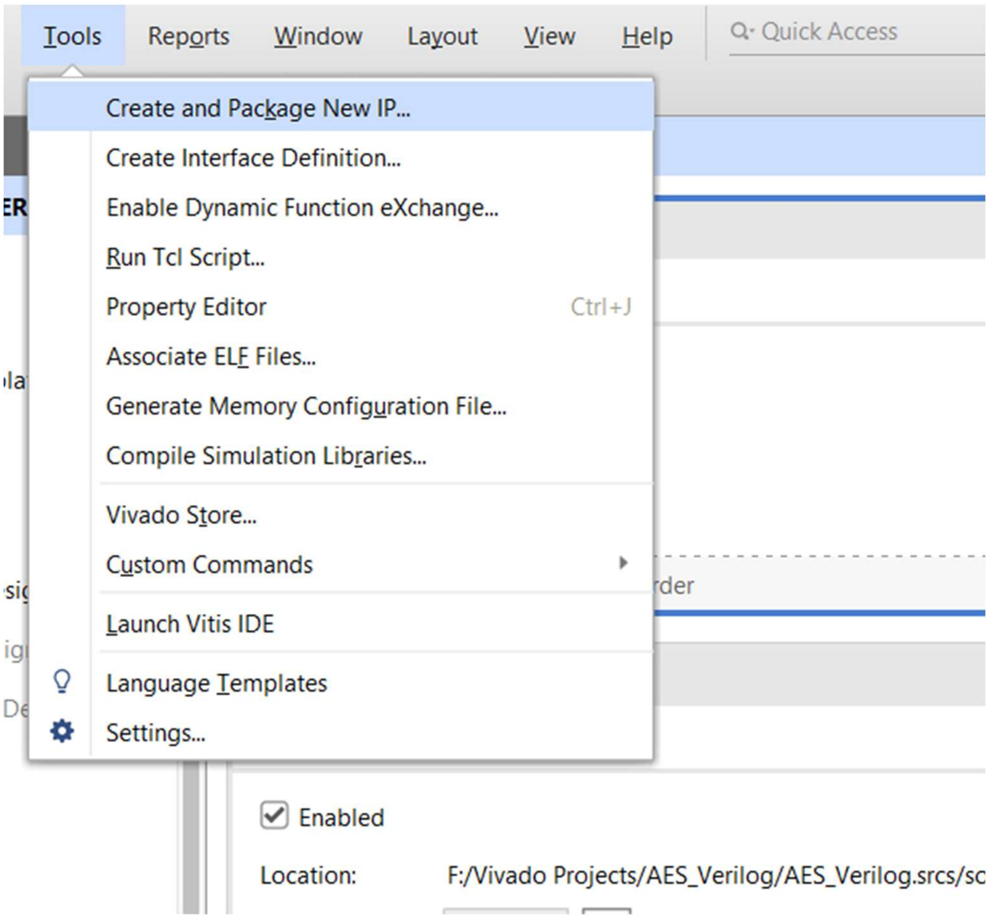
Creating Your Own IP in Vivado

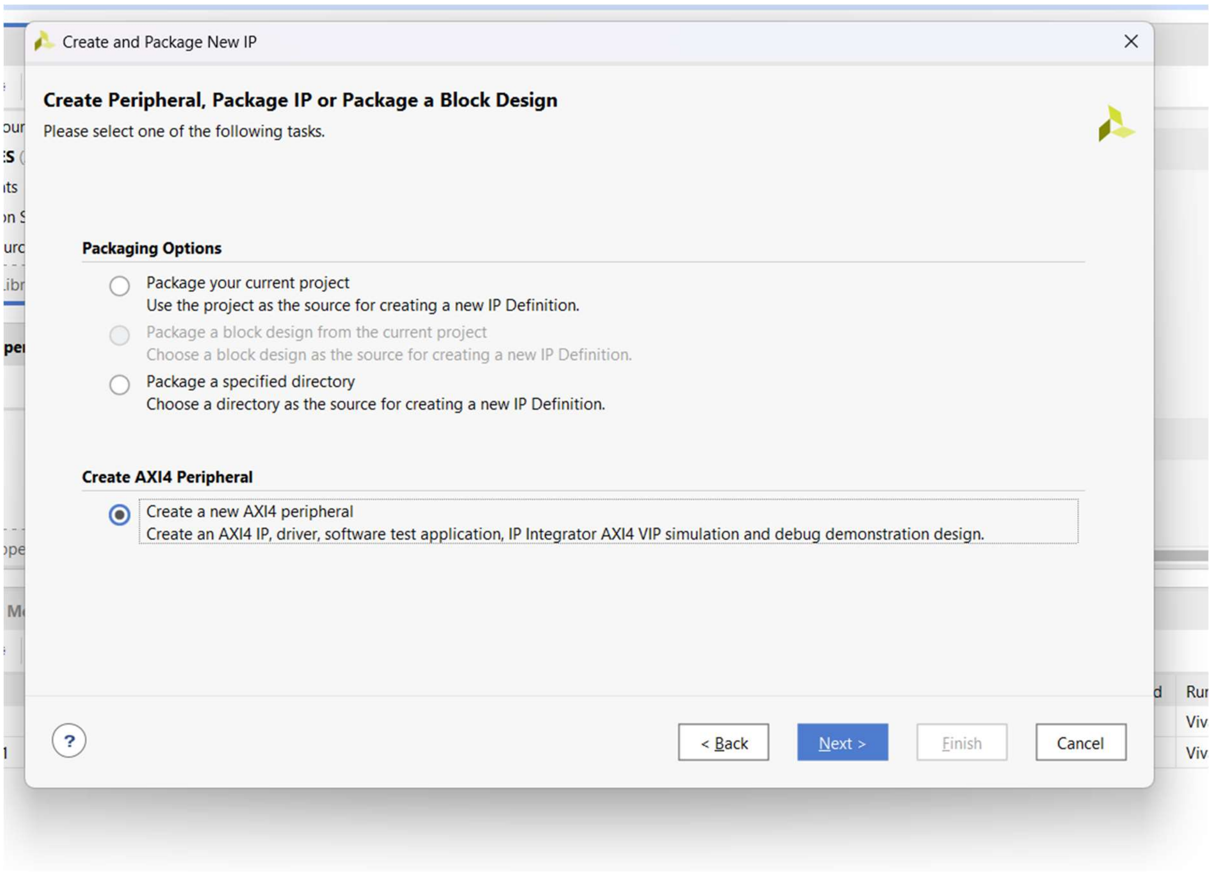
Step 1. Create your own vivado project in Verilog or VHDL by selecting the desired Board in the Boards/Parts section.

Step2. Tools > Create and Package new IP



Location: F:/Vivado Projects/AES_Verilog/AES_Verilog.srcs/sc

Step 3. Select AXI Peripherals



Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

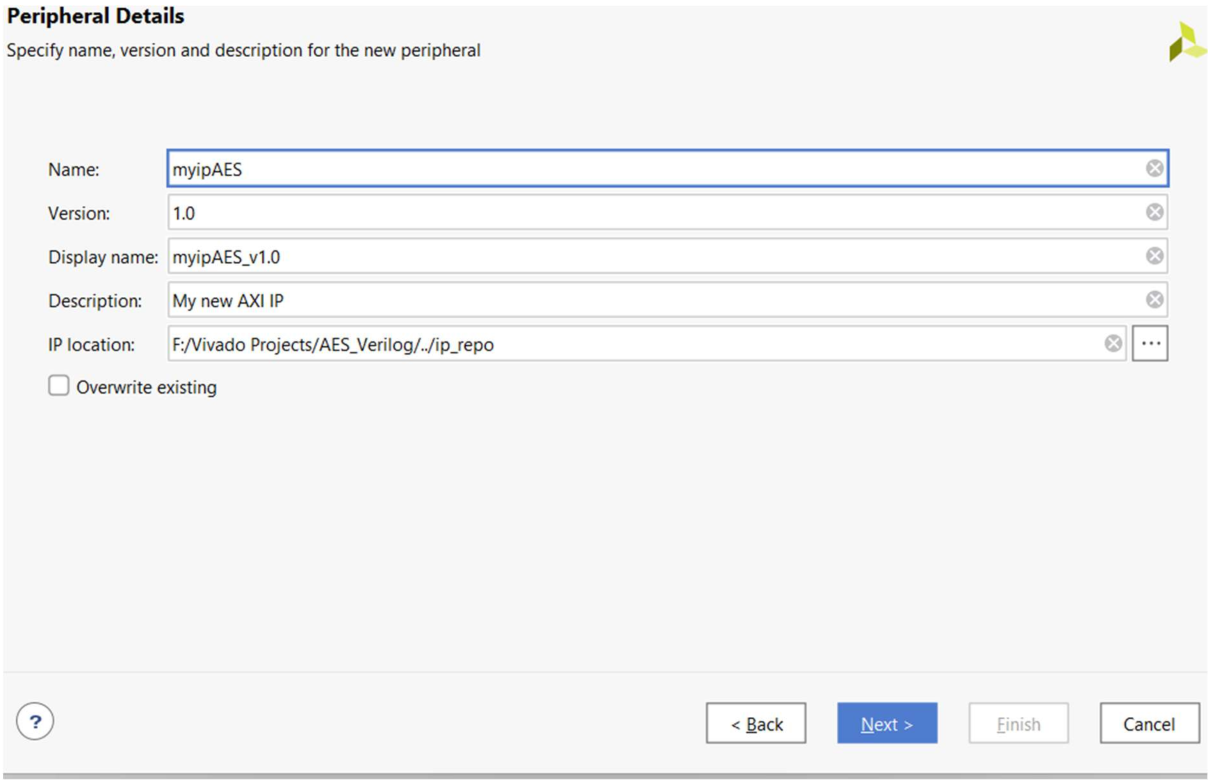
Packaging Options

- Package your current project
  Use the project as the source for creating a new IP Definition.
- Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
- Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

- [✓ Create a new AXI4 peripheral]
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

Step 4. Name your IP



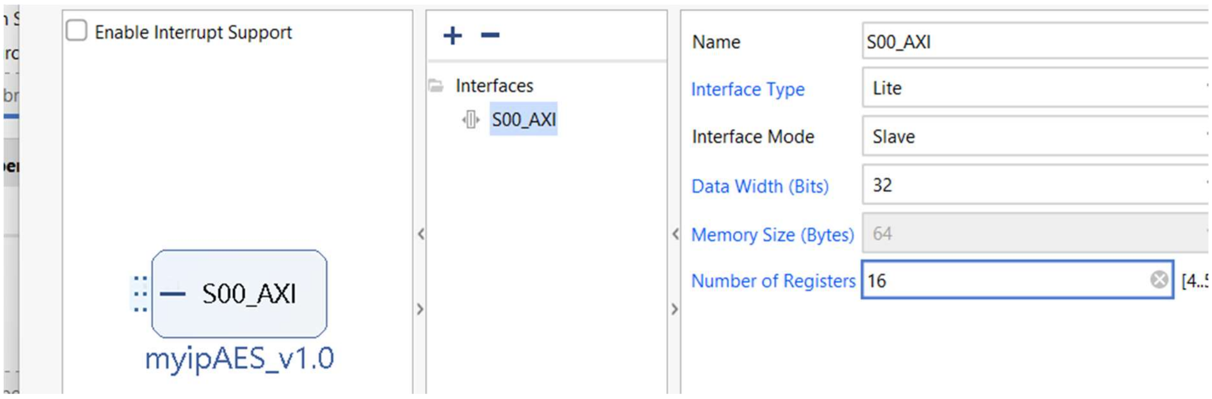
Peripheral Details

Specify name, version and description for the new peripheral

Name:	myipAES
Version:	1.0
Display name:	myipAES_v1.0
Description:	My new AXI IP
IP location:	F:/Vivado Projects/AES_Verilog/../ip_repo

☐ Overwrite existing

Step 5. Select Required number of Registers and their size

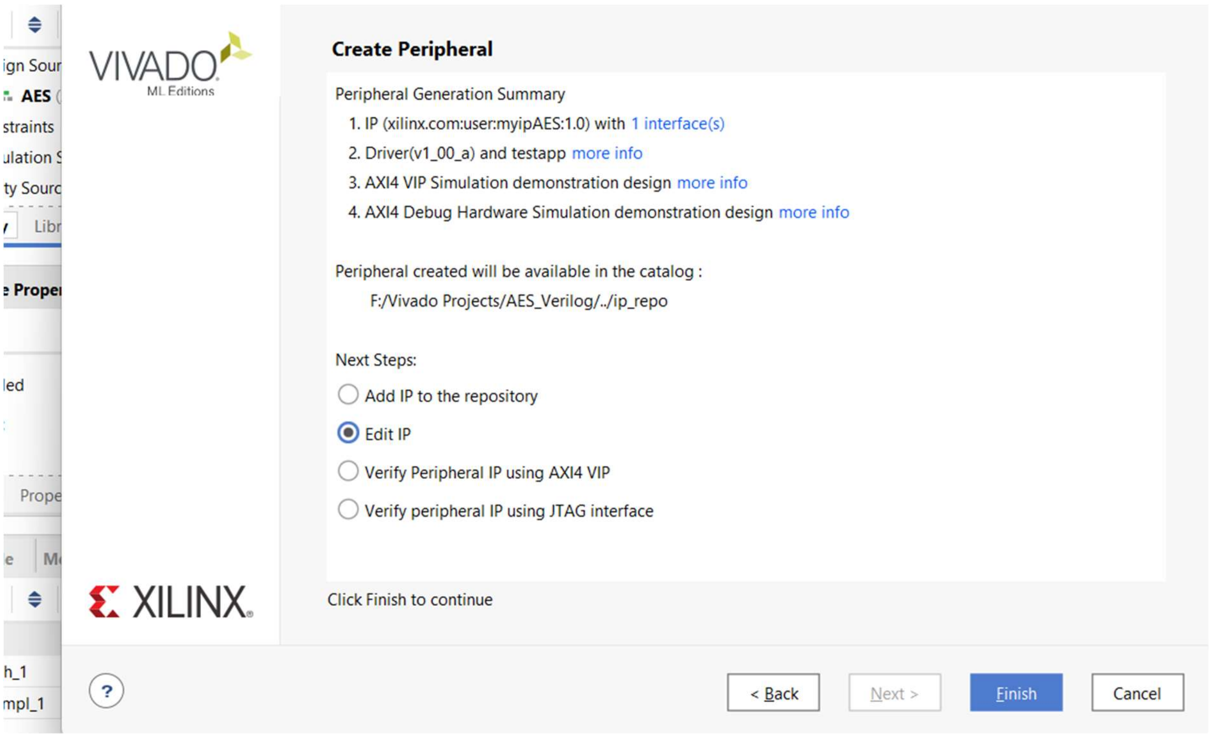


Enable Interrupt Support

Interfaces
S00_AXI

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	16

Step 6. Edit IP



Create Peripheral

Peripheral Generation Summary

1. IP (xilinx.com:user:myipAES:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

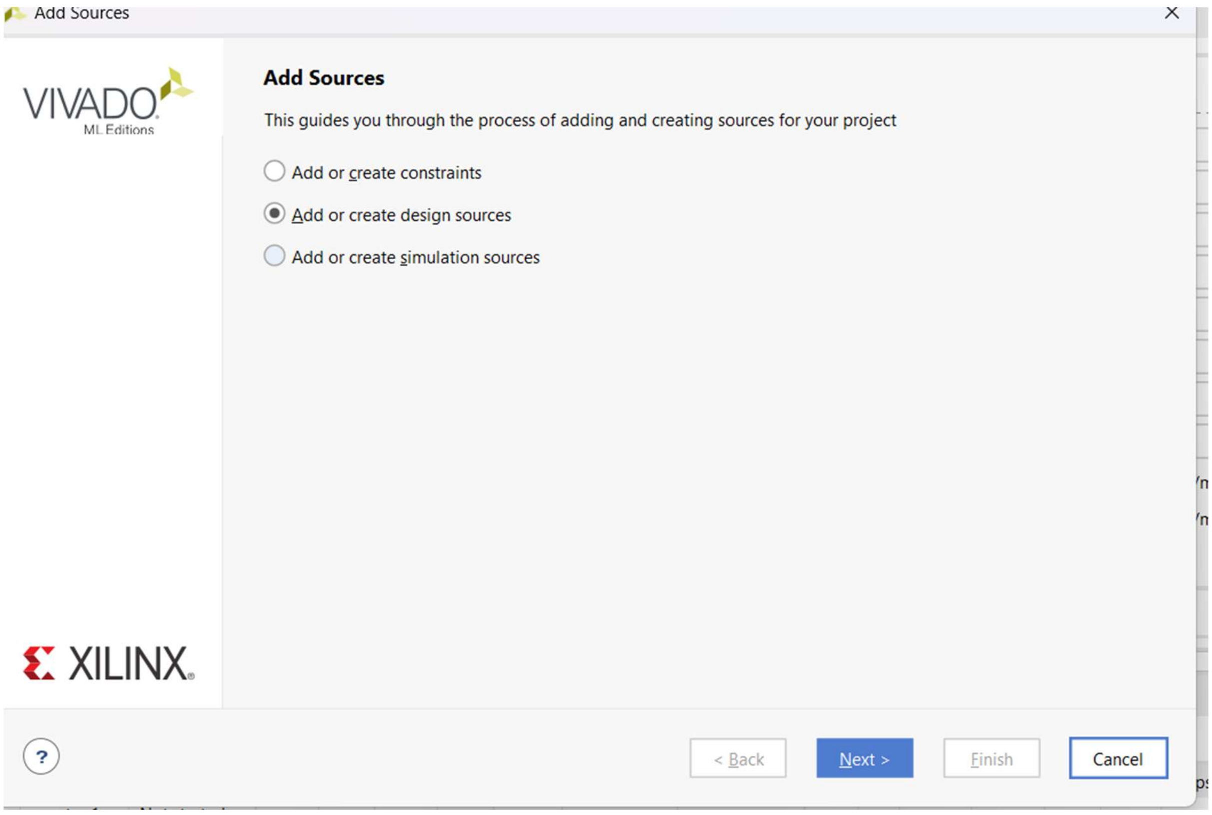
Peripheral created will be available in the catalog :
F:/Vivado Projects/AES_Verilog/../ip_repo

Next Steps:

- ○ Add IP to the repository
- [✓ Edit IP]
- ○ Verify Peripheral IP using AXI4 VIP
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

Step 7. Add all the design sources of your original Vivado project

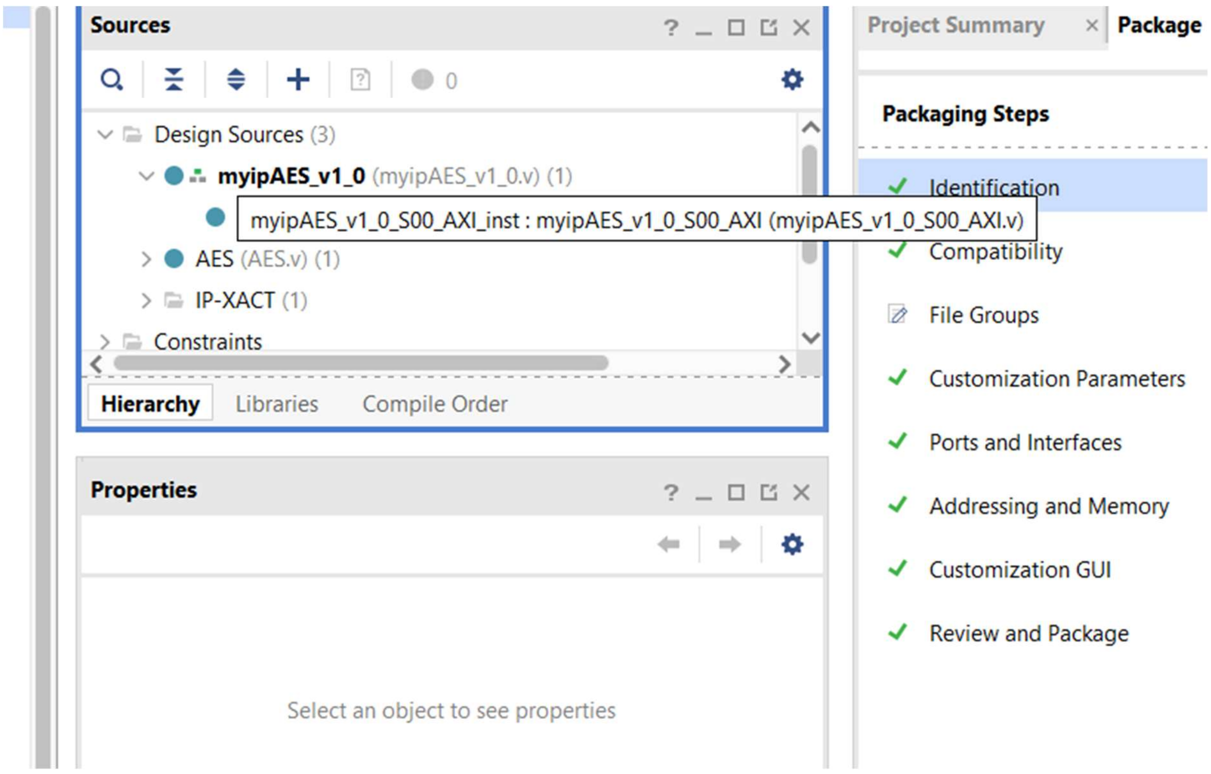


Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
[✓ Add or create design sources]
( ) Add or create simulation sources

Step 8. Open the inner file of the Top File

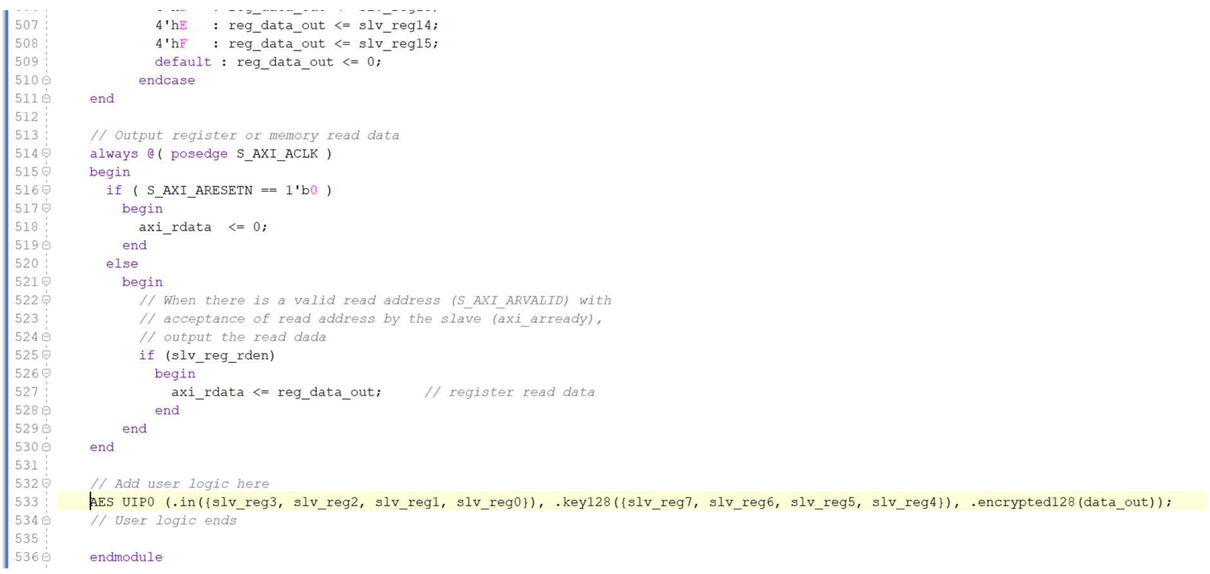


Design Sources (3)
myipAES_v1_0 (myipAES_v1_0.v) (1)
myipAES_v1_0_S00_AXI_inst : myipAES_v1_0_S00_AXI (myipAES_v1_0_S00_AXI.v)
AES (AES.v) (1)
IP-XACT (1)
Constraints

Packaging Steps

✓ Identification
✓ Compatibility
File Groups
✓ Customization Parameters
✓ Ports and Interfaces
✓ Addressing and Memory
✓ Customization GUI
✓ Review and Package

Step 9. Scroll down and instantiate your own vivado project with relevant slave registers and assign new wired vector datatype to the output variable.



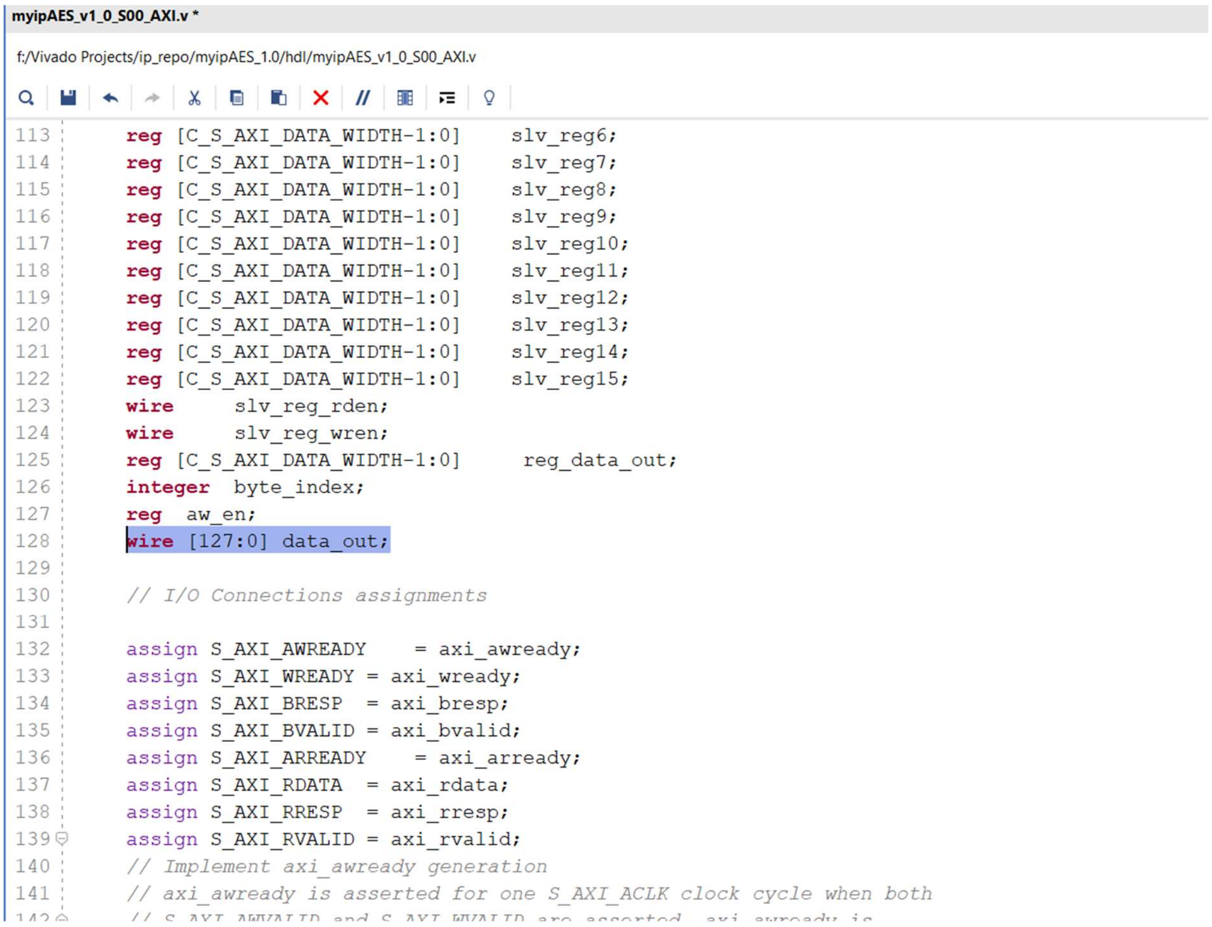
```verilog
          4'hE   : reg_data_out <= slv_reg14;
          4'hF   : reg_data_out <= slv_reg15;
          default : reg_data_out <= 0;
        endcase
    end

    // Output register or memory read data
    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_rdata  <= 0;
        end
      else
        begin
          // When there is a valid read address (S_AXI_ARVALID) with
          // acceptance of read address by the slave (axi_arready),
          // output the read dada
          if (slv_reg_rden)
            begin
              axi_rdata <= reg_data_out;     // register read data
            end
        end
    end

    // Add user logic here
    AES UIP0 (.in({slv_reg3, slv_reg2, slv_reg1, slv_reg0}), .key128({slv_reg7, slv_reg6, slv_reg5, slv_reg4}), .encrypted128(data_out));
    // User logic ends

    endmodule
```

Step 10. Scroll up and add that wired variable



myipAES_v1_0_S00_AXI.v *

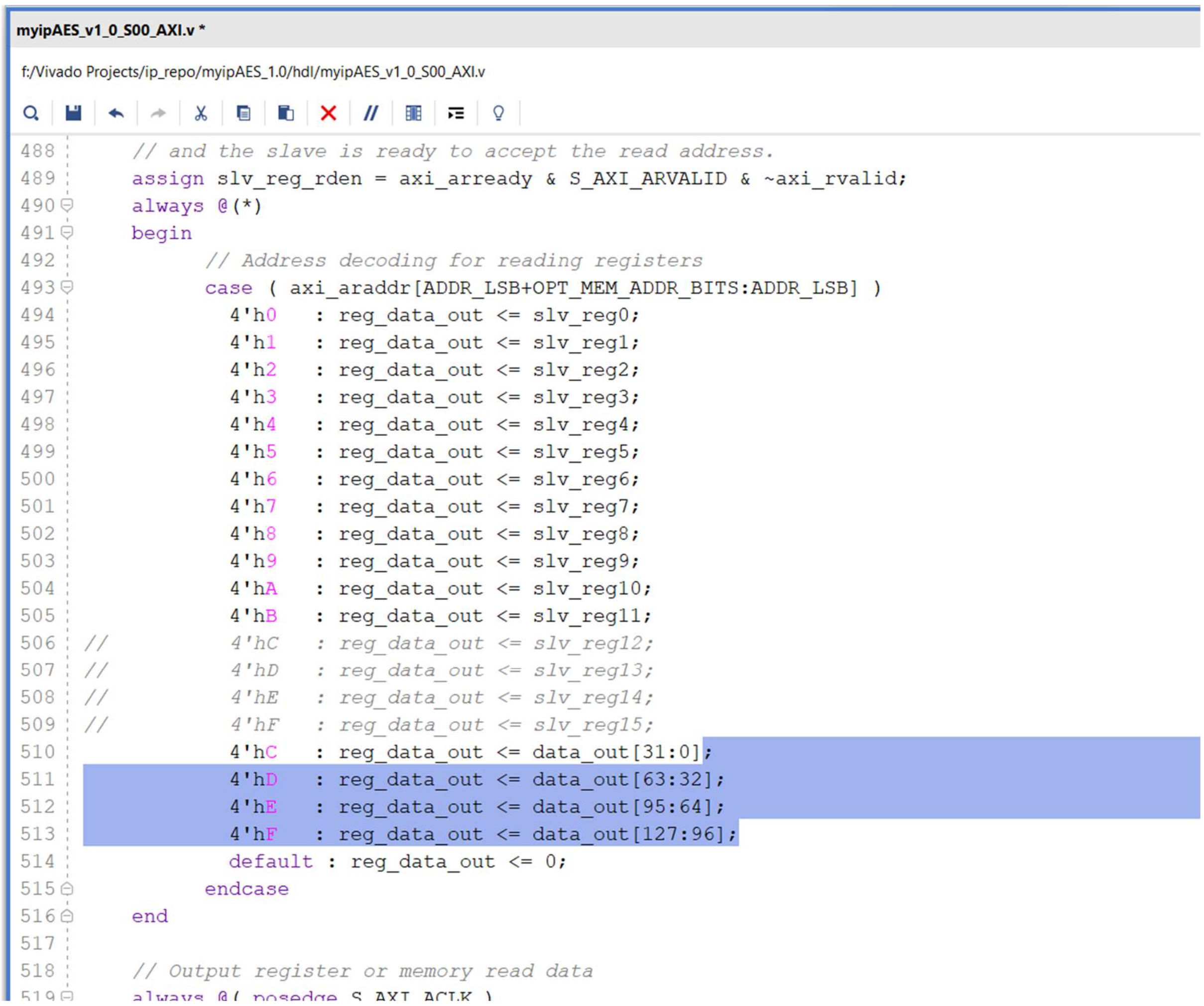
f:/Vivado Projects/ip_repo/myipAES_1.0/hdl/myipAES_v1_0_S00_AXI.v

```verilog
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg6;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg7;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg8;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg9;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg10;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg11;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg12;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg13;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg14;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg15;
    wire     slv_reg_rden;
    wire     slv_reg_wren;
    reg [C_S_AXI_DATA_WIDTH-1:0]     reg_data_out;
    integer  byte_index;
    reg  aw_en;
    wire [127:0] data_out;

    // I/O Connections assignments

    assign S_AXI_AWREADY    = axi_awready;
    assign S_AXI_WREADY = axi_wready;
    assign S_AXI_BRESP  = axi_bresp;
    assign S_AXI_BVALID = axi_bvalid;
    assign S_AXI_ARREADY    = axi_arready;
    assign S_AXI_RDATA  = axi_rdata;
    assign S_AXI_RRESP  = axi_rresp;
    assign S_AXI_RVALID = axi_rvalid;
    // Implement axi_awready generation
    // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
```

Step 11. Assign some address to your output variable appropriately.



myipAES_v1_0_S00_AXI.v *

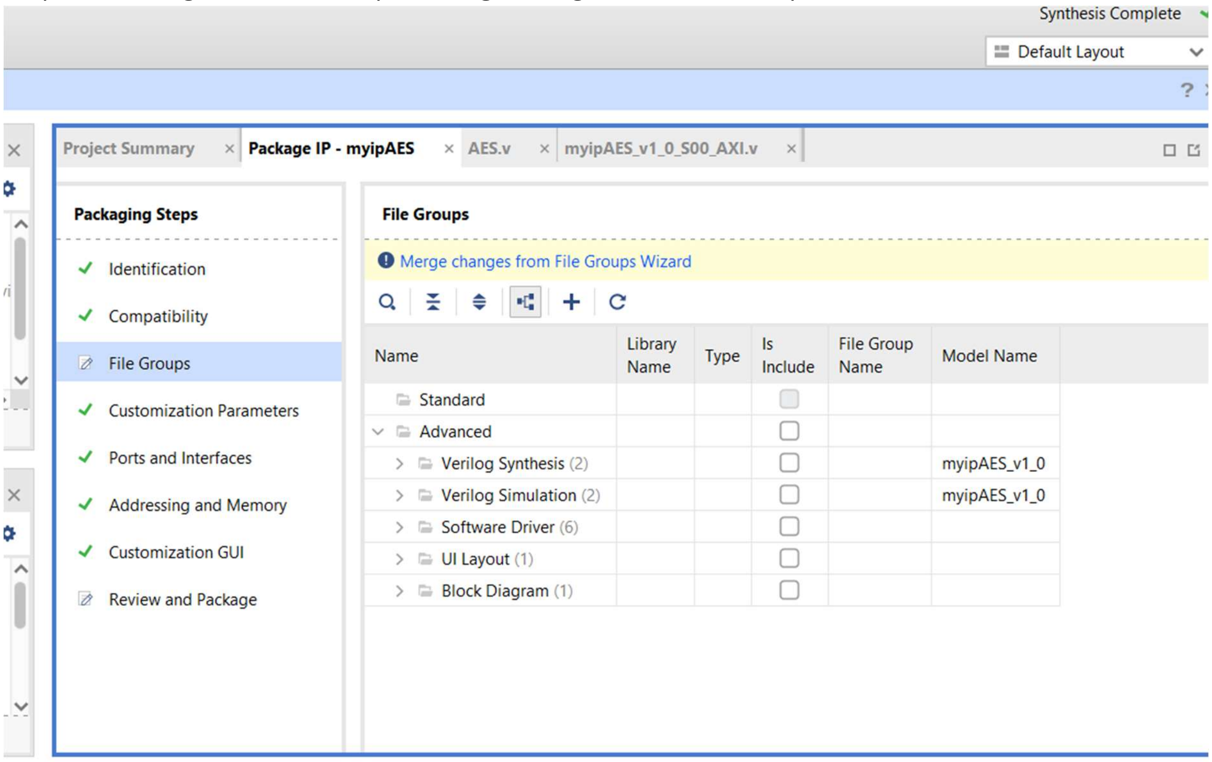
f:/Vivado Projects/ip_repo/myipAES_1.0/hdl/myipAES_v1_0_S00_AXI.v

```
    // and the slave is ready to accept the read address.
    assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
    always @(*)
    begin
          // Address decoding for reading registers
          case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
            4'h0   : reg_data_out <= slv_reg0;
            4'h1   : reg_data_out <= slv_reg1;
            4'h2   : reg_data_out <= slv_reg2;
            4'h3   : reg_data_out <= slv_reg3;
            4'h4   : reg_data_out <= slv_reg4;
            4'h5   : reg_data_out <= slv_reg5;
            4'h6   : reg_data_out <= slv_reg6;
            4'h7   : reg_data_out <= slv_reg7;
            4'h8   : reg_data_out <= slv_reg8;
            4'h9   : reg_data_out <= slv_reg9;
            4'hA   : reg_data_out <= slv_reg10;
            4'hB   : reg_data_out <= slv_reg11;
//          4'hC   : reg_data_out <= slv_reg12;
//          4'hD   : reg_data_out <= slv_reg13;
//          4'hE   : reg_data_out <= slv_reg14;
//          4'hF   : reg_data_out <= slv_reg15;
            4'hC   : reg_data_out <= data_out[31:0];
            4'hD   : reg_data_out <= data_out[63:32];
            4'hE   : reg_data_out <= data_out[95:64];
            4'hF   : reg_data_out <= data_out[127:96];
            default : reg_data_out <= 0;
          endcase
    end

    // Output register or memory read data
    always @( posedge S_AXI_ACLK )
```

Step 12. Now check correctness of your code by synthesizing it

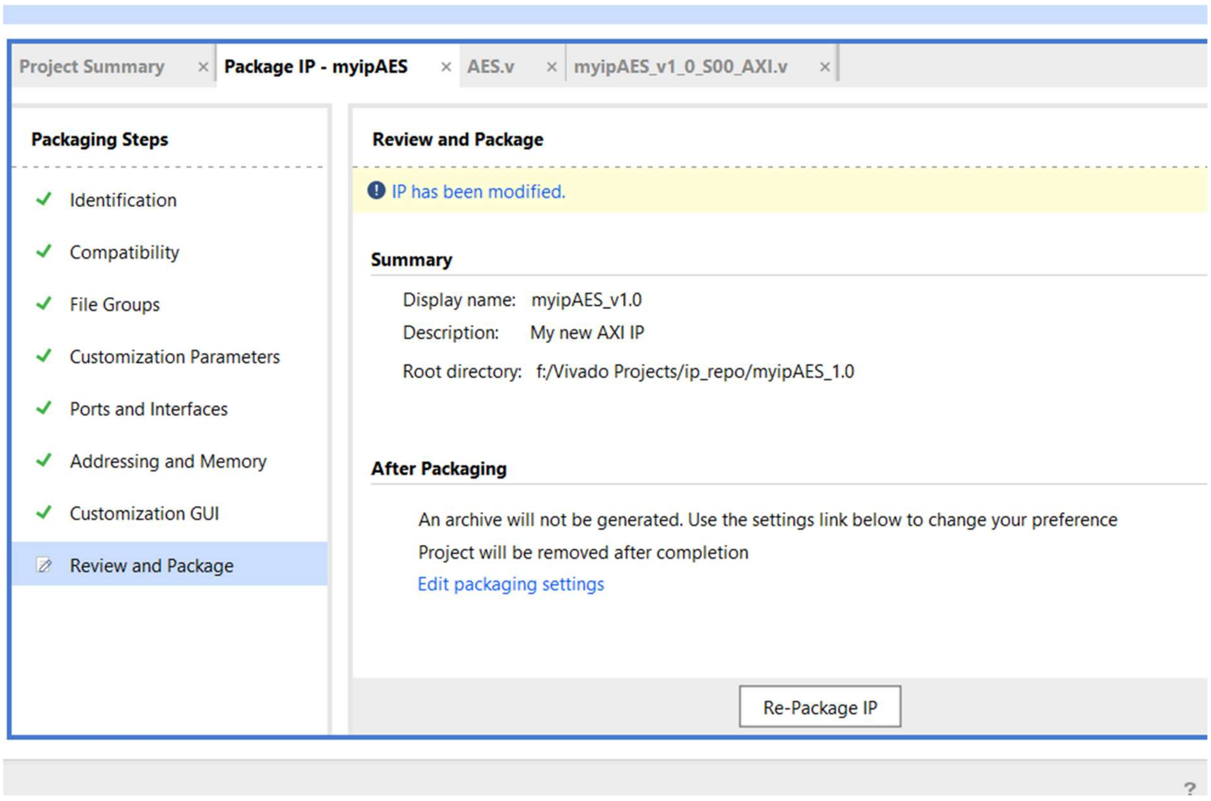
Step 13. Package IP > File Groups > Merge changes from File Groups Wizard



Merge changes from File Groups Wizard

Name	Library Name	Type	Is Include	File Group Name	Model Name
Standard			☐		
Advanced			☐		
Verilog Synthesis (2)			☐		myipAES_v1_0
Verilog Simulation (2)			☐		myipAES_v1_0
Software Driver (6)			☐		
UI Layout (1)			☐		
Block Diagram (1)			☐		

Step 14. Review and Package > Re-Package IP



IP has been modified.

Summary

Display name: myipAES_v1.0
Description: My new AXI IP
Root directory: f:/Vivado Projects/ip_repo/myipAES_1.0

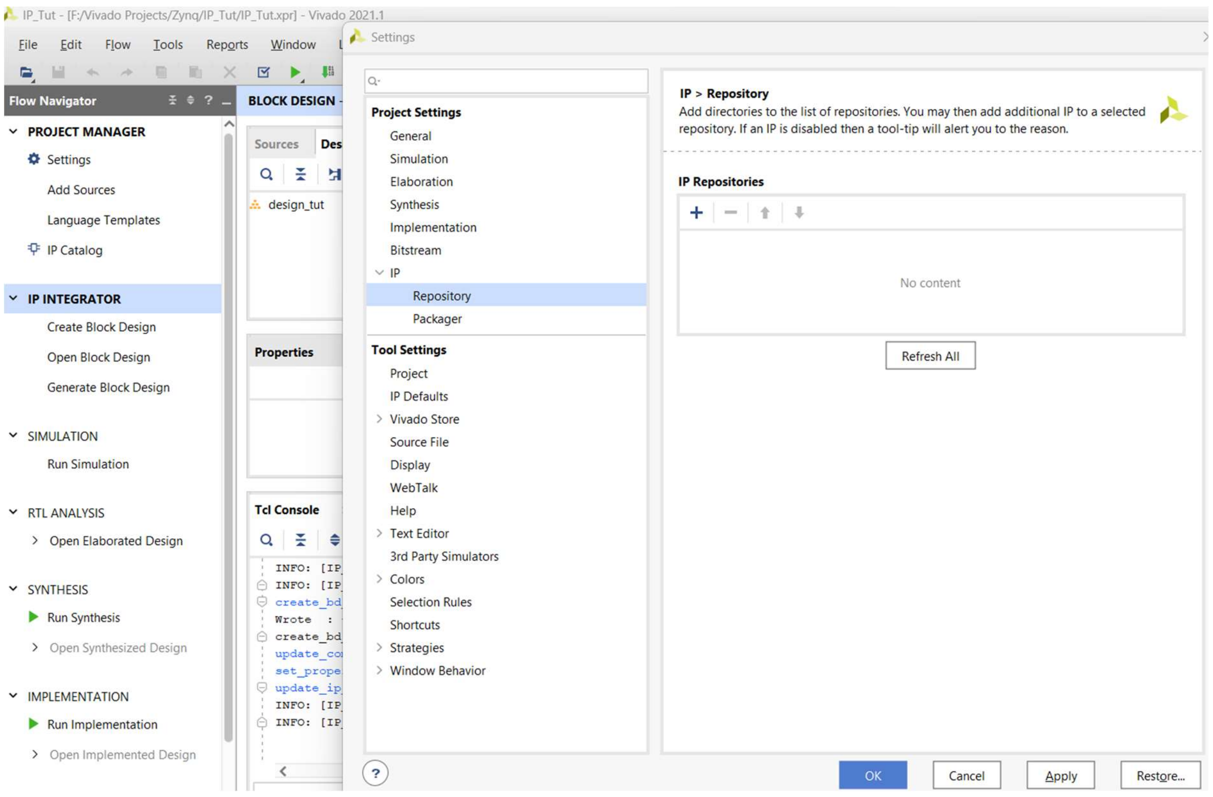
After Packaging

An archive will not be generated. Use the settings link below to change your preference
Project will be removed after completion
Edit packaging settings

That’s it You are done.

Now we are left to integrate the IP with rest of the important blocks.

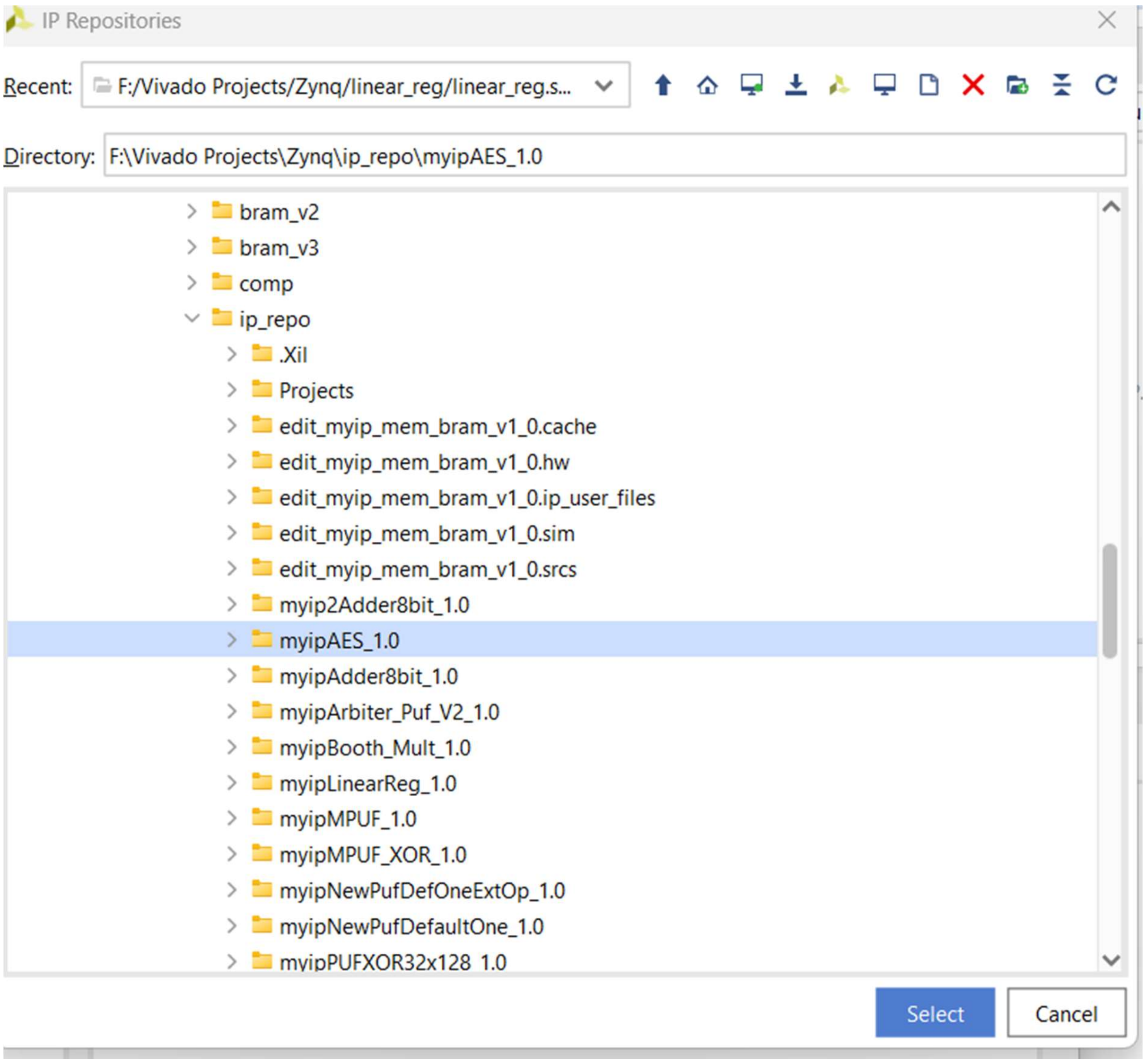
Step 15. Go to the settings > IP > Repository



IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

Step 16. Click on '+' icon > Select 'ip_repo' > select the folder of your IP [myipAES_1.0]


IP Repositories

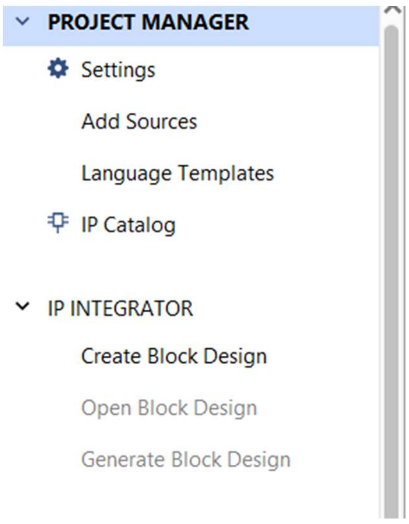
Recent: F:/Vivado Projects/Zynq/linear_reg/linear_reg.s...

Directory: F:\Vivado Projects\Zynq\ip_repo\myipAES_1.0

bram_v2
bram_v3
comp
ip_repo
.Xil
Projects
edit_myip_mem_bram_v1_0.cache
edit_myip_mem_bram_v1_0.hw
edit_myip_mem_bram_v1_0.ip_user_files
edit_myip_mem_bram_v1_0.sim
edit_myip_mem_bram_v1_0.srcs
myip2Adder8bit_1.0
myipAES_1.0
myipAdder8bit_1.0
myipArbiter_Puf_V2_1.0
myipBooth_Mult_1.0
myipLinearReg_1.0
myipMPUF_1.0
myipMPUF_XOR_1.0
myipNewPufDefOneExtOp_1.0
myipNewPufDefaultOne_1.0
myipPUFXOR32x128_1.0

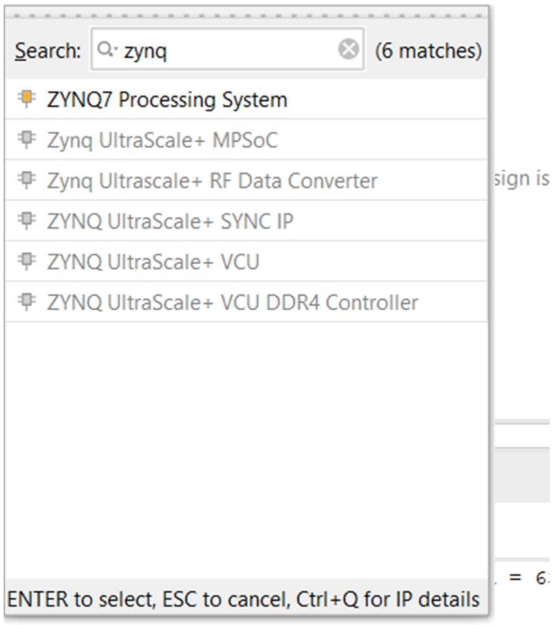
Step 17. OK > Apply > OK //Ip has been added to the repository

Step 18. Create Block Diagram



Step 19. Click on '+' icon to add IPs to the BD

Step 20. Search for 'Zynq Processor System' and add that using 'Enter'



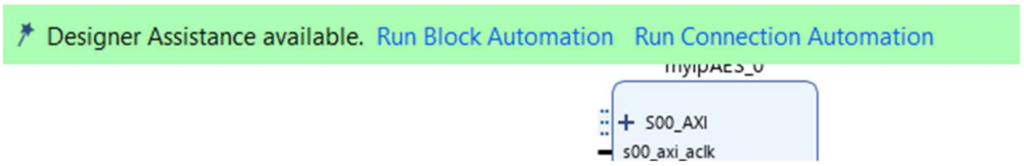
Search: zynq (6 matches)

- ZYNQ7 Processing System
- Zynq UltraScale+ MPSoC
- Zynq Ultrascale+ RF Data Converter
- ZYNQ UltraScale+ SYNC IP
- ZYNQ UltraScale+ VCU
- ZYNQ UltraScale+ VCU DDR4 Controller

ENTER to select, ESC to cancel, Ctrl+Q for IP details

Step 20. Similarly, search for your IPs to be integrated in the BD

Step 21. Click on the following blue options one after another, it will create necessary connections.



Designer Assistance available. Run Block Automation Run Connection Automation

Step 22. For mor customizations, double click on the block and customize options and connections.

Step 23. Click , it will regenerate layout with better view

Step 24. Click , it will validate the design.

Step 25. Go to Sources > your bd > Right Click > Generate Output Products > Use global option

Step 26. Similarly, Create HDL wrapper, and set the design as top

Step 27. Now Generate the bistreams.

Step 28. Go to 'File' > 'Export Hardware' > next > include bitsream > set the name of the wrapper > Finish

Your Design is exported. Now can be used using VITIS