Lab 11

Tasks to be done in this Lab:

1. Create a custom AXI4 Lite IP to perform the following operation

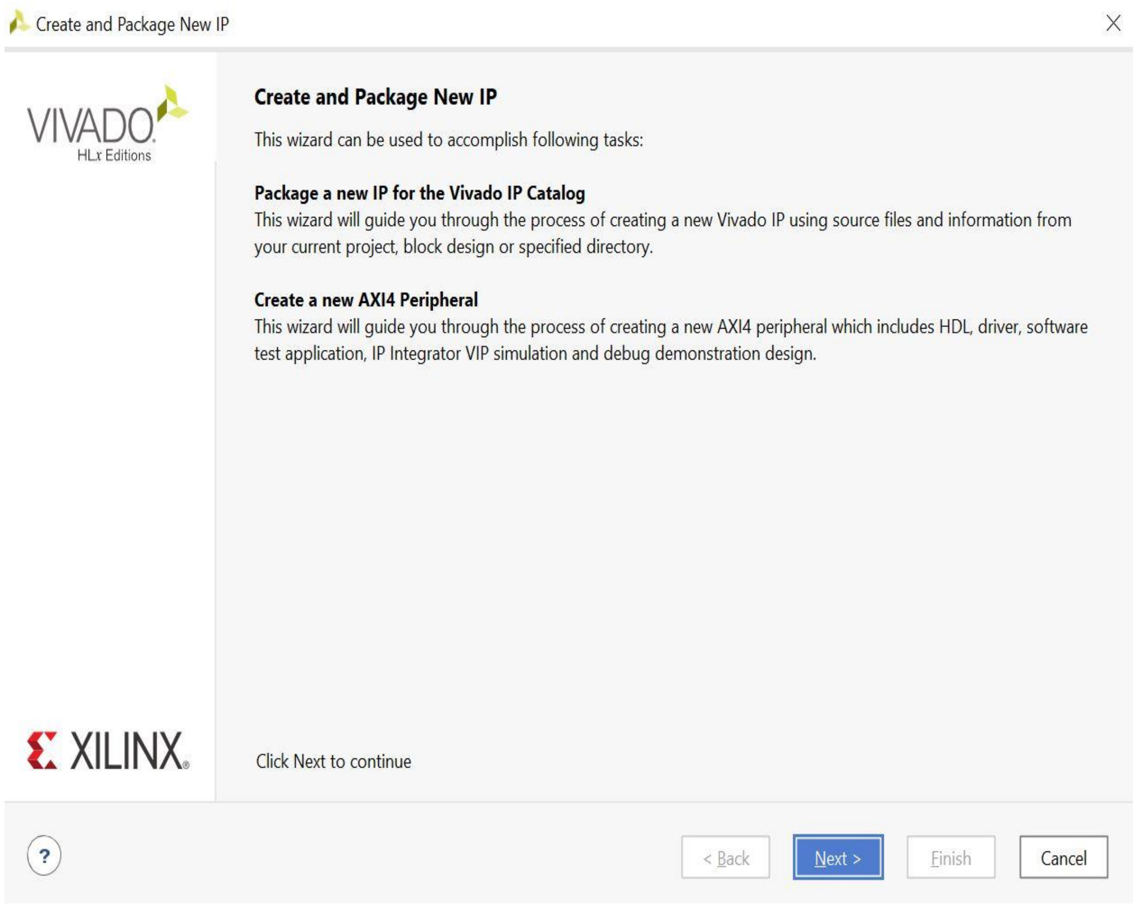
   Q= X/T + sqrt(α ln(N)/T)

2. Test the functionality using the testbench

Topics to explore: 1) Custom AXI IP Creation

Part -1

1. Create a new Vivado Project.
2. At the next window, tick "Do not specify sources at this time."
3. In the board selection window, select Zybo Z7-10 and click on Finish in the next window.
4. Go to **Tools->Create and Package New IP.**



Create and Package New IP

Create and Package New IP

This wizard can be used to accomplish following tasks:

Package a new IP for the Vivado IP Catalog

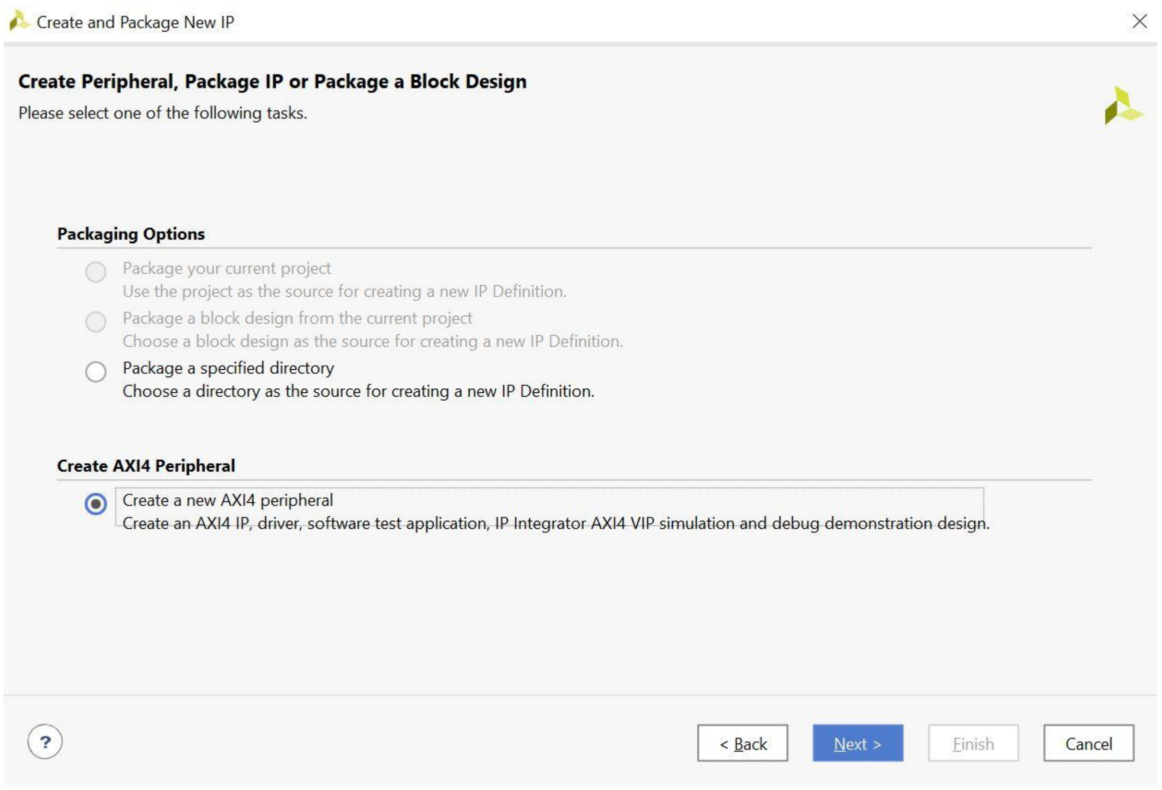
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral

This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Click Next to continue

5. Click on Next.
6. In the next window, select the “create a new AXI4 Peripheral Option” and click on Next.


Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

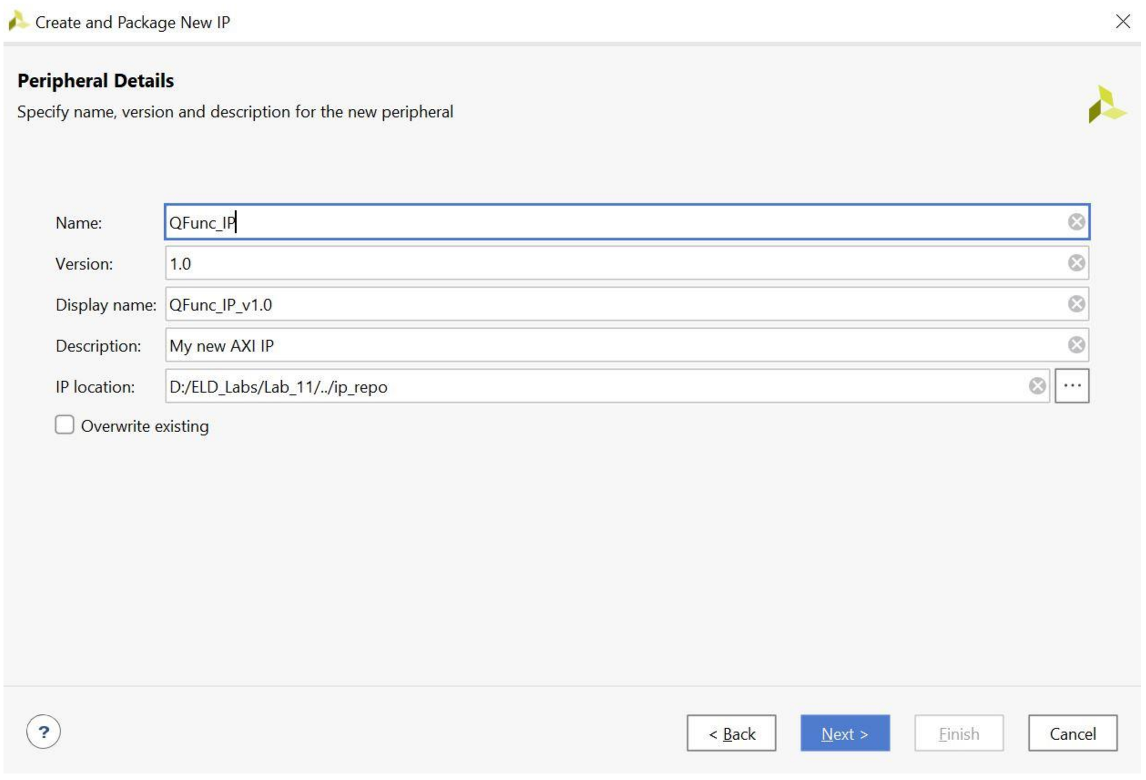
Packaging Options

- 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.

7. Add the name of yor IP and see the IP Location (A folder with the name ip_repo will be created in your working directory).The version of the IP will change whenever you will edit and repackage the IP in future


Create and Package New IP

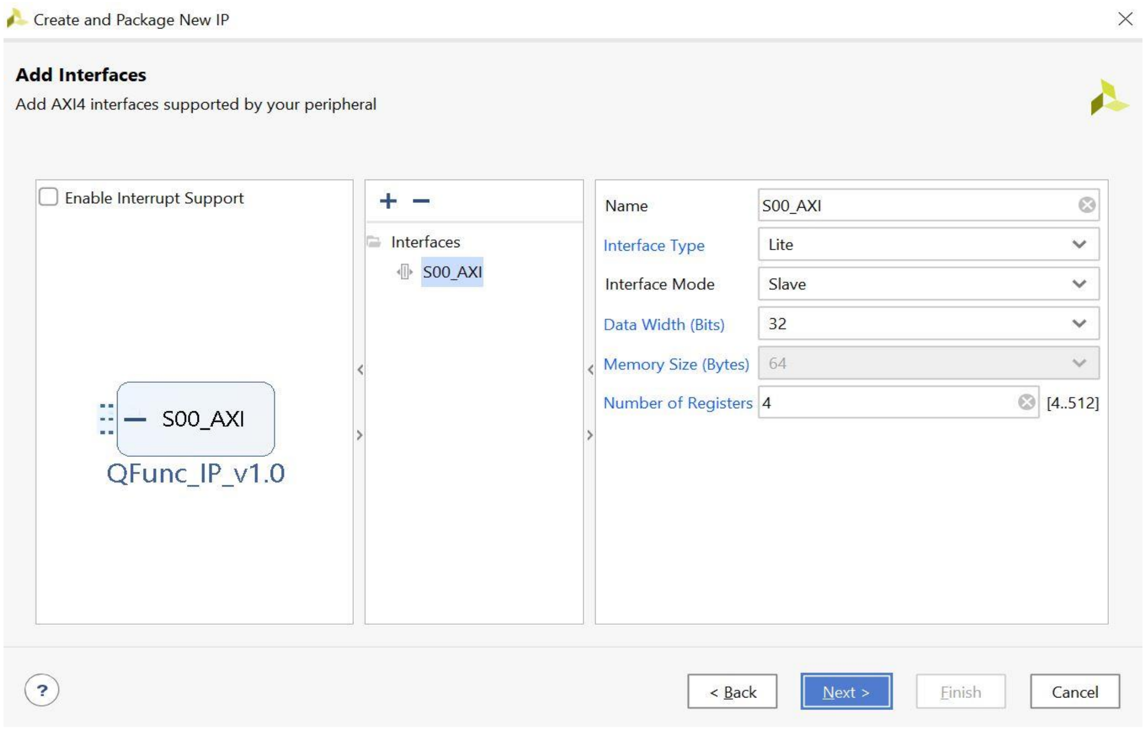
Peripheral Details

Specify name, version and description for the new peripheral

Name:	QFunc_IP
Version:	1.0
Display name:	QFunc_IP_v1.0
Description:	My new AXI IP
IP location:	D:/ELD_Labs/Lab_11/../ip_repo

☐ Overwrite existing

8. In the next window, keep the default option. You will be briefed about these options in the Lecture.



Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

Enable Interrupt Support [ ]

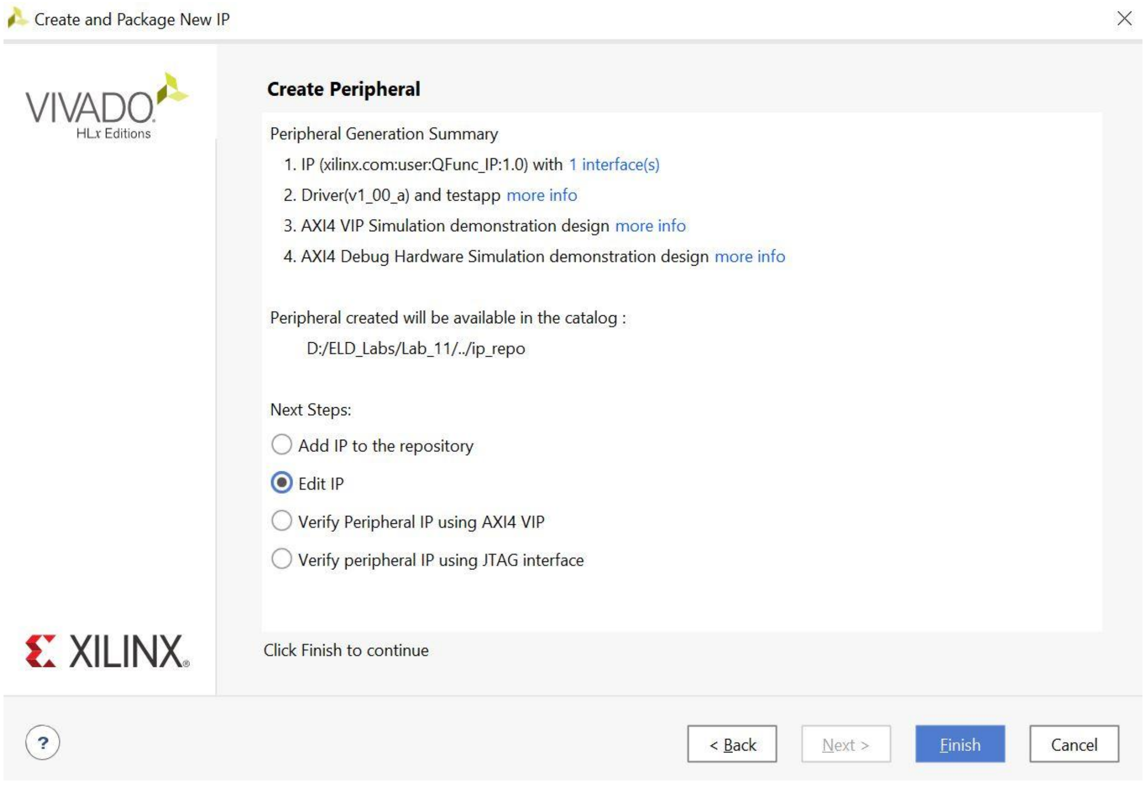
S00_AXI

QFunc_IP_v1.0

Interfaces
- S00_AXI

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

9. In the next window, select the edit IP option and click on Finish.



Create and Package New IP

VIVADO HLx Editions

Create Peripheral

Peripheral Generation Summary

1. IP (xilinx.com:user:QFunc_IP: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 :

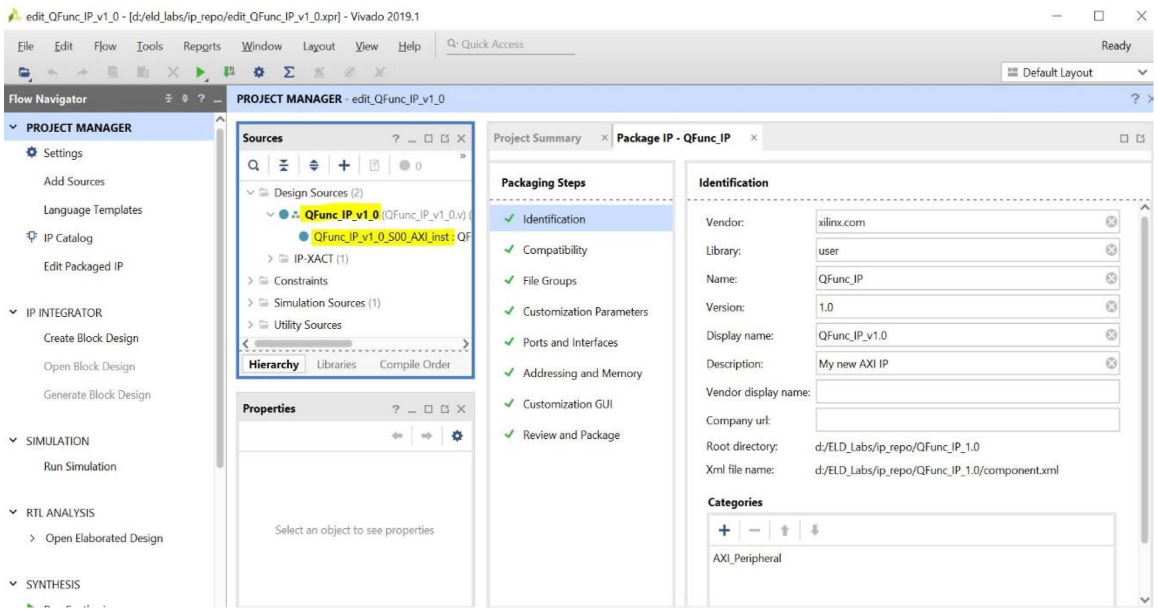
D:/ELD_Labs/Lab_11/../ip_repo

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

10. In the edit IP window you will see two Verilog files already created with the default codes for the AXI Lite. The top-level file contains a module which implements the AXI4-LITE interfacing logic, and an example design to write to and read from the number of registers we specified. We will make changes in these codes to customize the IP according to out needs.


edit_QFunc_IP_v1_0 - [d:/eld_labs/ip_repo/edit_QFunc_IP_v1_0.xpr] - Vivado 2019.1

PROJECT MANAGER - edit_QFunc_IP_v1_0

Sources: Design Sources (2)
- QFunc_IP_v1_0 (QFunc_IP_v1_0.v)
  - QFunc_IP_v1_0_S00_AXI_inst : QF
- IP-XACT (1)

Constraints
Simulation Sources (1)
Utility Sources

Package IP - QFunc_IP

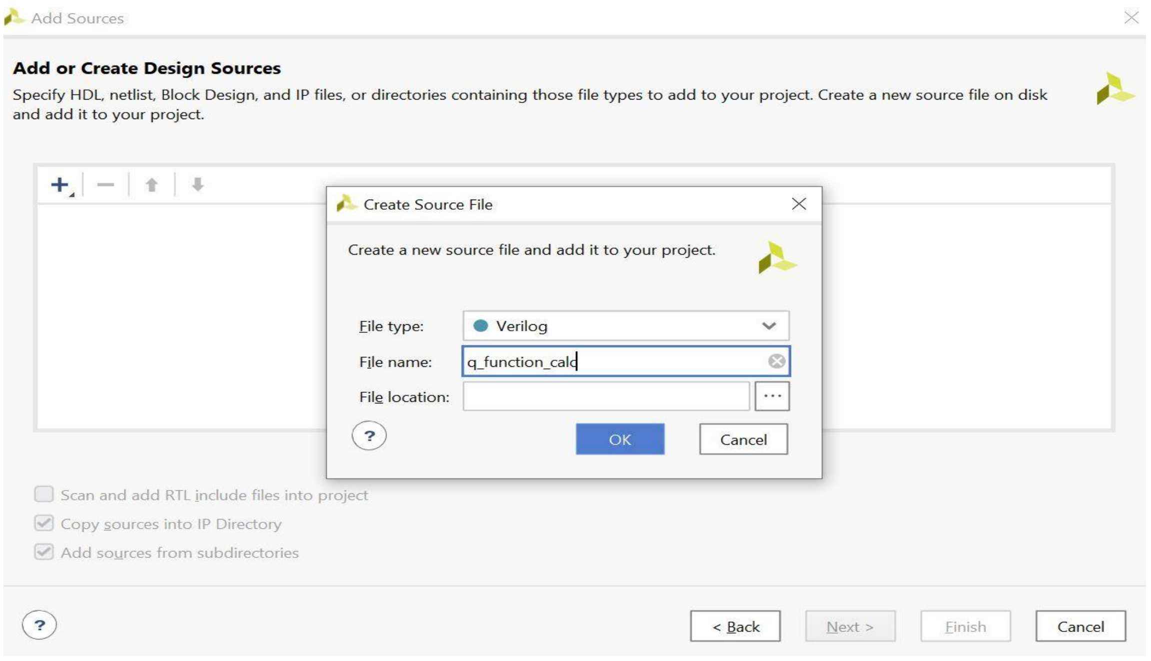
Packaging Steps: Identification, Compatibility, File Groups, Customization Parameters, Ports and Interfaces, Addressing and Memory, Customization GUI, Review and Package

Identification

Vendor:	xilinx.com
Library:	user
Name:	QFunc_IP
Version:	1.0
Display name:	QFunc_IP_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	d:/ELD_Labs/ip_repo/QFunc_IP_1.0
Xml file name:	d:/ELD_Labs/ip_repo/QFunc_IP_1.0/component.xml

Categories: AXI_Peripheral

11. Add a new Verilog source. This will contain the verilog code for the floating point operation.The code will written for the function Q= X/T + sqrt(alpha ln(N/T)). The code for this module is given below for your reference.Add the floating point IP with the required operation to your project.While adding the source, the File Location field will be empty. Enter the location of your IP in the ip_repo.


Add Sources

Add or Create Design Sources

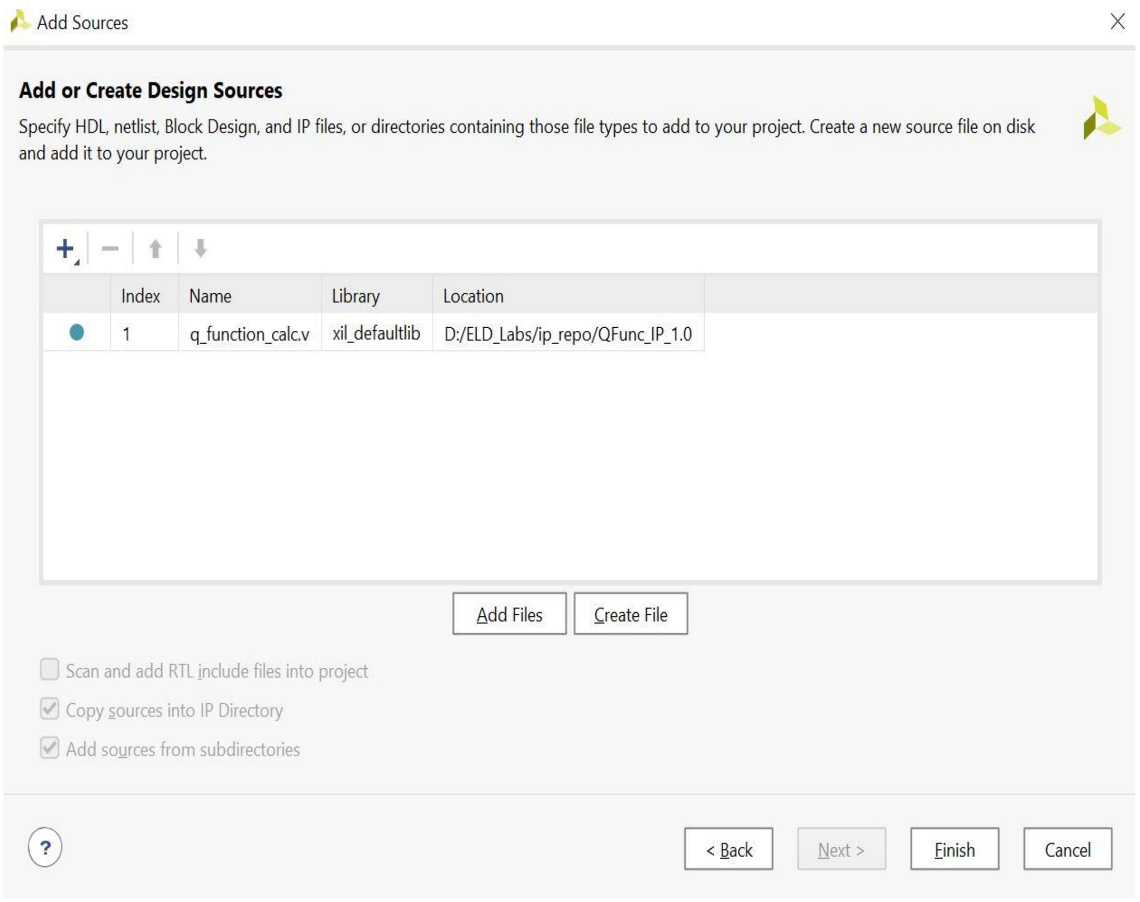
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

File type: Verilog
File name: q_function_calc
File location:

[ ] Scan and add RTL include files into project
[✓] Copy sources into IP Directory
[✓] Add sources from subdirectories



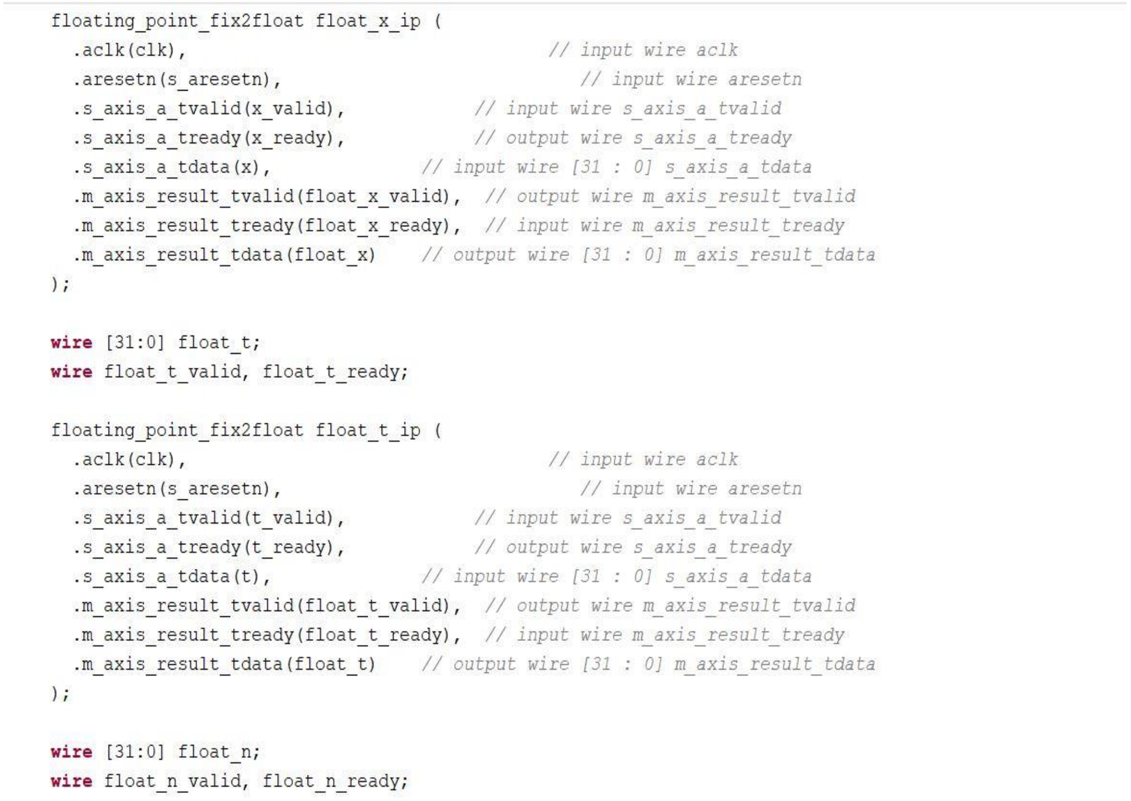
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

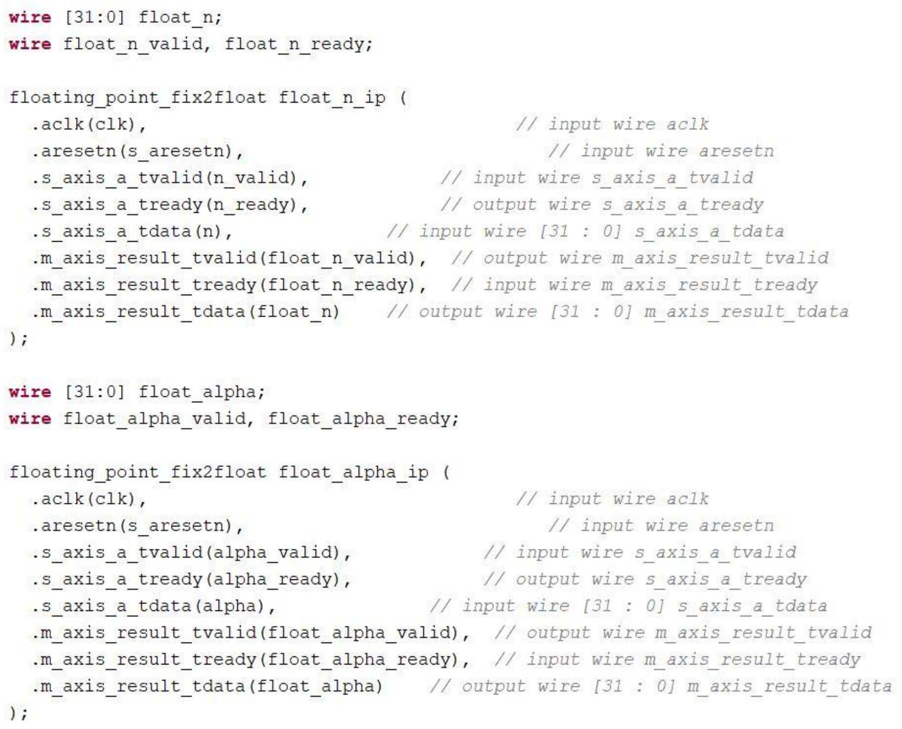
	Index	Name	Library	Location
●	1	q_function_calc.v	xil_defaultlib	D:/ELD_Labs/ip_repo/QFunc_IP_1.0

☐ Scan and add RTL include files into project
☑ Copy sources into IP Directory
☑ Add sources from subdirectories

```
`timescale 1ns / 1ps

module q_function_calc(
    input clk,
    input s_aresetn,
    input [31:0] alpha,
    input alpha_valid,
    output alpha_ready,
    input [31:0] x,
    input x_valid,
    output x_ready,
    input [31:0] t,
    input t_valid,
    output t_ready,
    input [31:0] n,
    input n_valid,
    output n_ready,
    output [31:0] q,
    output q_valid
    );

    wire [31:0] float_x;
    wire float_x_valid, float_x_ready;
```



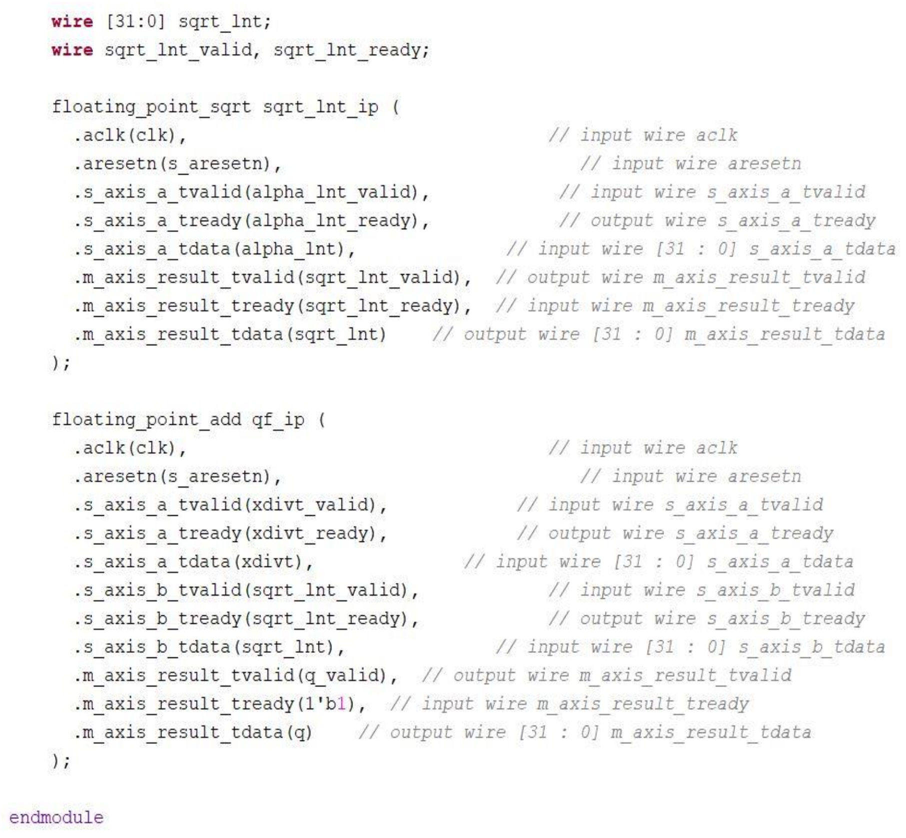
```verilog
floating_point_fix2float float_x_ip (
  .aclk(clk),                                  // input wire aclk
  .aresetn(s_aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(x_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(x_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(x),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(float_x_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(float_x_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(float_x)    // output wire [31 : 0] m_axis_result_tdata
);

wire [31:0] float_t;
wire float_t_valid, float_t_ready;

floating_point_fix2float float_t_ip (
  .aclk(clk),                                  // input wire aclk
  .aresetn(s_aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(t_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(t_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(t),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(float_t_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(float_t_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(float_t)    // output wire [31 : 0] m_axis_result_tdata
);

wire [31:0] float_n;
wire float_n_valid, float_n_ready;
```



```verilog
wire [31:0] float_n;
wire float_n_valid, float_n_ready;

floating_point_fix2float float_n_ip (
  .aclk(clk),                                  // input wire aclk
  .aresetn(s_aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(n_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(n_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(n),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(float_n_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(float_n_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(float_n)    // output wire [31 : 0] m_axis_result_tdata
);

wire [31:0] float_alpha;
wire float_alpha_valid, float_alpha_ready;

floating_point_fix2float float_alpha_ip (
  .aclk(clk),                                  // input wire aclk
  .aresetn(s_aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(alpha_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(alpha_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(alpha),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(float_alpha_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(float_alpha_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(float_alpha)    // output wire [31 : 0] m_axis_result_tdata
);
```

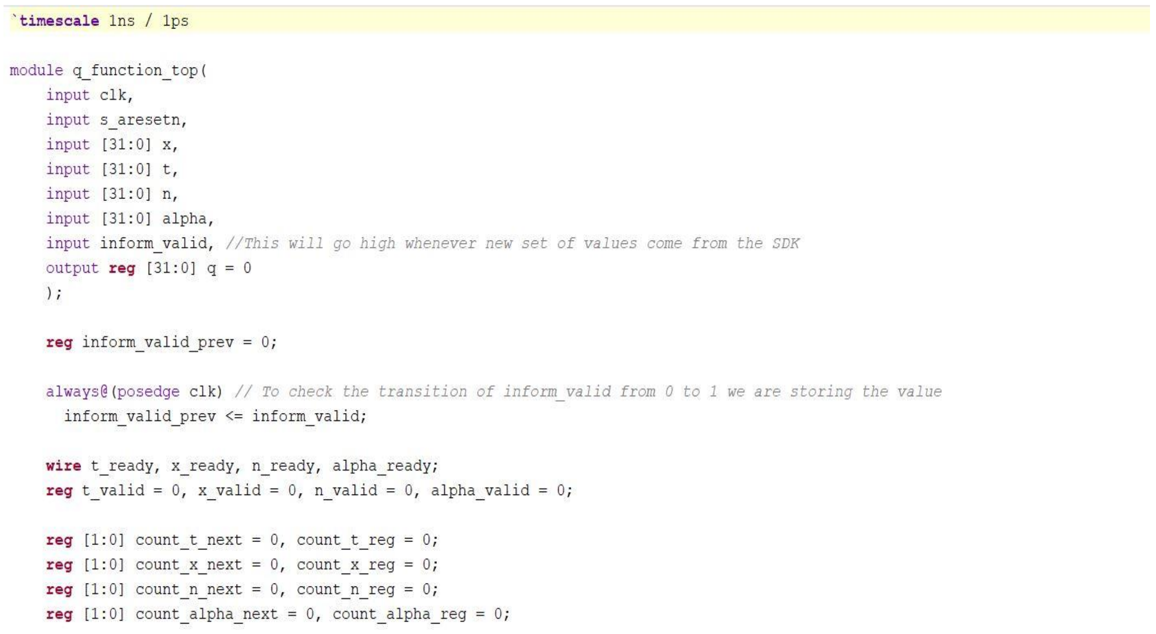
```
wire [31:0] xdivt;
wire xdivt_valid, xdivt_ready;

floating_point_div xdivt_ip (
  .aclk(clk),                                    // input wire aclk
  .aresetn(s_aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(float_x_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(float_x_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(float_x),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(float_t_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(float_t_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(float_t),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(xdivt_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(xdivt_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(xdivt)    // output wire [31 : 0] m_axis_result_tdata
);

wire [31:0] log_n;
wire log_n_valid, log_n_ready;

floating_point_ln log_n_ip (
  .aclk(clk),                                    // input wire aclk
  .aresetn(s_aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(float_n_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(float_n_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(float_n),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(log_n_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(log_n_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(log_n)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
  wire [31:0] log_n_byt;
  wire log_n_byt_valid, log_n_byt_ready;

  floating_point_div log_n_byt_ip (
    .aclk(clk),                                   // input wire aclk
    .aresetn(s_aresetn),                             // input wire aresetn
    .s_axis_a_tvalid(log_n_valid),              // input wire s_axis_a_tvalid
    .s_axis_a_tready(log_n_ready),              // output wire s_axis_a_tready
    .s_axis_a_tdata(log_n),                // input wire [31 : 0] s_axis_a_tdata
    .s_axis_b_tvalid(float_t_valid),             // input wire s_axis_b_tvalid
//      .s_axis_b_tready(float_t_ready),             // output wire s_axis_b_tready
    .s_axis_b_tdata(float_t),              // input wire [31 : 0] s_axis_b_tdata
    .m_axis_result_tvalid(log_n_byt_valid),  // output wire m_axis_result_tvalid
    .m_axis_result_tready(log_n_byt_ready),  // input wire m_axis_result_tready
    .m_axis_result_tdata(log_n_byt)    // output wire [31 : 0] m_axis_result_tdata
  );

  wire [31:0] alpha_lnt;
  wire alpha_lnt_valid, alpha_lnt_ready;

  floating_point_mul alpha_lnt_ip (
    .aclk(clk),                                   // input wire aclk
    .aresetn(s_aresetn),                             // input wire aresetn
    .s_axis_a_tvalid(float_alpha_valid),             // input wire s_axis_a_tvalid
    .s_axis_a_tready(float_alpha_ready),             // output wire s_axis_a_tready
    .s_axis_a_tdata(float_alpha),             // input wire [31 : 0] s_axis_a_tdata
    .s_axis_b_tvalid(log_n_byt_valid),            // input wire s_axis_b_tvalid
    .s_axis_b_tready(log_n_byt_ready),            // output wire s_axis_b_tready
    .s_axis_b_tdata(log_n_byt),             // input wire [31 : 0] s_axis_b_tdata
    .m_axis_result_tvalid(alpha_lnt_valid),  // output wire m_axis_result_tvalid
    .m_axis_result_tready(alpha_lnt_ready),  // input wire m_axis_result_tready
    .m_axis_result_tdata(alpha_lnt)    // output wire [31 : 0] m_axis_result_tdata
  );
```

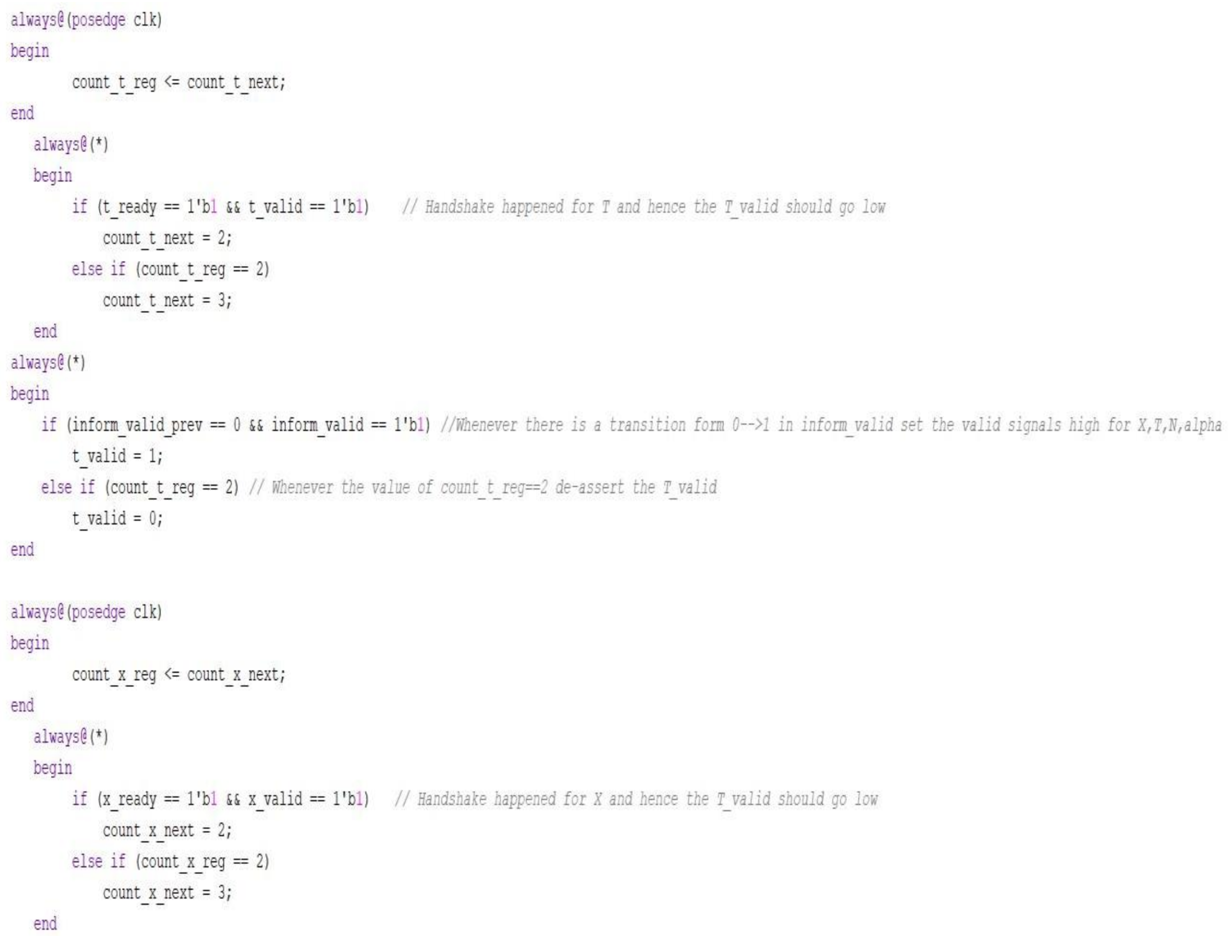


```
    wire [31:0] sqrt_lnt;
    wire sqrt_lnt_valid, sqrt_lnt_ready;

    floating_point_sqrt sqrt_lnt_ip (
      .aclk(clk),                                  // input wire aclk
      .aresetn(s_aresetn),                            // input wire aresetn
      .s_axis_a_tvalid(alpha_lnt_valid),            // input wire s_axis_a_tvalid
      .s_axis_a_tready(alpha_lnt_ready),            // output wire s_axis_a_tready
      .s_axis_a_tdata(alpha_lnt),              // input wire [31 : 0] s_axis_a_tdata
      .m_axis_result_tvalid(sqrt_lnt_valid),  // output wire m_axis_result_tvalid
      .m_axis_result_tready(sqrt_lnt_ready),  // input wire m_axis_result_tready
      .m_axis_result_tdata(sqrt_lnt)    // output wire [31 : 0] m_axis_result_tdata
    );

    floating_point_add qf_ip (
      .aclk(clk),                                  // input wire aclk
      .aresetn(s_aresetn),                            // input wire aresetn
      .s_axis_a_tvalid(xdivt_valid),            // input wire s_axis_a_tvalid
      .s_axis_a_tready(xdivt_ready),            // output wire s_axis_a_tready
      .s_axis_a_tdata(xdivt),              // input wire [31 : 0] s_axis_a_tdata
      .s_axis_b_tvalid(sqrt_lnt_valid),            // input wire s_axis_b_tvalid
      .s_axis_b_tready(sqrt_lnt_ready),            // output wire s_axis_b_tready
      .s_axis_b_tdata(sqrt_lnt),              // input wire [31 : 0] s_axis_b_tdata
      .m_axis_result_tvalid(q_valid),  // output wire m_axis_result_tvalid
      .m_axis_result_tready(1'b1),  // input wire m_axis_result_tready
      .m_axis_result_tdata(q)    // output wire [31 : 0] m_axis_result_tdata
    );

endmodule
```

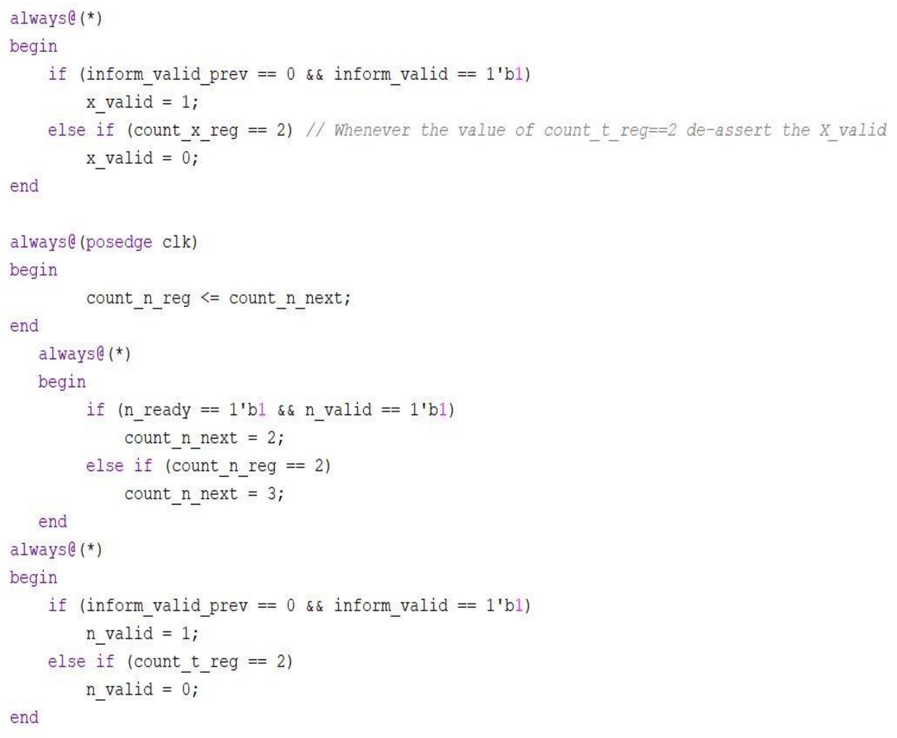
12. Next, we will create the driver/top module of the Q function module we created above which will provide the necessary signals to the Q function module. Add a new design source and write the following code.



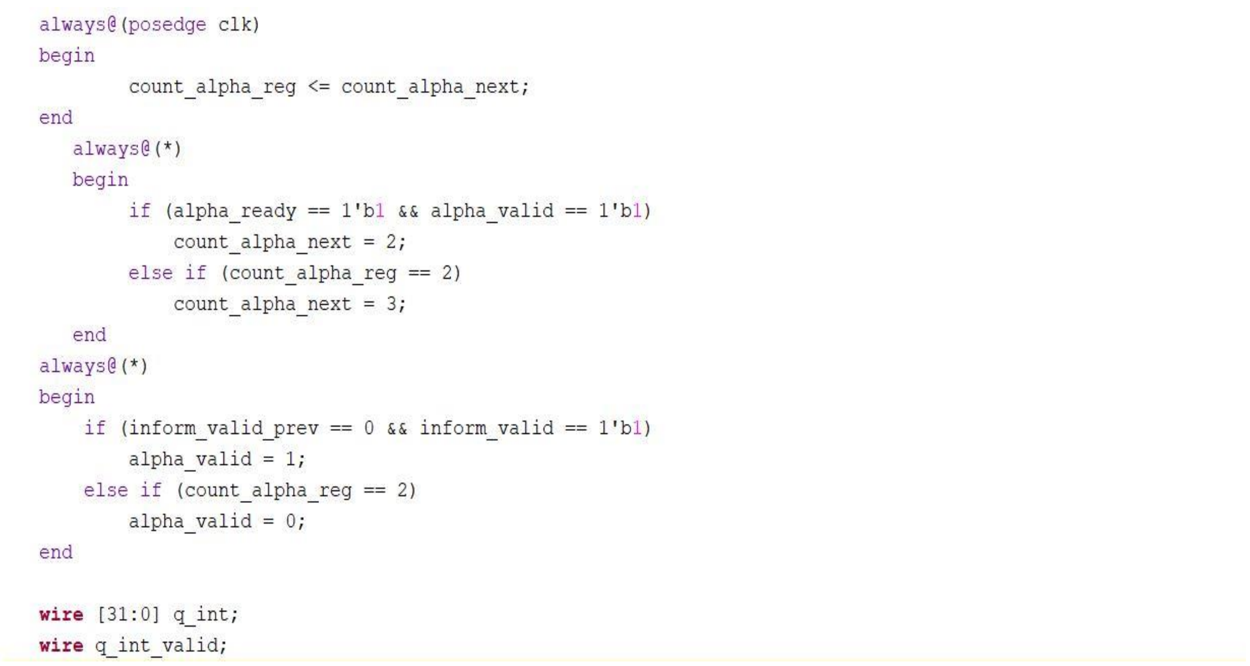
```
`timescale 1ns / 1ps

module q_function_top(
    input clk,
    input s_aresetn,
    input [31:0] x,
    input [31:0] t,
    input [31:0] n,
    input [31:0] alpha,
    input inform_valid, //This will go high whenever new set of values come from the SDK
    output reg [31:0] q = 0
    );

    reg inform_valid_prev = 0;

    always@(posedge clk) // To check the transition of inform_valid from 0 to 1 we are storing the value
      inform_valid_prev <= inform_valid;

    wire t_ready, x_ready, n_ready, alpha_ready;
    reg t_valid = 0, x_valid = 0, n_valid = 0, alpha_valid = 0;

    reg [1:0] count_t_next = 0, count_t_reg = 0;
    reg [1:0] count_x_next = 0, count_x_reg = 0;
    reg [1:0] count_n_next = 0, count_n_reg = 0;
    reg [1:0] count_alpha_next = 0, count_alpha_reg = 0;
```



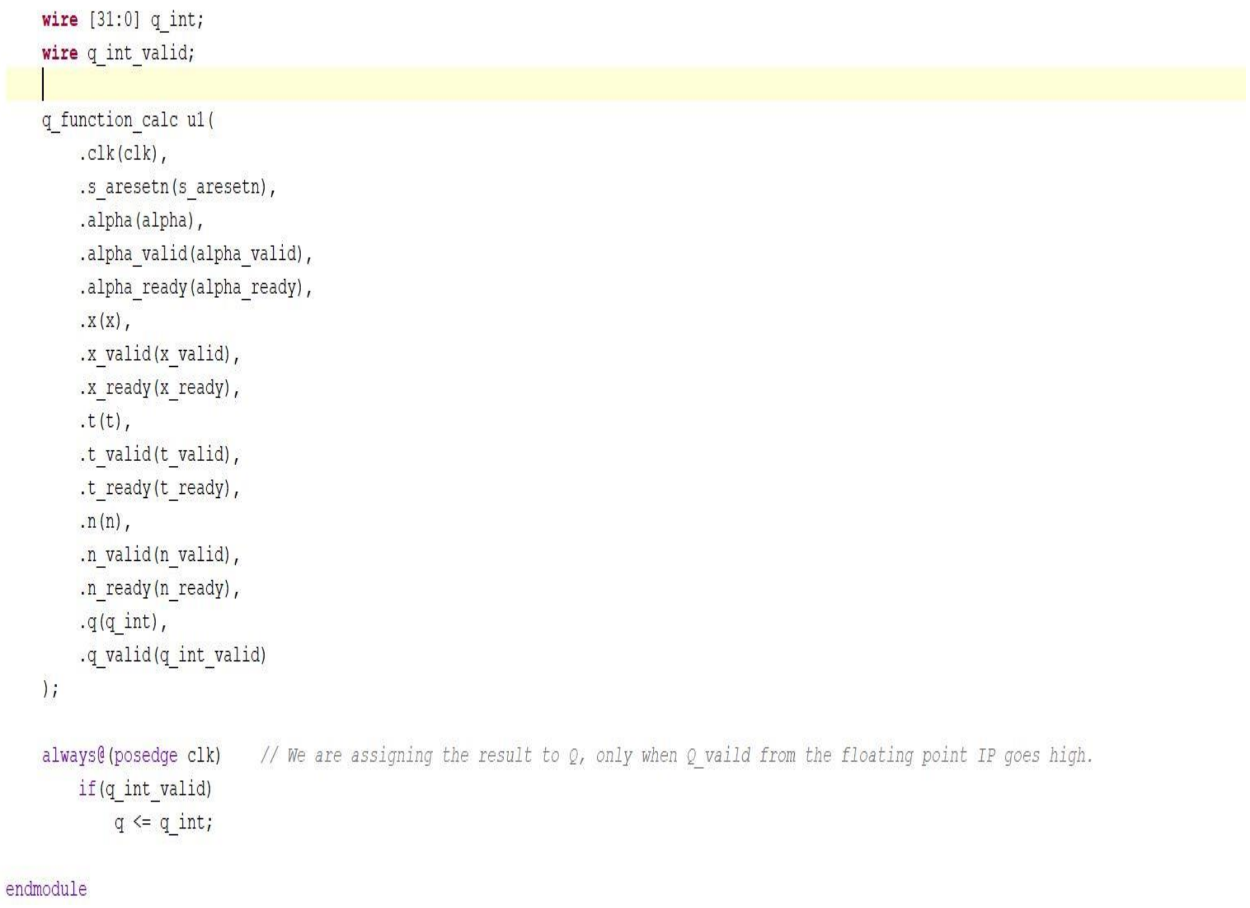
```
always@(posedge clk)
begin
        count_t_reg <= count_t_next;
end
   always@(*)
   begin
        if (t_ready == 1'b1 && t_valid == 1'b1)    // Handshake happened for T and hence the T_valid should go low
            count_t_next = 2;
        else if (count_t_reg == 2)
            count_t_next = 3;
   end
always@(*)
begin
    if (inform_valid_prev == 0 && inform_valid == 1'b1) //Whenever there is a transition form 0-->1 in inform_valid set the valid signals high for X,T,N,alpha
        t_valid = 1;
    else if (count_t_reg == 2) // Whenever the value of count_t_reg==2 de-assert the T_valid
        t_valid = 0;
end

always@(posedge clk)
begin
        count_x_reg <= count_x_next;
end
   always@(*)
   begin
        if (x_ready == 1'b1 && x_valid == 1'b1)   // Handshake happened for X and hence the T_valid should go low
            count_x_next = 2;
        else if (count_x_reg == 2)
            count_x_next = 3;
   end
```



```
always@(*)
begin
    if (inform_valid_prev == 0 && inform_valid == 1'b1)
        x_valid = 1;
    else if (count_x_reg == 2) // Whenever the value of count_t_reg==2 de-assert the X_valid
        x_valid = 0;
end

always@(posedge clk)
begin
        count_n_reg <= count_n_next;
end
   always@(*)
   begin
        if (n_ready == 1'b1 && n_valid == 1'b1)
            count_n_next = 2;
        else if (count_n_reg == 2)
            count_n_next = 3;
   end
always@(*)
begin
    if (inform_valid_prev == 0 && inform_valid == 1'b1)
        n_valid = 1;
    else if (count_t_reg == 2)
        n_valid = 0;
end
```



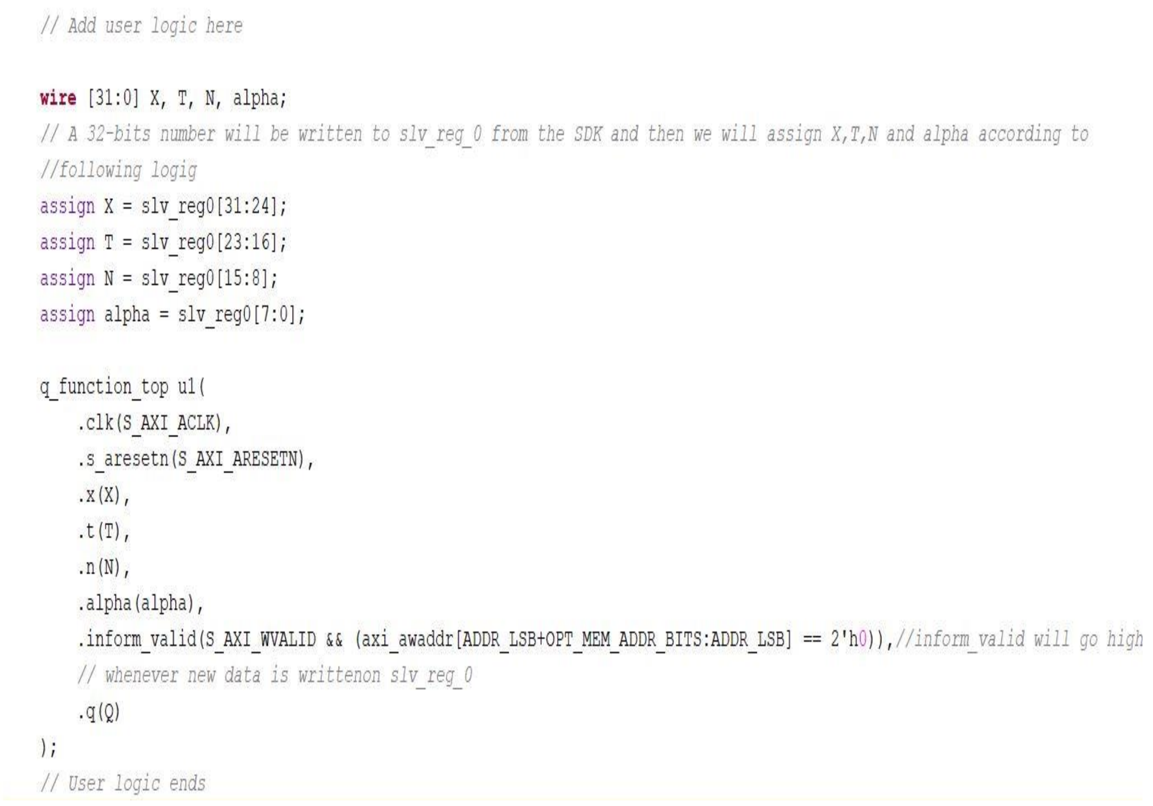
```
always@(posedge clk)
begin
        count_alpha_reg <= count_alpha_next;
end
   always@(*)
   begin
        if (alpha_ready == 1'b1 && alpha_valid == 1'b1)
            count_alpha_next = 2;
        else if (count_alpha_reg == 2)
            count_alpha_next = 3;
   end
always@(*)
begin
    if (inform_valid_prev == 0 && inform_valid == 1'b1)
        alpha_valid = 1;
    else if (count_alpha_reg == 2)
        alpha_valid = 0;
end

wire [31:0] q_int;
wire q_int_valid;
```



```
wire [31:0] q_int;
wire q_int_valid;

q_function_calc u1(
    .clk(clk),
    .s_aresetn(s_aresetn),
    .alpha(alpha),
    .alpha_valid(alpha_valid),
    .alpha_ready(alpha_ready),
    .x(x),
    .x_valid(x_valid),
    .x_ready(x_ready),
    .t(t),
    .t_valid(t_valid),
    .t_ready(t_ready),
    .n(n),
    .n_valid(n_valid),
    .n_ready(n_ready),
    .q(q_int),
    .q_valid(q_int_valid)
);

always@(posedge clk)    // We are assigning the result to Q, only when Q_vaild from the floating point IP goes high.
    if(q_int_valid)
        q <= q_int;

endmodule
```

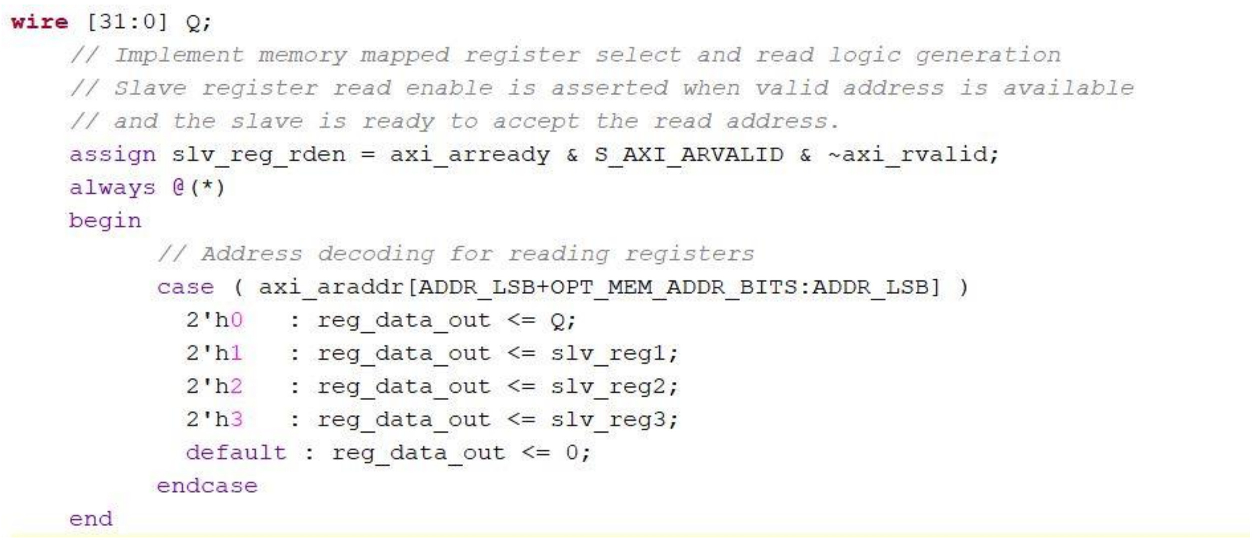
13. Next, we need to make changes in the QFunc_IP_v1_0_S00_AXI file. This will provide the X,N,T,alpha and inform_valid signals to the q_function_top module.No changes needs to done in QFunc_IP_v1_0.

    Change 1:



```
// Add user logic here

wire [31:0] X, T, N, alpha;
// A 32-bits number will be written to slv_reg_0 from the SDK and then we will assign X,T,N and alpha according to
//following logig
assign X = slv_reg0[31:24];
assign T = slv_reg0[23:16];
assign N = slv_reg0[15:8];
assign alpha = slv_reg0[7:0];

q_function_top u1(
    .clk(S_AXI_ACLK),
    .s_aresetn(S_AXI_ARESETN),
    .x(X),
    .t(T),
    .n(N),
    .alpha(alpha),
    .inform_valid(S_AXI_WVALID && (axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] == 2'h0)),//inform_valid will go high
    // whenever new data is writtenon slv_reg_0
    .q(Q)
);
// User logic ends
```

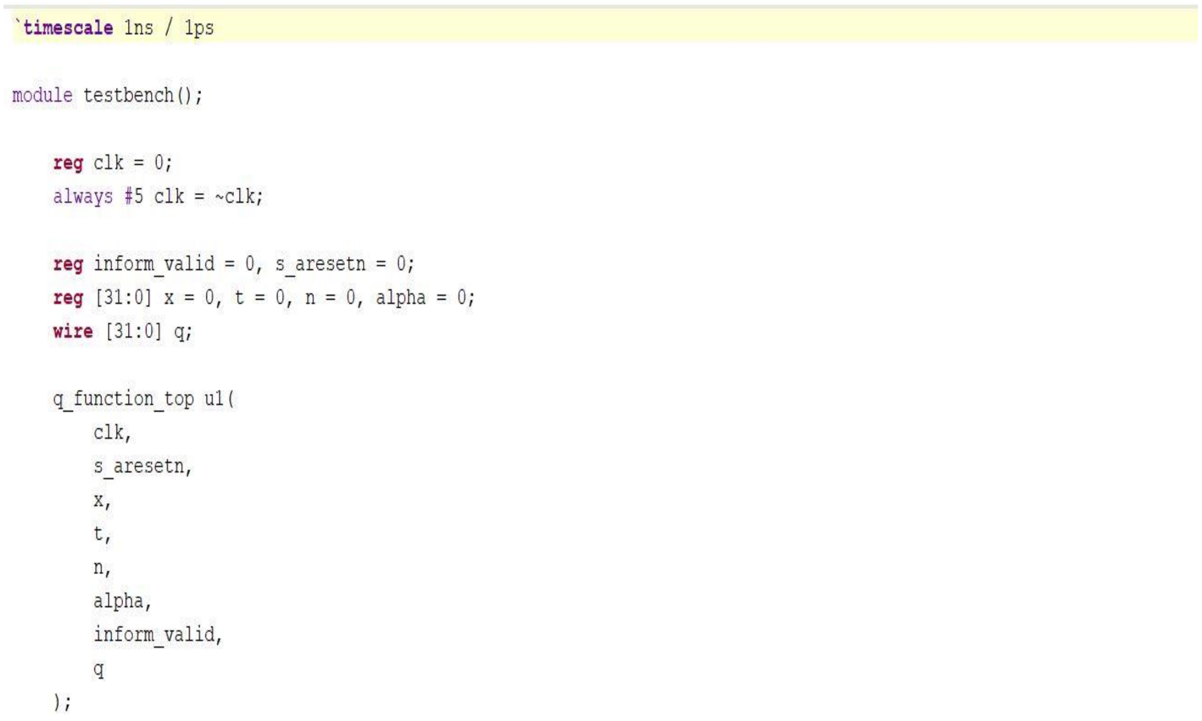
    Change 2:



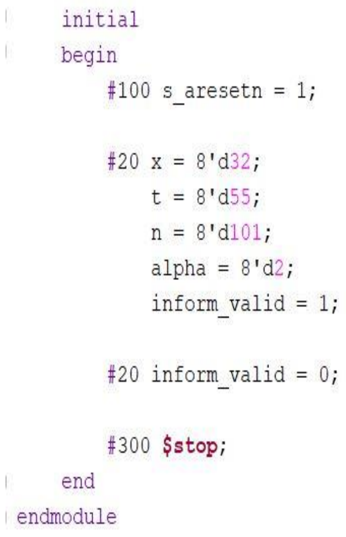
```
wire [31:0] Q;
    // Implement memory mapped register select and read logic generation
    // Slave register read enable is asserted when valid address is available
    // and the slave is ready to accept the read address.
    assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
    always @(*)
    begin
          // Address decoding for reading registers
          case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
            2'h0   : reg_data_out <= Q;
            2'h1   : reg_data_out <= slv_reg1;
            2'h2   : reg_data_out <= slv_reg2;
            2'h3   : reg_data_out <= slv_reg3;
            default : reg_data_out <= 0;
          endcase
    end
```

Part-2

1. Add a simulation source and write the following testbench.

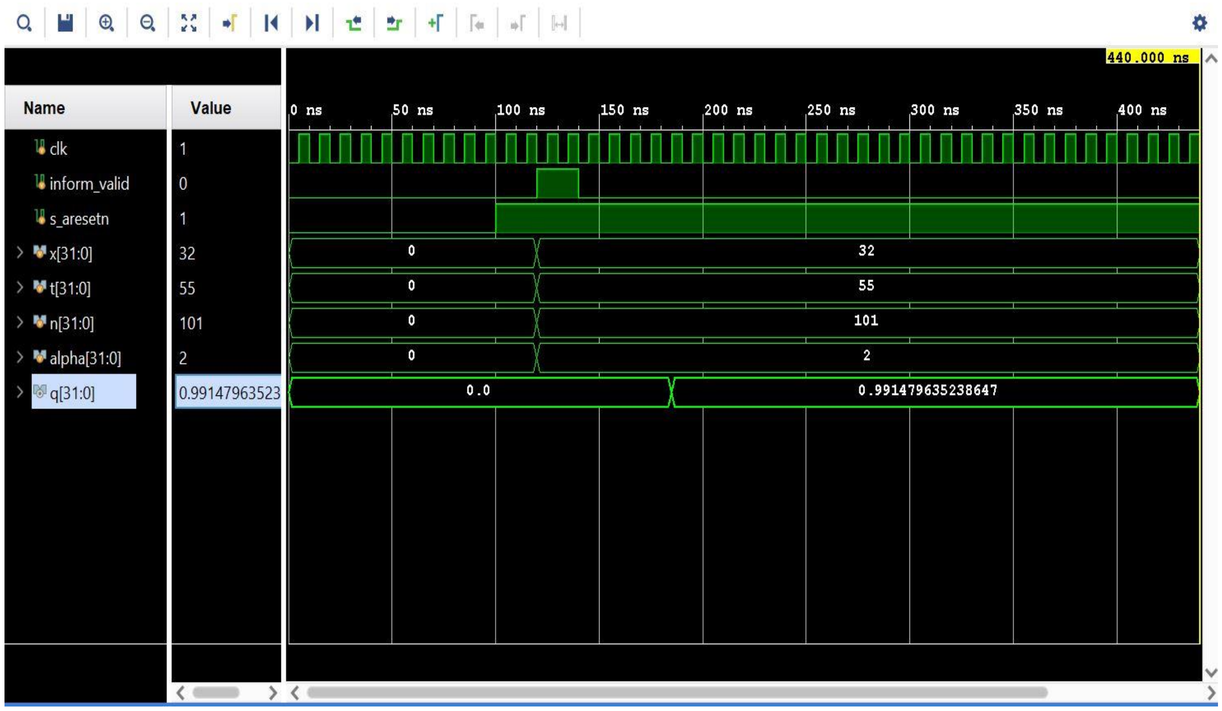


```
`timescale 1ns / 1ps

module testbench();

    reg clk = 0;
    always #5 clk = ~clk;

    reg inform_valid = 0, s_aresetn = 0;
    reg [31:0] x = 0, t = 0, n = 0, alpha = 0;
    wire [31:0] q;

    q_function_top u1(
        clk,
        s_aresetn,
        x,
        t,
        n,
        alpha,
        inform_valid,
        q
    );
```

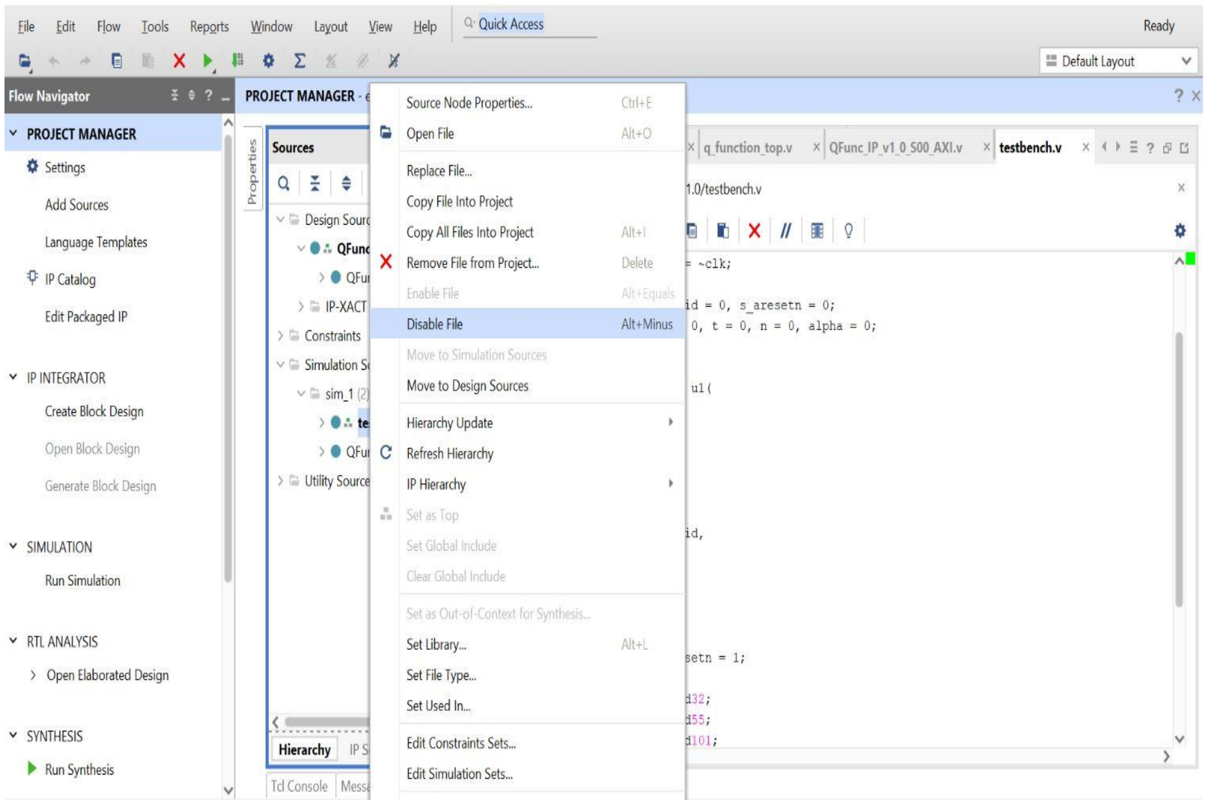


```
    initial
    begin
        #100 s_aresetn = 1;

        #20 x = 8'd32;
            t = 8'd55;
            n = 8'd101;
            alpha = 8'd2;
            inform_valid = 1;

        #20 inform_valid = 0;

        #300 $stop;
    end
endmodule
```

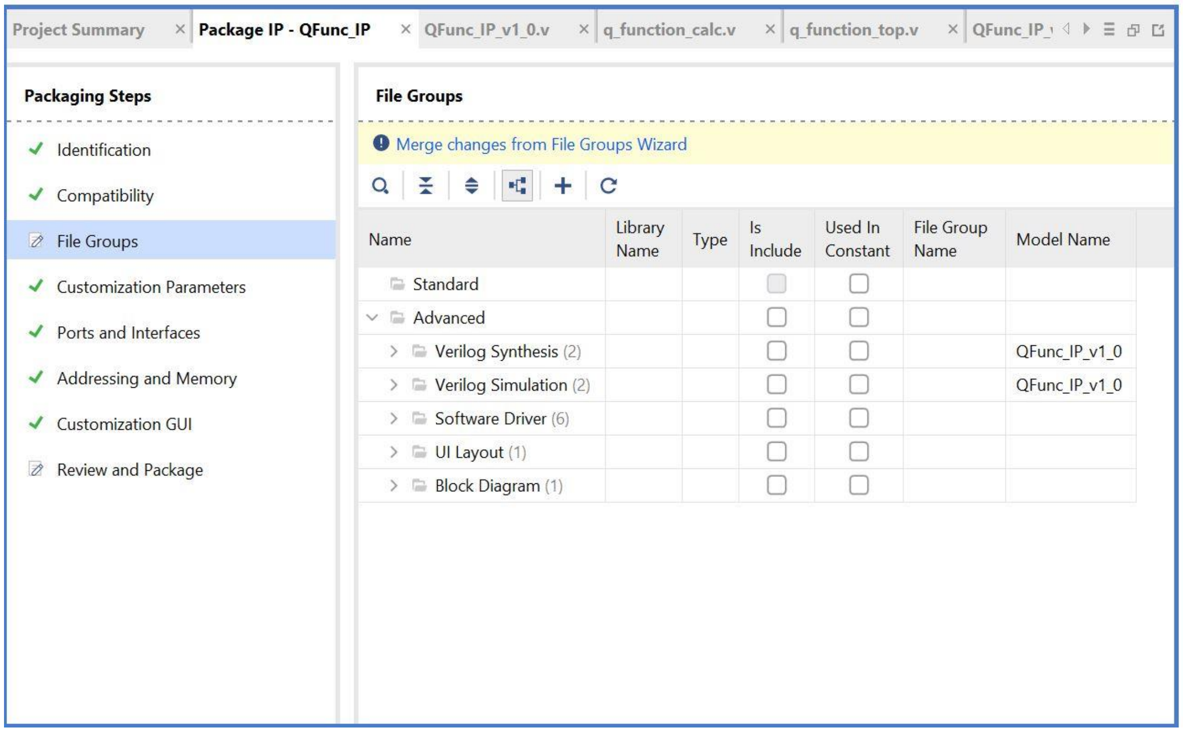
2. Run the simulation and check the correctness.



3. Next, we will package the IP. Before doing that disable the simulation file.

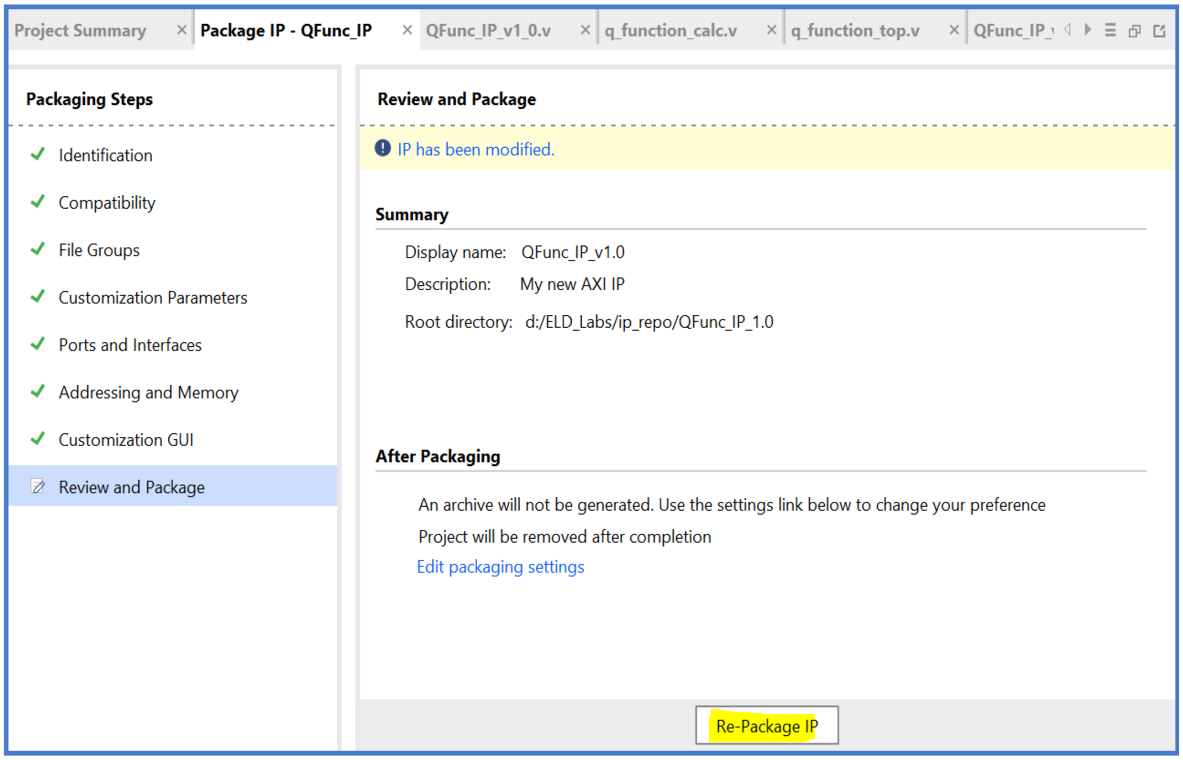


4. Go to package IP window and merge changes in the File Groups Window.



Name	Library Name	Type	Is Include	Used In Constant	File Group Name	Model Name
Standard			☐	☐		
Advanced			☐	☐		
Verilog Synthesis (2)			☐	☐		QFunc_IP_v1_0
Verilog Simulation (2)			☐	☐		QFunc_IP_v1_0
Software Driver (6)			☐	☐		
UI Layout (1)			☐	☐		
Block Diagram (1)			☐	☐		

5. Go to Review and Package window and Re-Package IP. Close the project.



Summary

Display name: QFunc_IP_v1.0
Description: My new AXI IP
Root directory: d:/ELD_Labs/ip_repo/QFunc_IP_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

Homework(Graded):

1. **Paste all the codes and screenshot of simulation results.**

Self-Study:

Write the tesbench for verifying the Q Function IP