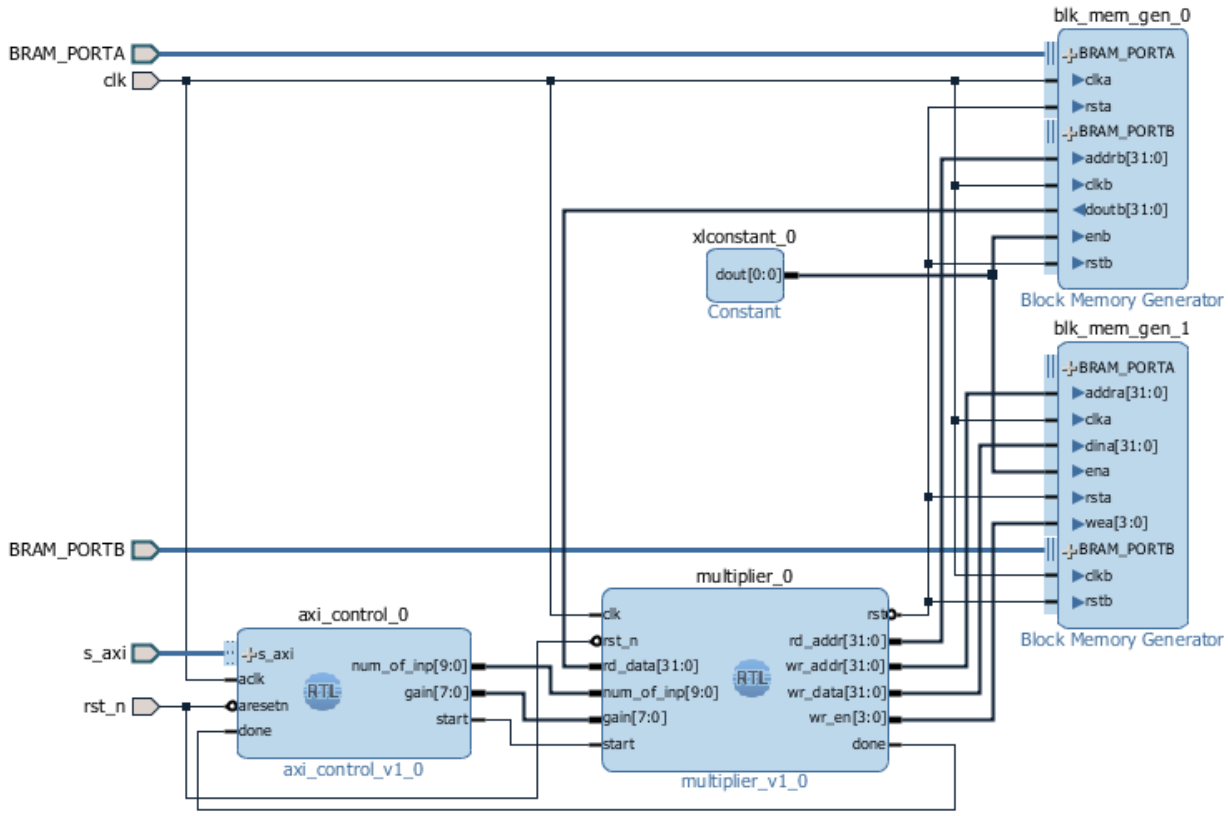
4. Copy block design_3 dengan cara **CTRL+A** lalu **CTRL+C,** kemudian **CTRL-V** pada design_4.

5. Tambahkan module **axi_control** dengan cara **right-click** kemudian pilih **Add Module**.

6. **Right-click** pada port **s_axi** dari **axi_control** kemudian pilih **Create Interface Port**.

7. Hubungkan **aclk** dan **aresetn** ke **clk** dan **rst_n**.

8. Delete port **gain, num_of_input, start,** dan **done** dari IP multiplier, kemudian hubungkan port tersebut dengan port pada IP axi_control, sehingga hasil akhirnya seperti pada gambar berikut ini.



9. Buat **HDL wrapper** kemudian **generate output product**.

10. Buat file testbench **design_4_wrapper_tb** sebagai berikut.

```
`timescale 1ns / 1ps

module design_4_wrapper_tb();
    localparam T = 8;

    // *** Multiplier ***
    reg [31:0]BRAM_PORTA_addr;
    reg [31:0]BRAM_PORTA_din;
    wire [31:0]BRAM_PORTA_dout;
    reg BRAM_PORTA_en;
    reg [3:0]BRAM_PORTA_we;
    reg [31:0]BRAM_PORTB_addr;
    reg [31:0]BRAM_PORTB_din;
    wire [31:0]BRAM_PORTB_dout;
    reg BRAM_PORTB_en;
    reg [3:0]BRAM_PORTB_we;
    reg clk;
    reg rst_n;
    // *** AXI control ***
    wire s_axi_arready;
    reg [31:0] s_axi_araddr;
    reg s_axi_arvalid;
    wire s_axi_awready;
    reg [31:0] s_axi_awaddr;
    reg s_axi_awvalid;
```

```
    reg s_axi_bready;
    wire [1:0] s_axi_bresp;
    wire s_axi_bvalid;
    reg s_axi_rready;
    wire [31:0] s_axi_rdata;
    wire [1:0] s_axi_rresp;
    wire s_axi_rvalid;
    wire s_axi_wready;
    reg [31:0] s_axi_wdata;
    reg [3:0] s_axi_wstrb;
    reg s_axi_wvalid;

    integer i;

    design_4_wrapper uut
    (
        .BRAM_PORTA_addr(BRAM_PORTA_addr),
        .BRAM_PORTA_din(BRAM_PORTA_din),
        .BRAM_PORTA_dout(BRAM_PORTA_dout),
        .BRAM_PORTA_en(BRAM_PORTA_en),
        .BRAM_PORTA_we(BRAM_PORTA_we),
        .BRAM_PORTB_addr(BRAM_PORTB_addr),
        .BRAM_PORTB_din(BRAM_PORTB_din),
        .BRAM_PORTB_dout(BRAM_PORTB_dout),
        .BRAM_PORTB_en(BRAM_PORTB_en),
        .BRAM_PORTB_we(BRAM_PORTB_we),
        .clk(clk),
        .rst_n(rst_n),
        .s_axi_araddr(s_axi_araddr),
        .s_axi_arready(s_axi_arready),
        .s_axi_arvalid(s_axi_arvalid),
        .s_axi_awaddr(s_axi_awaddr),
        .s_axi_awready(s_axi_awready),
        .s_axi_awvalid(s_axi_awvalid),
        .s_axi_bready(s_axi_bready),
        .s_axi_bresp(s_axi_bresp),
        .s_axi_bvalid(s_axi_bvalid),
        .s_axi_rdata(s_axi_rdata),
        .s_axi_rready(s_axi_rready),
        .s_axi_rresp(s_axi_rresp),
        .s_axi_rvalid(s_axi_rvalid),
        .s_axi_wdata(s_axi_wdata),
        .s_axi_wready(s_axi_wready),
        .s_axi_wstrb(s_axi_wstrb),
        .s_axi_wvalid(s_axi_wvalid)
    );

    always
    begin
        // *** Clock ***
        clk = 0;
        #(T/2);
        clk = 1;
        #(T/2);
    end

    initial
```

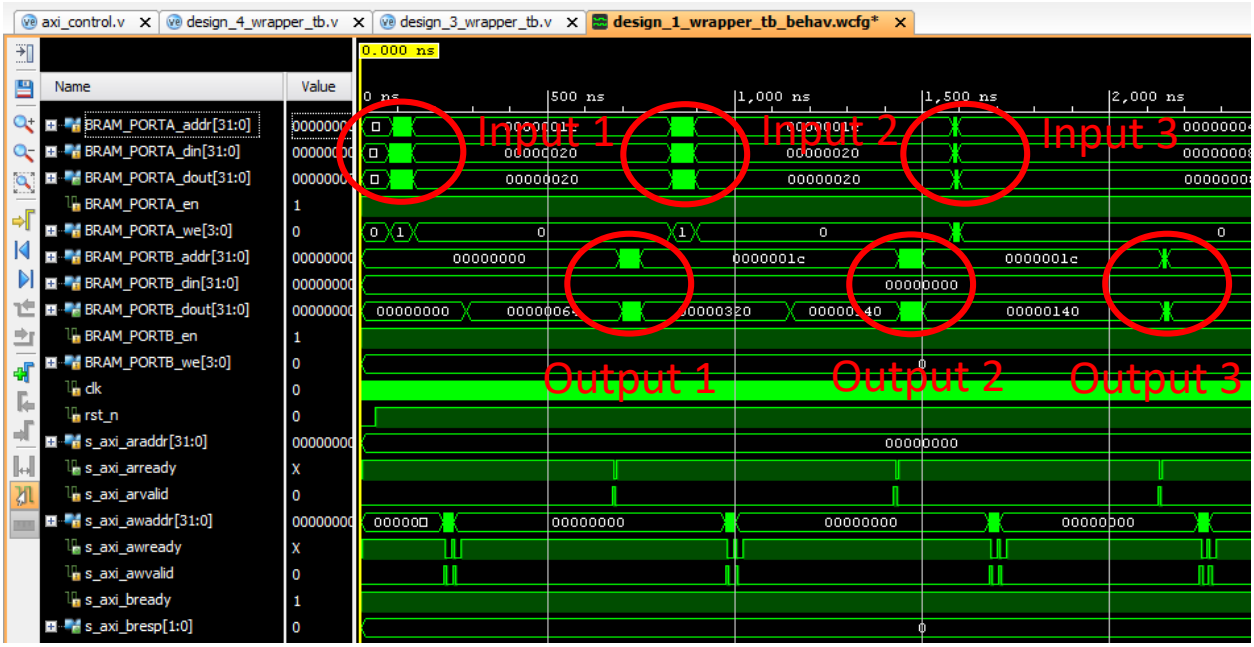
```
    begin
        // *** Init ***
        BRAM_PORTA_addr = 0;
        BRAM_PORTA_din = 0;
        BRAM_PORTA_en = 1;
        BRAM_PORTA_we = 0;
        BRAM_PORTB_addr = 0;
        BRAM_PORTB_din = 0;
        BRAM_PORTB_en = 1;
        BRAM_PORTB_we = 0;
        s_axi_awaddr = 0;
        s_axi_awvalid = 0;
        s_axi_wstrb = 0;
        s_axi_wdata = 0;
        s_axi_wvalid = 0;
        s_axi_bready = 1;
        s_axi_araddr = 0;
        s_axi_arvalid = 0;
        s_axi_rready = 1;

        // *** Reset ***
        rst_n = 0;
        #(T*5);
        rst_n = 1;
        #(T*5);

        // ### 1 ###
        BRAM_PORTA_we = 1;
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTA_addr = i;
            BRAM_PORTA_din = i+4;
            #T;
        end
        BRAM_PORTA_we = 0;
        #(T*10);
        // *** Configuration and start ***
        axi_control_write(4'h4, 8'd25);
        axi_control_write(4'h0, 12'h420);
        // Wait until process is done
        #(T*50);
        axi_control_read(4'h0);
        // *** Read output ***
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTB_addr = i;
            #T;
        end
        #(T*10);

        // ### 2 ###
        BRAM_PORTA_we = 1;
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTA_addr = i;
            BRAM_PORTA_din = i+4;
            #T;
```

```
        end
        BRAM_PORTA_we = 0;
        #(T*10);
        // *** Configuration and start ***
        axi_control_write(4'h4, 8'd10);
        axi_control_write(4'h0, 12'h420);
        // Wait until process is done
        #(T*50);
        axi_control_read(4'h0);
        // *** Read output ***
        for (i = 0; i < 32; i = i+4)
        begin
            BRAM_PORTB_addr = i;
            #T;
        end
        #(T*10);

        // ### 3 ###
        BRAM_PORTA_we = 1;
        for (i = 0; i < 8; i = i+4)
        begin
            BRAM_PORTA_addr = i;
            BRAM_PORTA_din = i+4;
            #T;
        end
        BRAM_PORTA_we = 0;
        #(T*10);
        // *** Configuration and start ***
        axi_control_write(4'h4, 8'd5);
        axi_control_write(4'h0, 12'h408);
        // Wait until process is done
        #(T*50);
        axi_control_read(4'h0);
        // *** Read output ***
        for (i = 0; i < 8; i = i+4)
        begin
            BRAM_PORTB_addr = i;
            #T;
        end
        #(T*10);

        // ### 4 ###
        // *** Configuration and start ***
        axi_control_write(4'h4, 8'd3);
        axi_control_write(4'h0, 12'h400);
        // Wait until process is done
        #(T*50);
        axi_control_read(4'h0);
    end

    task axi_control_write;
        input [31:0] awaddr;
        input [31:0] wdata;
        begin
            // *** Write address ***
            s_axi_awaddr = awaddr;
            s_axi_awvalid = 1;
```

```
            #T;
            s_axi_awvalid = 0;
            // *** Write data ***
            s_axi_wdata = wdata;
            s_axi_wstrb = 4'hf;
            s_axi_wvalid = 1;
            #T;
            s_axi_wvalid = 0;
            #T;
        end
    endtask

    task axi_control_read;
        input [31:0] araddr;
        begin
            // *** Read address ***
            s_axi_araddr = araddr;
            s_axi_arvalid = 1;
            #T;
            s_axi_arvalid = 0;
            #T;
        end
    endtask

endmodule
```

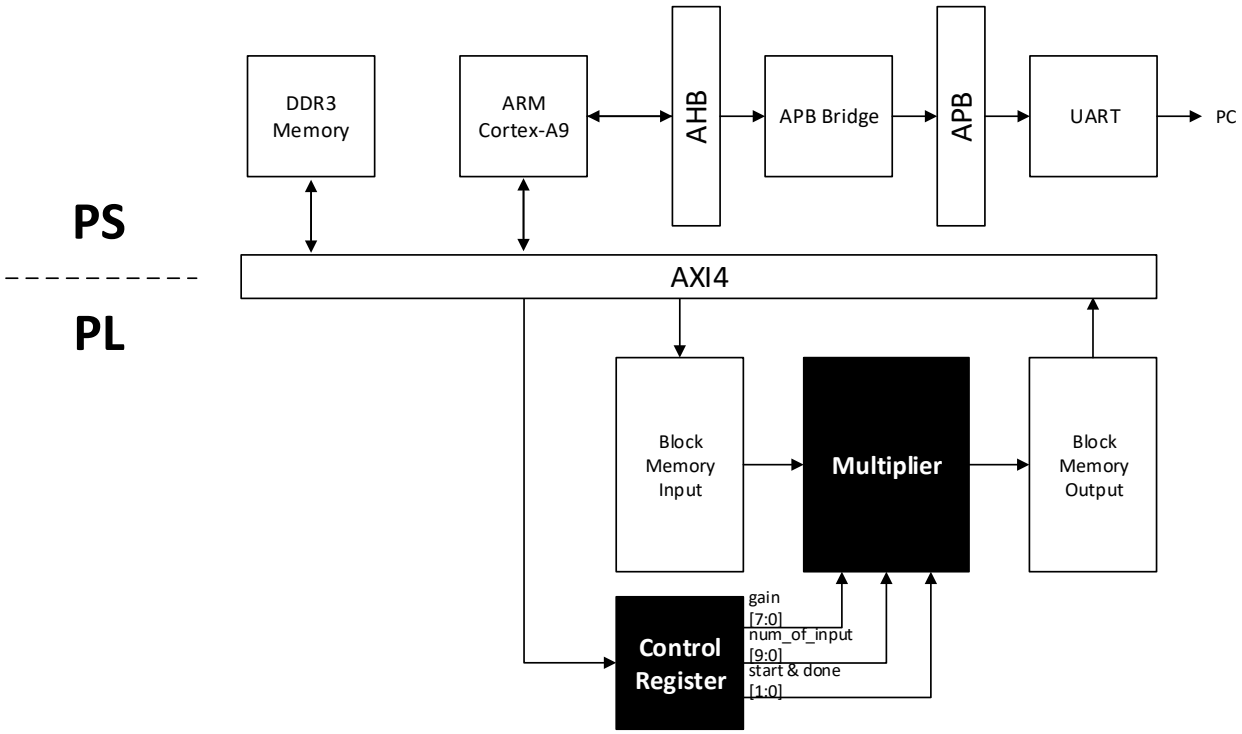
11. Ubah top module pada **Simulation Settings** menjadi **design_4_wrapper_tb**. kemudian **Run Simulation**.

12. Jika tidak ada error, maka hasilnya seperti pada gambar berikut ini. Ada tiga kali proses multiplication.



V. Implementasi Keseluruhan Sistem

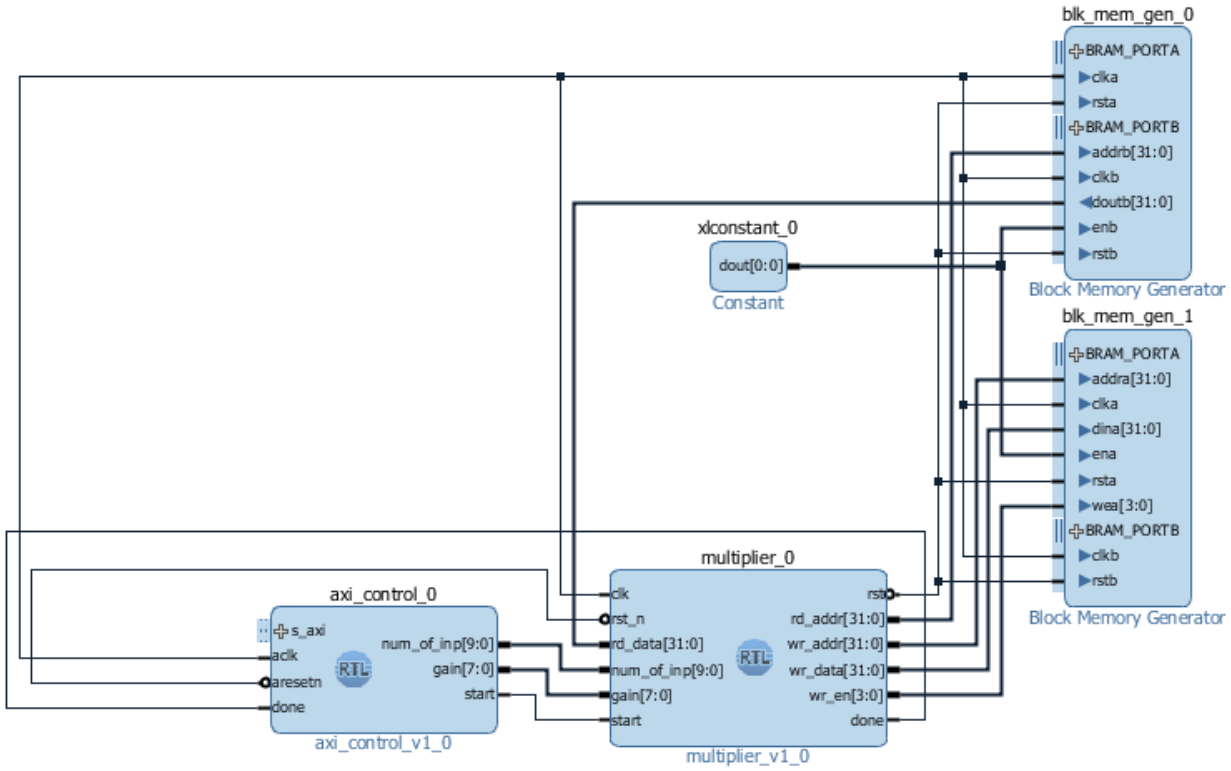
Pada design ini akan dilakukan implementasi keseluruhan sistem. Blok pada design_4 akan diintegrasikan dengan ZYNQ system seperti pada block diagram berikut ini.



IP AXI Control dan BRAM memory terhubung ke processor melalui bus AXI4. Pada PS juga UART1 diaktifkan untuk melakukan debug. UART terhubung ke processor melalui bus AHB dan APB dengan AHB-to-APB bridge. Bus AHB, APB, dan AHB-to-APB bridge merupakan bagian dari PS dan sudah ada, kita hanya perlu mengaktifkan UART1 nya saja.

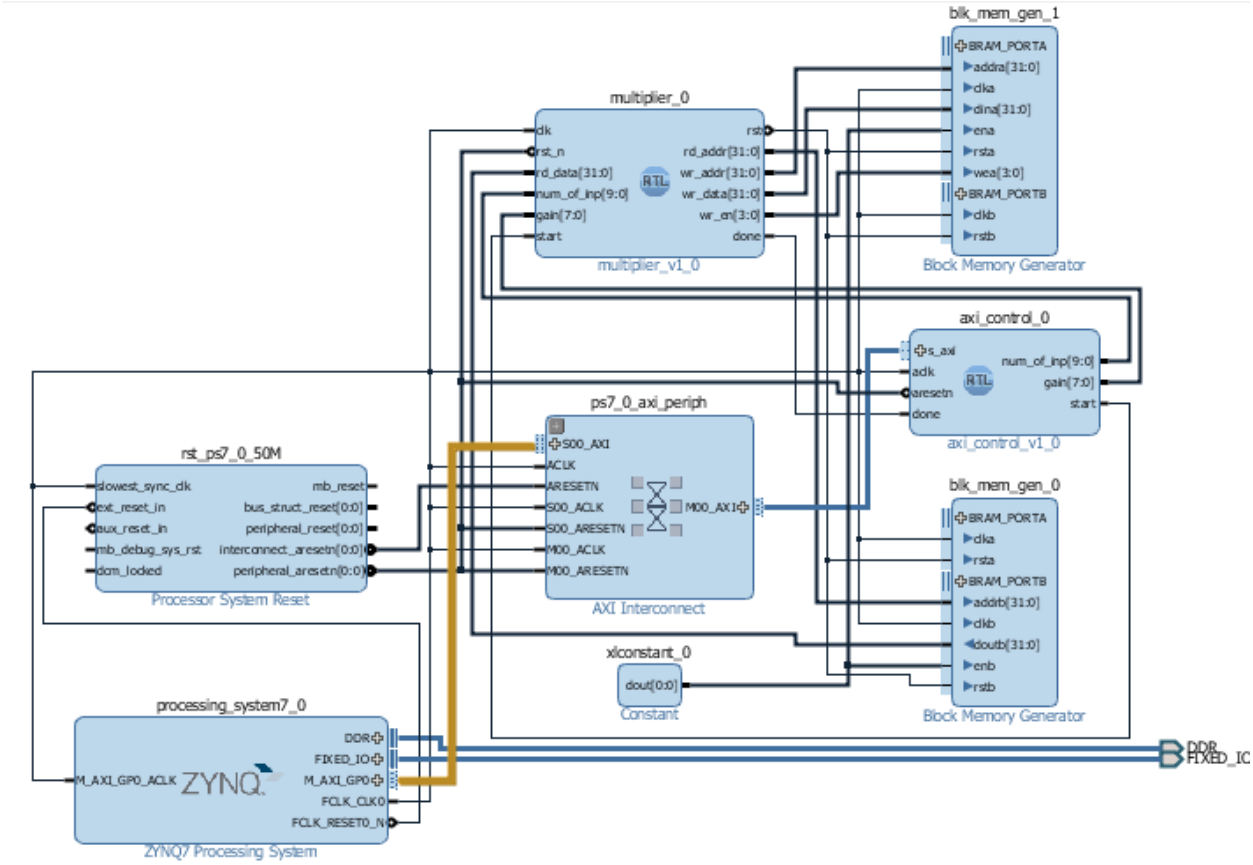
1**.** Buat block design baru dengan nama **design_5**.

2. Buat block design seperti pada gambar berikut ini. Bagian memory, multiplier, dan axi_control dapat dicopy dari **design_4**. **Delete** port **BRAM_PORTA**, **BRAM_PORTB**, **s_axi**, **clk**, dan **rst_n**, karena akan dihubungkan ke system ZYNQ.

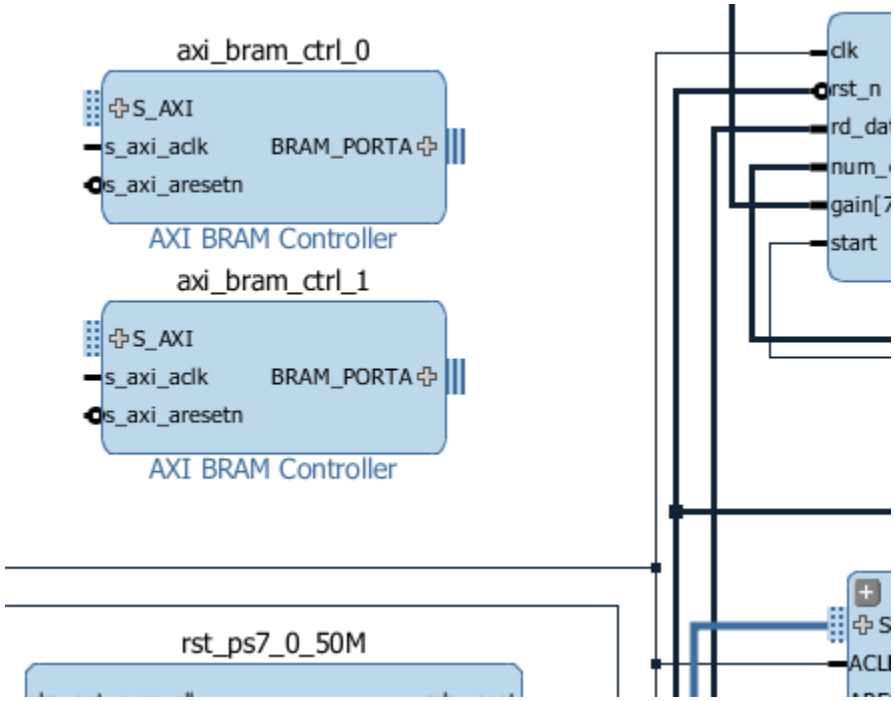


3. Tambahkan block **ZYNQ7 Processing System**, aktifkan **UART1**, kemudian **Run Block Automation**.

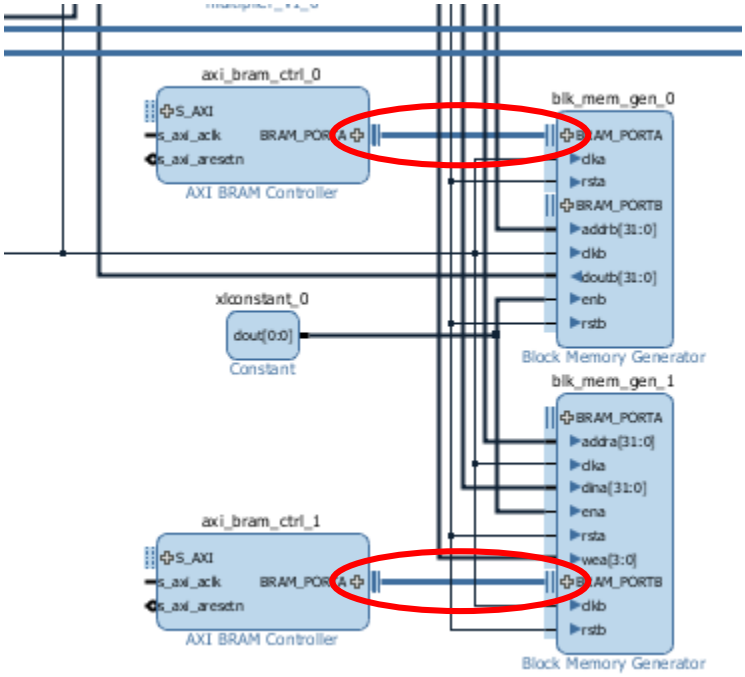
4. **Run Connection Automation** untuk menghubungkan s_axi, aclk, dan aresetn dari AXI Control ke ZYNQ7 melalui AXI Interconnect.



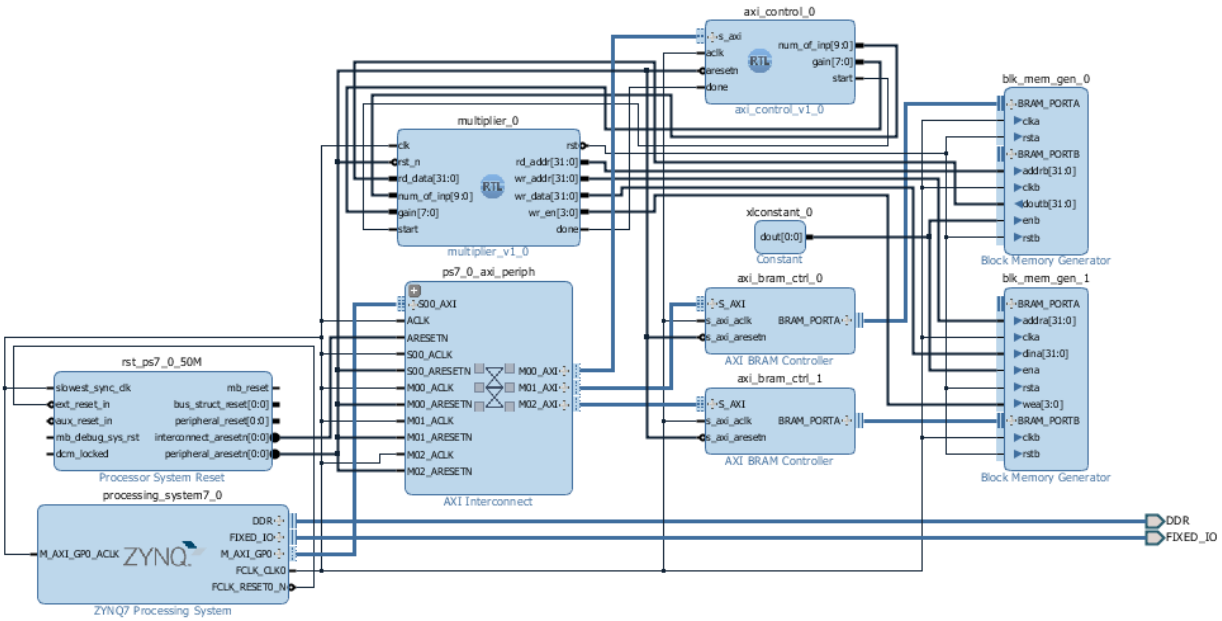
5. **Add IP AXI BRAM Controller**, kemudian ubah **Number of BRAM interfaces** menjadi **1**.



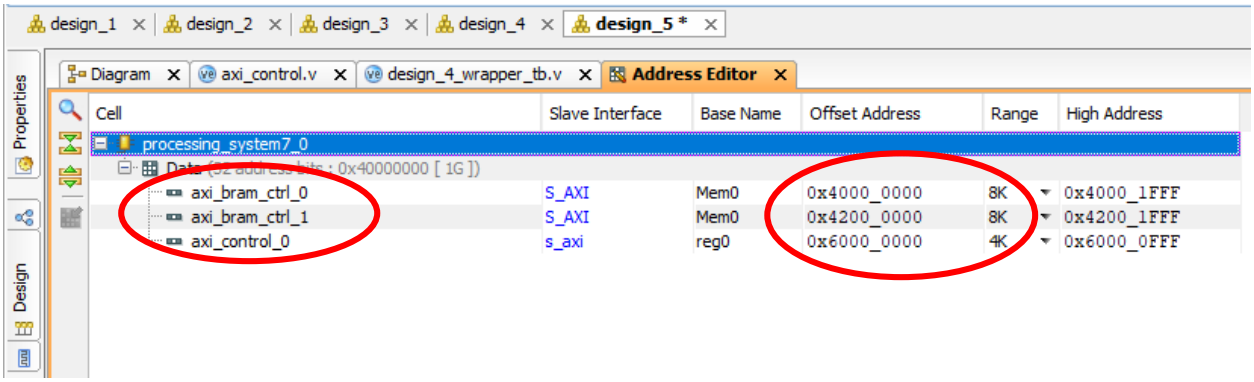
6. Hubungkan **BRAM_PORTA dari axi_bram_ctrl_0** ke **BRAM_PORTA dari blk_mem_gen_0 (memory input)**, kemudian **BRAM_PORTA dari axi_bram_ctrl_1** ke **BRAM_PORTB dari blk_mem_gen_1 (memory output)**.



7. **Run Connection Automation** untuk menghubungkan ZYNQ7 dengan AXI BRAM Controller.

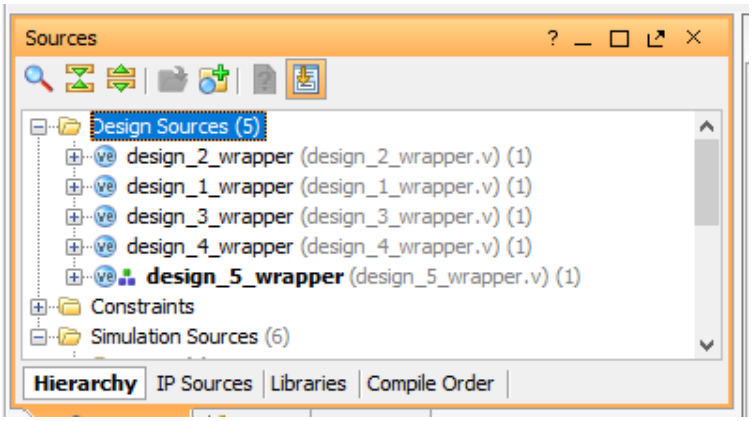


8. Pada **Address Editor,** ubah **range** menjadi **8K** untuk semua BRAM (axi_bram_ctrl_0, axi_bram_ctrl_1) dan 4K untuk AXI control (axi_control_0), kemudian ubah **Offset Address** sesuai dengan gambar berikut ini.



9. Buat **HDL wrapper** kemudian **generate output product**.

10. Ubah design_5_wrapper menjadi top module.



11. **Run Synthesis**, **Run Implementaton**, dan **Generate Bitstream**.

12. **Export Hardware**, kemudian **Launch SDK**.

13. Pada SDK, **buat project baru** dengan template helloworld.

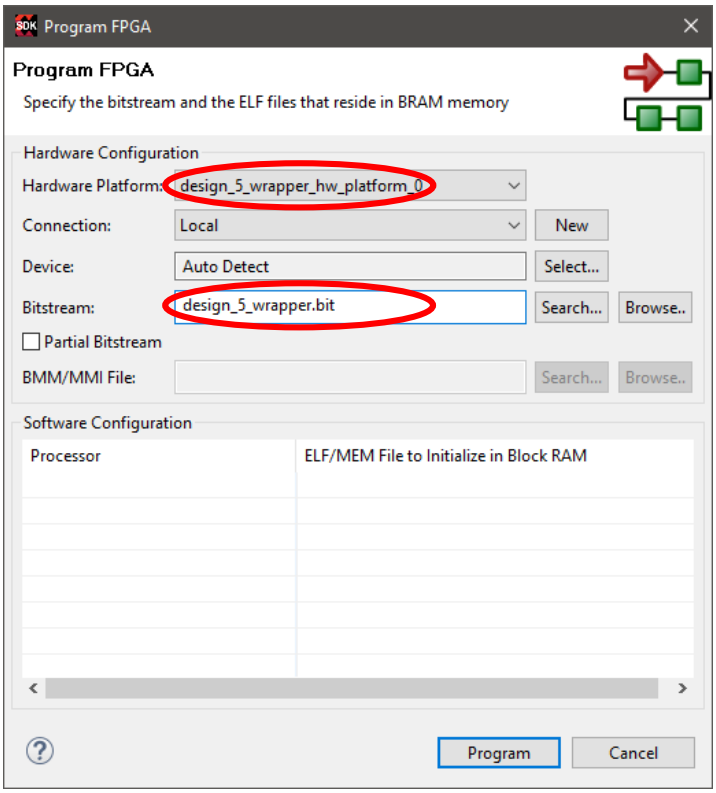
14. Tambahkan code C berikut ini.

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

#define MEM_INP_BASE 0x40000000
#define MEM_OUT_BASE 0x42000000
#define CTRL_BASE 0x60000000
#define NUM_OF_INPUT 8

uint32_t *meminp_p, *memout_p, *ctrl_p;

int main()
{
    while (1)
    {
        // *** Initialize pointer ***
        meminp_p = (uint32_t *)MEM_INP_BASE;
```

```
        memout_p = (uint32_t *)MEM_OUT_BASE;
        ctrl_p = (uint32_t *)CTRL_BASE;

        // Write gain
        *(ctrl_p+1) = 10;
        // *** Write input ***
        printf("\nInput: ");
        for (int i = 0; i < NUM_OF_INPUT; i++)
        {
            *(meminp_p+i) = i+1;
            printf("%d, ", (unsigned int)*(meminp_p+i));
        }

        // Write number of input and set start bit
        *(ctrl_p+0) = (1 << 10) | NUM_OF_INPUT*4;

        // Polling until process is done
        while (!(*(ctrl_p+0) & (1 << 11)));

        // *** Read from block memory output ***
        printf("\nOutput: ");
        for (int i = 0; i < NUM_OF_INPUT; i++)
            printf("%d, ", (unsigned int)*(memout_p+i));
        printf("\n");

        sleep(1);
    }

    return 0;
}
```

15. Build project.

16. Program FPGA dengan **design_5_wrapper.bit.**



17. Program CPU.

18. Jika tidak ada error, maka hasil akhirnya seperti gambar berikut ini.