Lab 7

Tasks to be done in this Lab:

1. In this Lab, we will implement and verify the following function using the floating-point IP
   - **Q = (X/T) + sqrt (ln(N)/T)**

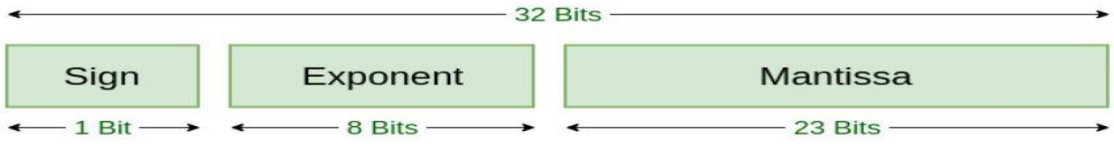
   Here X, T, N are integers such that X <=T and N > T. Write a test bench to verify the corresponding value of Q for the given value of X, T, N.
2. Write a testbench taking care of valid, ready signals to verify the functionality of the design.

Topics to explore: 1) Use of Floating-Point IP, 2) Floating point numbers, 3) Test bench to verify the functionality

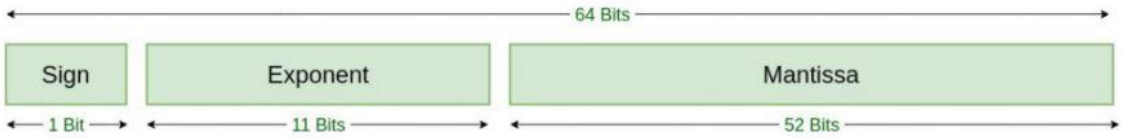
The whole Lab will be divided into three parts.

1. Logarithmic function (ln(N)/T) calculation and verification.
2. Square Root (sqrt (ln(N)/T)) calculation and verification.
3. Final result calculation and verification.

Before starting with the implementation, let's revisit the single-precision and double-precision floating-point representation. IEEE 754 is a technical standard for the representation and computation of floating-point numbers. Two representation is possible according to this standard, namely, single precision and double precision. The format for both is given below:



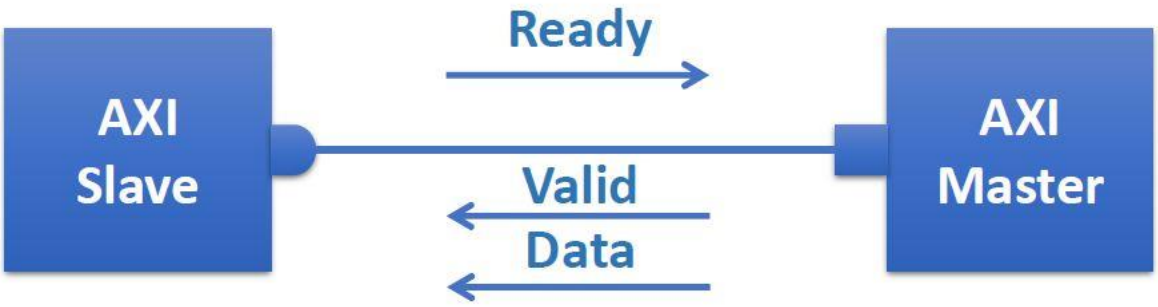
Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

Where Sign bit is for representing the sign of the number (0 for positive 1 for negative), the exponent is the biased exponent, and the mantissa is the normalized mantissa.

The Floating-point IP will be using the AXI Protocols (AXI Stream in particular). Two concepts that will be useful in this Lab are Master-Slave and Valid Ready Signals. In AXI, the transactions take place between Master and Slave, as shown below:



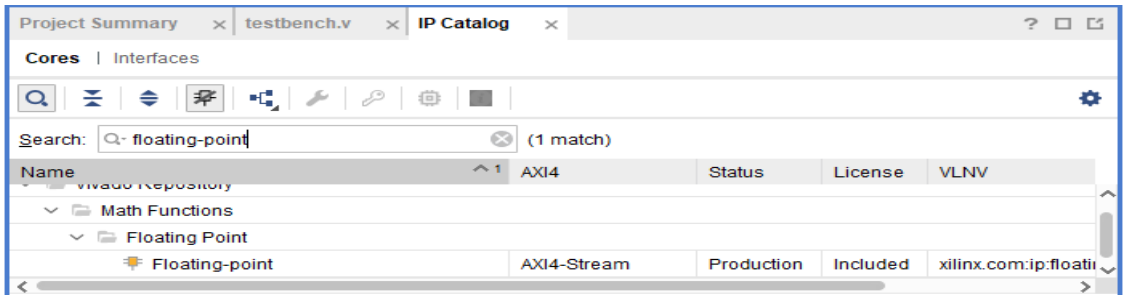
The AXI Master initiates the transactions, and the slave responds to it. Many signals are associated with a transaction out of which the valid and ready signals are useful for this Lab. More on these signals is covered in the Lab tutorial video.

Part -1

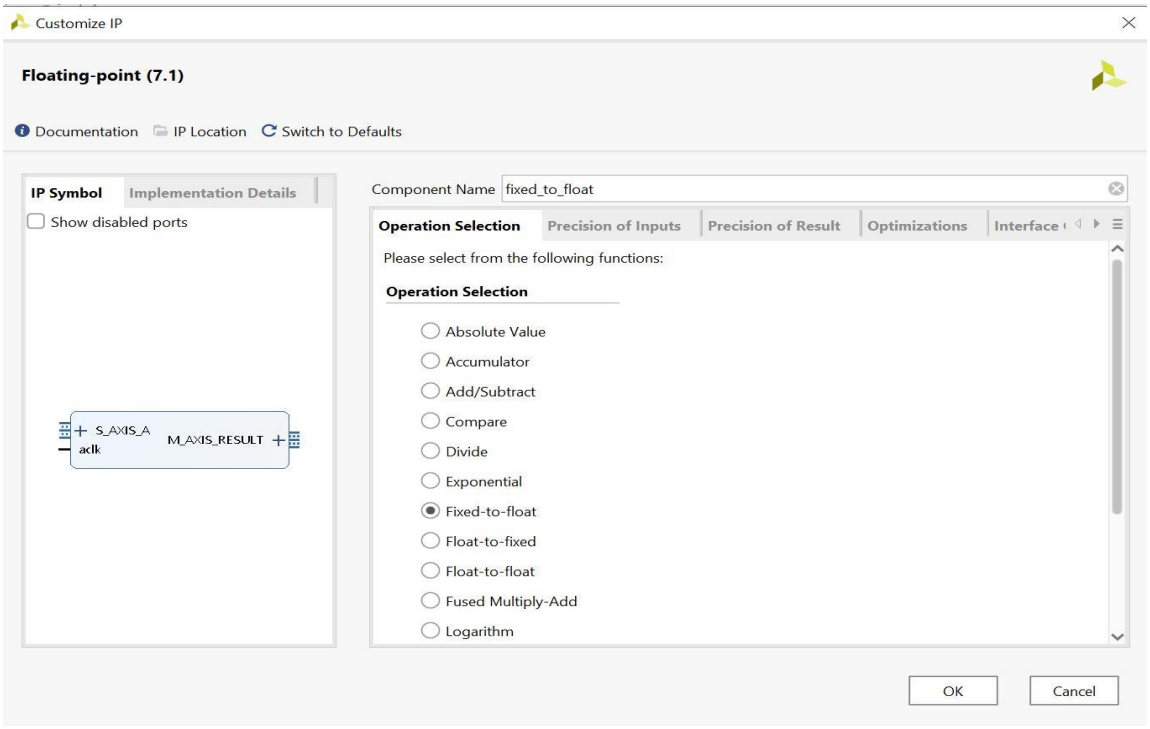
1. Create a top module top_QFunc with the following ports.

```
module top_QFunc(
    input aclk,
    input aresetn,
    input [31:0] N,
    input N_valid,
    output N_ready,
    input [31:0] X,
    input X_valid,
    output X_ready,
    input [31:0] T,
    input T_valid,
    output T_ready,
    input Q_ready,
    output [31:0] Q,
    output Q_valid
    );
```

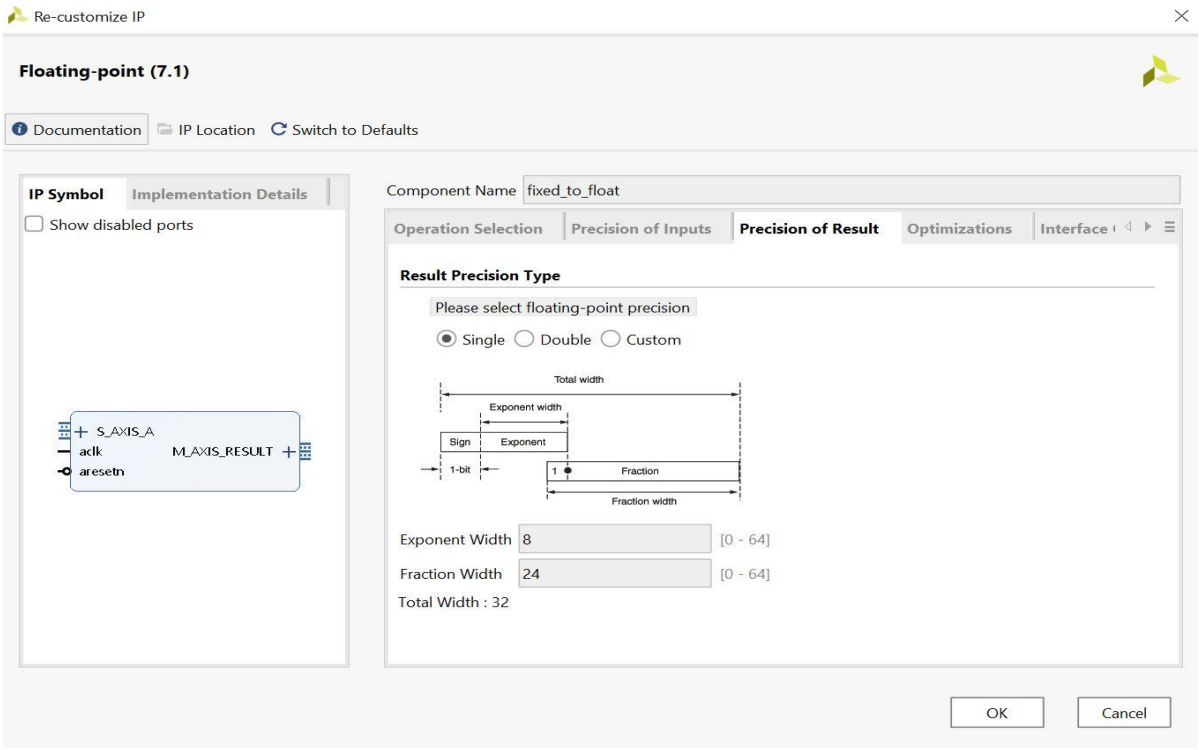
2. The numbers N, T, and X will be provided through the Testbench along with the valid signals. The data format for all will be fixed point, but the floating-point IP works on the floating-point numbers. So the first task is to convert the fixed-point numbers into floating-point numbers. To do that, please follow the steps given below.
    - Add the floating-point IP



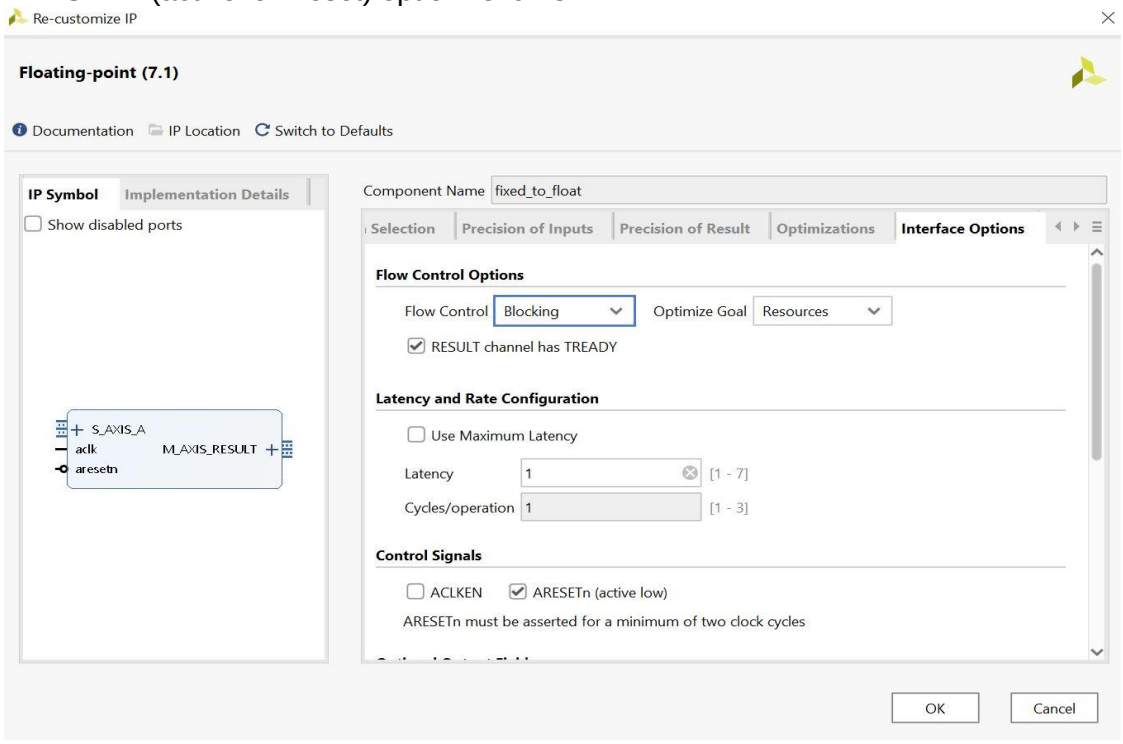
- Select fixed-to-float option.



- Keep the precision of input as default Int32.
- For the precision of results select, single precision.



- In the interface options, select latency as 1 (output will come faster) and tick the ARESETN (active low reset) option. Click OK.



3. Once you have added the fixed_to_float IP to your design, instantiate it three times to convert N, T, and X to floating-point.

```
    wire [31:0] FN;
    wire FN_valid,FN_ready;

    fixed_to_float N_float (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(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(FN_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FN_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FN)    // output wire [31 : 0] m_axis_result_tdata
);

    wire [31:0] FX;
    wire FX_valid,FX_ready;

    fixed_to_float X_float (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(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(FX_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FX_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FX)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
    wire [31:0] FT;
    wire FT_valid,FT_ready;

    fixed_to_float T_float (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(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(FT_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FT_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FT)    // output wire [31 : 0] m_axis_result_tdata
);
```

4. The next step is to calculate ln(N). For this, add the floating-point IP and select the Logarithm operation and instantiate that in your top module.

```
wire [31:0] ln_N;
wire ln_N_ready,ln_N_valid;

ln ln__N(
  .aclk(aclk),                                    // input wire aclk
  .aresetn(aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(FN_valid),                  // input wire s_axis_a_tvalid
  .s_axis_a_tready(FN_ready),                  // output wire s_axis_a_tready
  .s_axis_a_tdata(FN),                      // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(ln_N_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(ln_N_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(ln_N)    // output wire [31 : 0] m_axis_result_tdata
);
```

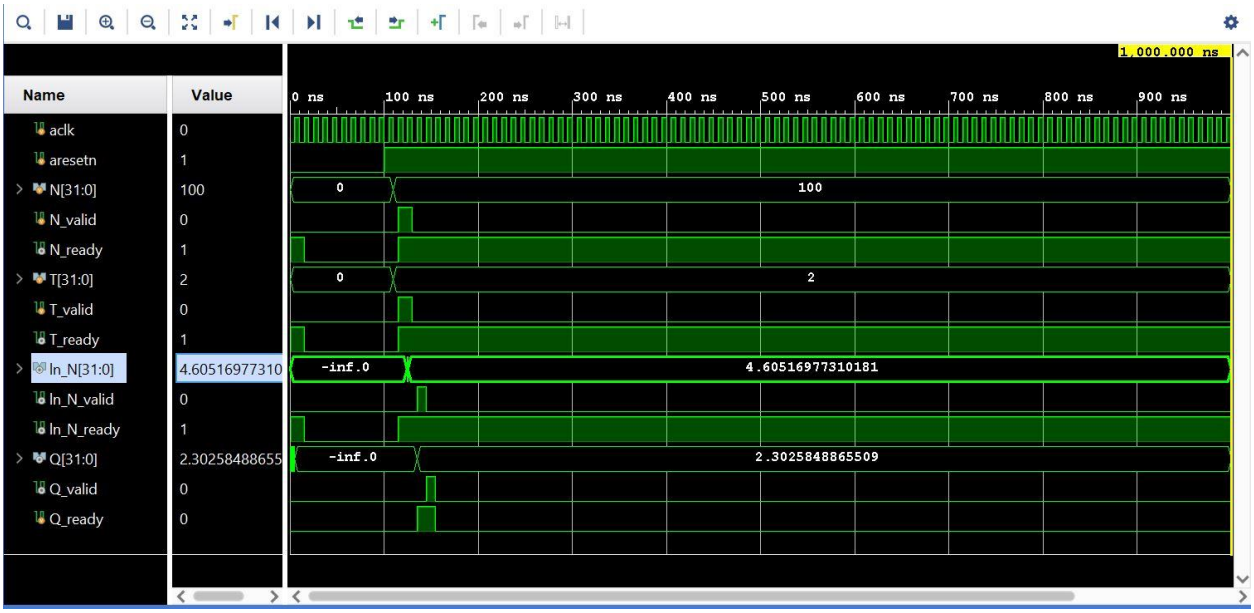
5. The next step is to calculate ln(N)/T. For this, add the floating-point IP and select the Divide operation and instantiate that in your top module.

```
divide lnN_div_T (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(ln_N_valid),                // input wire s_axis_a_tvalid
  .s_axis_a_tready(ln_N_ready),                // output wire s_axis_a_tready
  .s_axis_a_tdata(ln_N),                    // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),                  // input wire s_axis_b_tvalid
  .s_axis_b_tready(FT_ready),                  // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),                      // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(Q_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(Q_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(Q)    // output wire [31 : 0] m_axis_result_tdata
);
```

Note: The output of the IP is connected to the Q interface (Q, Q_valid, Q_ready) of the design as the first part was to calculate the Logarithmic function and verify the results.

6. Write the Testbench as shown below and verify the results of ln(N)/T by running the simulation. Please note that we need to consider the result only when we get a Q_valid signal.

```
module q_tb(

    );

    reg aclk,aresetn,N_valid,X_valid,T_valid,Q_ready;
    reg [31:0] N,X,T;
    wire [31:0] Q;
    wire N_ready,X_ready,T_ready,Q_valid;

    top_QFunc tb1(.aclk(aclk),.aresetn(aresetn),.N(N),.X(X),.T(T),.Q(Q),
                .N_valid(N_valid),.X_valid(X_valid),.T_valid(T_valid),.Q_valid(Q_valid),
                .N_ready(N_ready),.X_ready(X_ready),.T_ready(T_ready),.Q_ready(Q_ready));

     initial
        begin
            aclk=0;
            aresetn=0;
            N=0;
            N_valid=0;
            X=0;
            X_valid=0;
            T=0;
            T_valid=0;
            Q_ready=0;
        end

     always #5 aclk=~aclk;


    initial
        begin
        #100 aresetn=1;
        #10 N=100;
        #5 N_valid=1;
            while(N_ready==0)     // Wait for the ready signal from the IP
            #5 N_valid=1;
         #10 N_valid=0;          // Set the valid signal '0' once the handshake is complete
    end

   initial
        begin
        #110 T=2;
        #5 T_valid=1;
            while(T_ready==0)  // Wait for the ready signal from the IP
            #5 T_valid=1;
         #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete
    end

    initial
        begin
        #110 X=2;
        #5 X_valid=1;
            while(X_ready==0)  // Wait for the ready signal from the IP
            #5 X_valid=1;
         #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
         #5 Q_ready=1'b1;

         wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
            #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
         #5 Q_ready=1'b0;
    end
endmodule
```

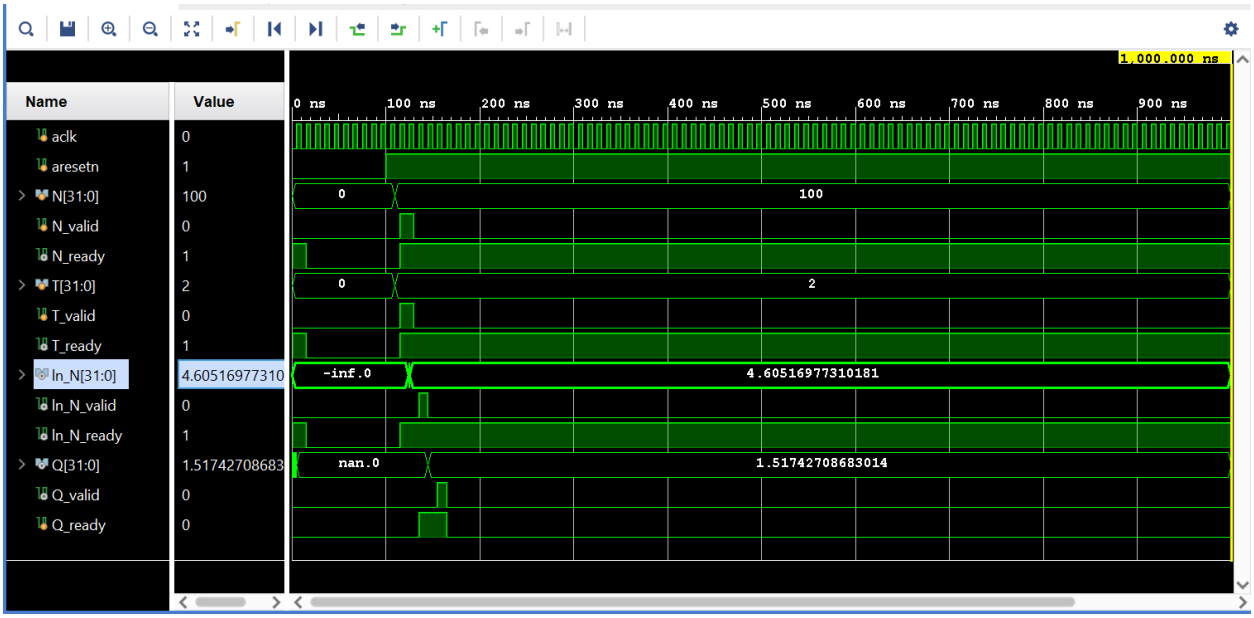


Part-2

1. In this part, we need to calculate sqrt(ln(N)/T) and calculate its functionality.
2. For this, add the floating-point IP with the square root option and instantiate that in your top module. But before that, we need to make changes in the divide IP we instantiated earlier, as shown below:

```
wire [31:0] ln_N_div_T;
wire ln_N_div_T_ready,ln_N_div_T_valid;

divide lnN_div_T (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(ln_N_valid),                 // input wire s_axis_a_tvalid
  .s_axis_a_tready(ln_N_ready),                 // output wire s_axis_a_tready
  .s_axis_a_tdata(ln_N),                      // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),                   // input wire s_axis_b_tvalid
  .s_axis_b_tready(FT_ready),                   // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),                      // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(ln_N_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(ln_N_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(ln_N_div_T)    // output wire [31 : 0] m_axis_result_tdata
);

//wire [31:0] root;
//wire root_ready,root_valid;

sqrt sqare_root (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(ln_N_div_T_valid),             // input wire s_axis_a_tvalid
  .s_axis_a_tready(ln_N_div_T_ready),             // output wire s_axis_a_tready
  .s_axis_a_tdata(ln_N_div_T),                // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(Q_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(Q_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(Q)    // output wire [31 : 0] m_axis_result_tdata
);
```

3. Simulate the code using the same Testbench and verify the results.



Part-3

Finally, instantiate the divide IP again to find X/T and add the floating-point IP with Addition operation and instantiate it.

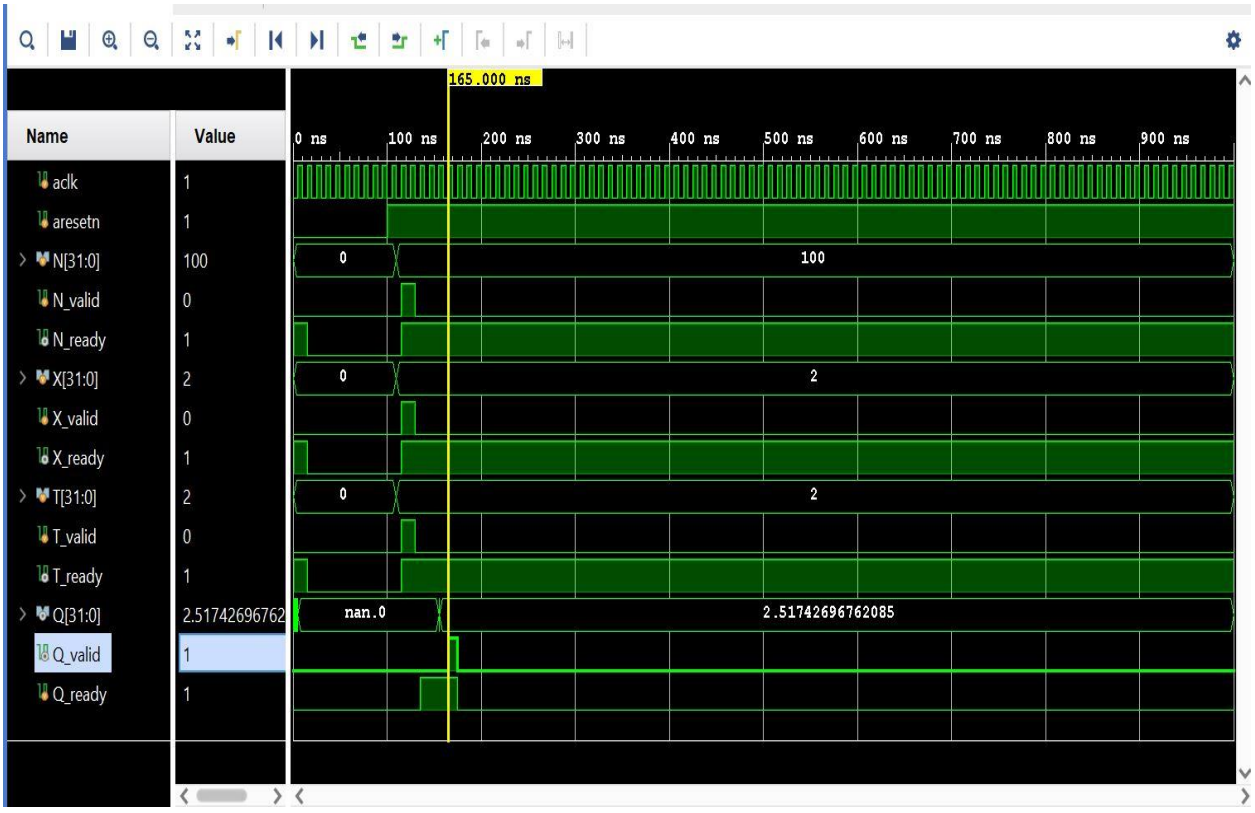
```
wire [31:0] root;
wire root_ready,root_valid;

sqrt sqare_root (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(ln_N_div_T_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(ln_N_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(ln_N_div_T),                 // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(root_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(root_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(root)    // output wire [31 : 0] m_axis_result_tdata
);

wire [31:0] X_div_T;
wire X_div_T_ready,X_div_T_valid;

divide x_div_T (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(FX_valid),              // input wire s_axis_a_tvalid
  .s_axis_a_tready(FX_ready),              // output wire s_axis_a_tready
  .s_axis_a_tdata(FX),                // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),              // input wire s_axis_b_tvalid
  //.s_axis_b_tready(FT_ready),              // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),                // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(X_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(X_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(X_div_T)    // output wire [31 : 0] m_axis_result_tdata
);
```

Note: We commented the ready signal for FT as it will be a case of a multidriven net, and moreover, it is an output from the IP to indicate that it's ready for the operation. So commenting it won't create an issue in the functionality.

```
add result (
  .aclk(aclk),                                    // input wire aclk
  .aresetn(aresetn),                              // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),               // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_div_T_ready),               // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),                     // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(root_valid),                  // input wire s_axis_b_tvalid
  .s_axis_b_tready(root_ready),                  // output wire s_axis_b_tready
  .s_axis_b_tdata(root),                        // input wire [31 : 0] s_axis_b_tdata
  .s_axis_operation_tvalid(1'b1),  // input wire s_axis_operation_tvalid
  //.s_axis_operation_tready(op_ready),  // output wire s_axis_operation_tready
  .s_axis_operation_tdata(8'b00000000),    // input wire [7 : 0] s_axis_operation_tdata
  .m_axis_result_tvalid(Q_valid),        // output wire m_axis_result_tvalid
  .m_axis_result_tready(Q_ready),        // input wire m_axis_result_tready
  .m_axis_result_tdata(Q)          // output wire [31 : 0] m_axis_result_tdata
);
```

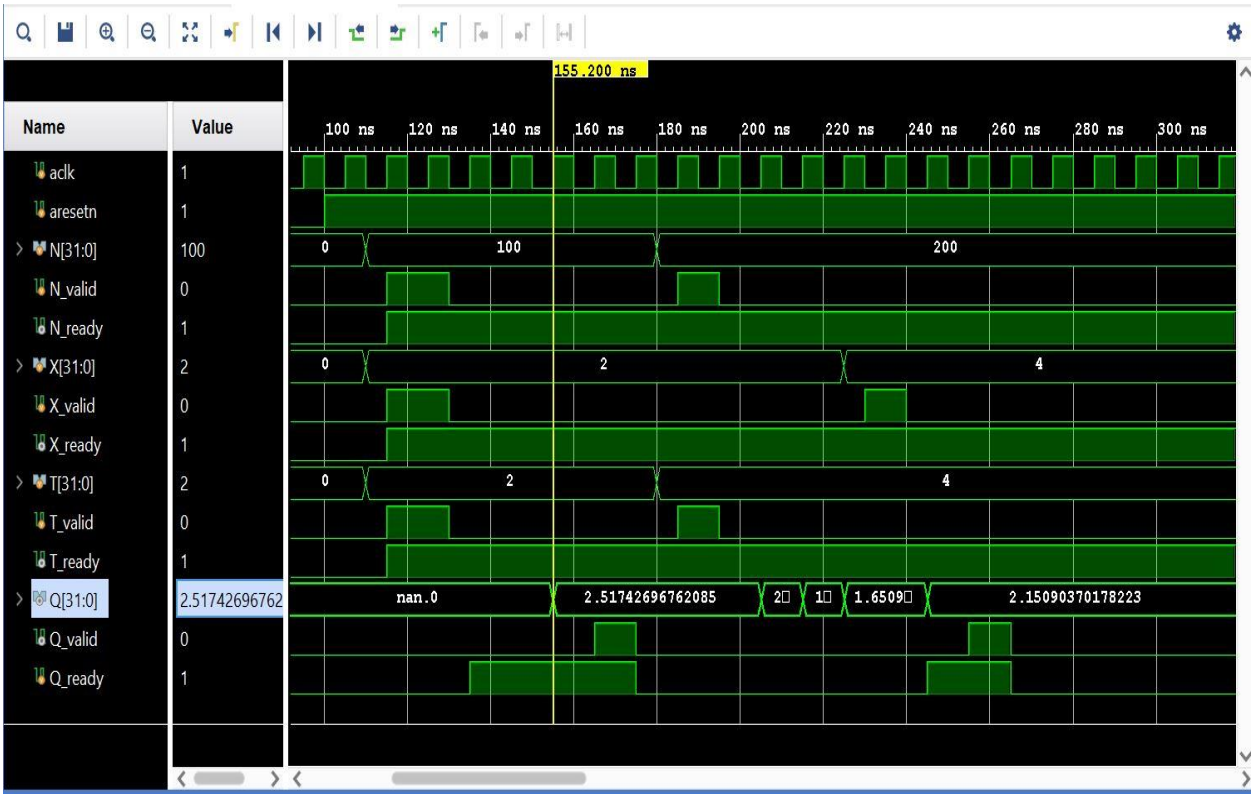
Note: The operation interface is used for defining the operation in the Addition IP. For example, 8'd0 is for the addition operation, and 8'd1 is for subtraction.

Run the simulation and check the final results.



To check the results for different sets of input, make the following changes in the Testbench, and run the simulation.

```
 initial
     begin
     #100 aresetn=1;
     #10 N=100;
     #5 N_valid=1;
         while(N_ready==0)    // Wait for the ready signal from the IP
         #5 N_valid=1;
      #10 N_valid=0;         // Set the valid signal '0' once the handshake is complete

     #50 N=200;
     #5 N_valid=1;
         while(N_ready==0)    // Wait for the ready signal from the IP
         #5 N_valid=1;
      #10 N_valid=0;         // Set the valid signal '0' once the handshake is complete
     end

initial
     begin
     #110 T=2;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete

     #50 T=4;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete
     end

   initial
       begin
       #110 X=2;
       #5 X_valid=1;
           while(X_ready==0)  // Wait for the ready signal from the IP
           #5 X_valid=1;
        #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
        #5 Q_ready=1'b1;

        wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
           #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
        #5 Q_ready=1'b0;

       #50 X=4;
       #5 X_valid=1;
           while(X_ready==0)  // Wait for the ready signal from the IP
           #5 X_valid=1;
        #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
        #5 Q_ready=1'b1;

        wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
           #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
        #5 Q_ready=1'b0;
   end
endmodule
```



Deliverables for Lab 7 Submission(Graded):

1. PDF with code, Testbench, simulation screenshot for Part 1,2, and 3.
2. The simulation must be done for more than one input combination in all the parts.