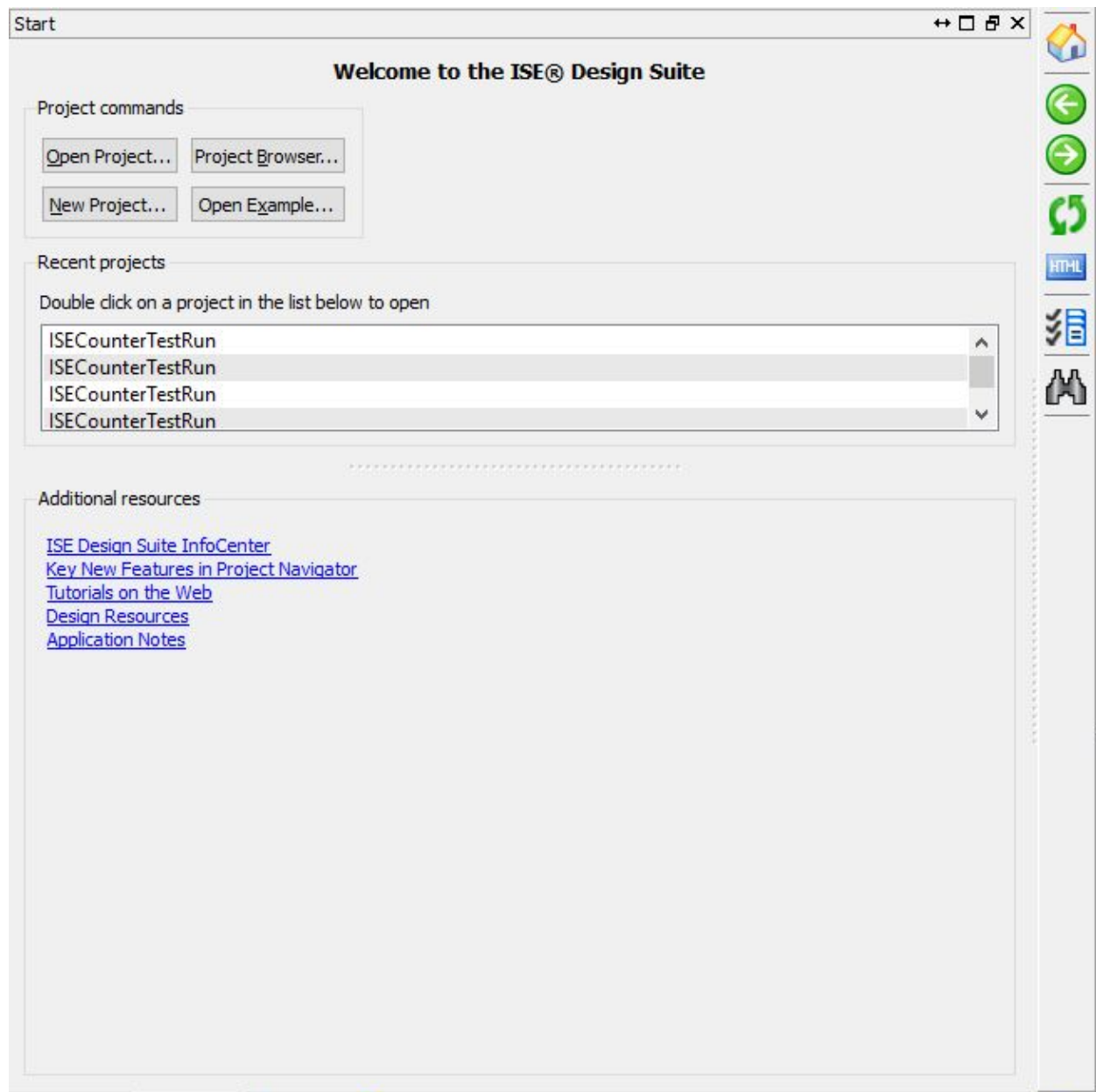
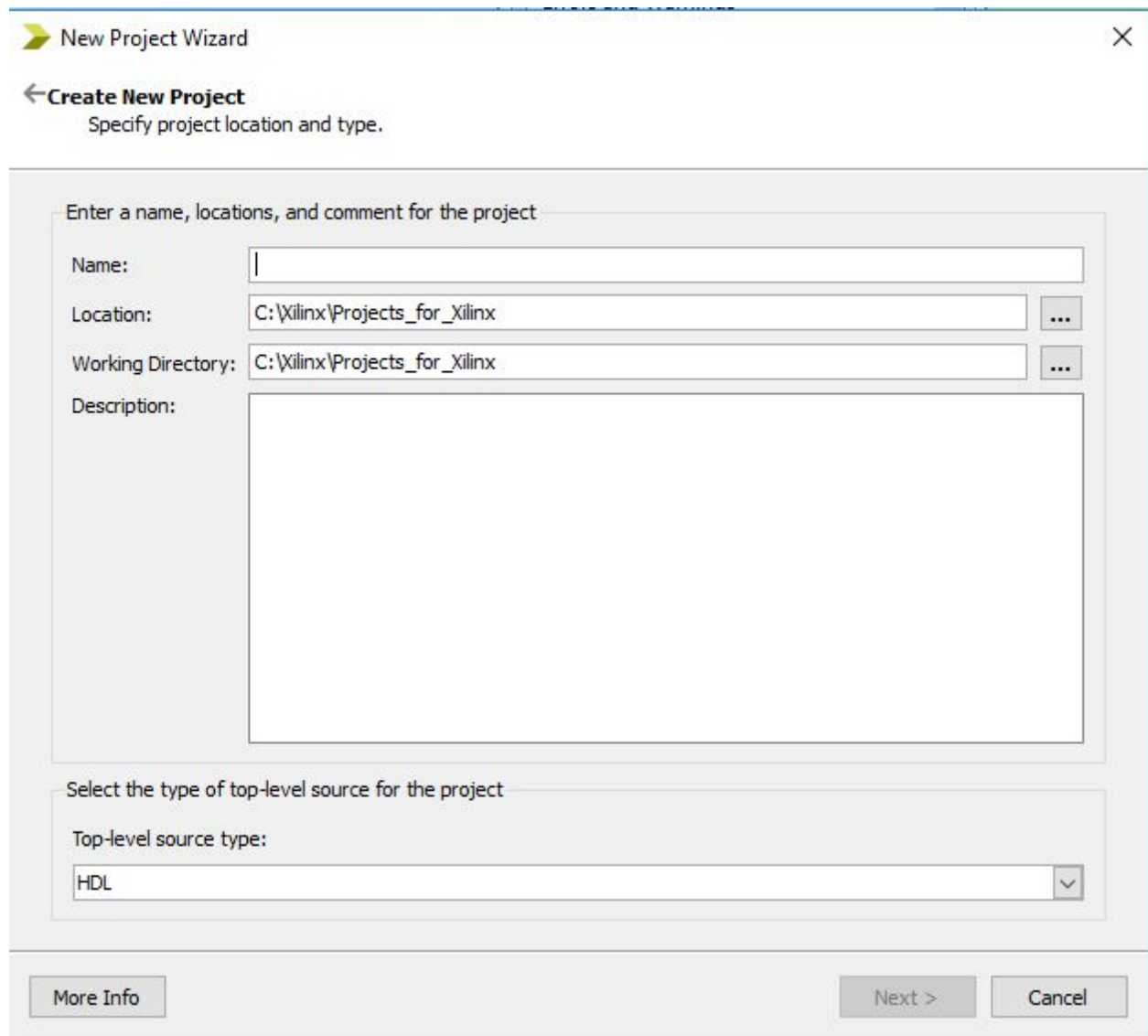


Walkthrough for frequency counter w/ clock dividers

1) Create a new project using the side panel (Start in lower right hand corner)



2) Name it and select top level\_source to be hdl:



The image shows a 'New Project Wizard' dialog box with a title bar containing a green arrow icon, the text 'New Project Wizard', and a close button (X). Below the title bar, there is a back arrow icon followed by the text 'Create New Project' and a subtitle 'Specify project location and type.' The main content area is divided into two sections. The first section is titled 'Enter a name, locations, and comment for the project' and contains four fields: 'Name:' with an empty text box, 'Location:' with a text box containing 'C:\Xilinx\Projects\_for\_Xilinx' and a browse button (...), 'Working Directory:' with a text box containing 'C:\Xilinx\Projects\_for\_Xilinx' and a browse button (...), and 'Description:' with a large empty text area. The second section is titled 'Select the type of top-level source for the project' and contains a 'Top-level source type:' label and a dropdown menu currently showing 'HDL'. At the bottom of the dialog, there are three buttons: 'More Info' on the left, and 'Next >' and 'Cancel' on the right.

New Project Wizard

← Create New Project  
Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location:  ...

Working Directory:  ...

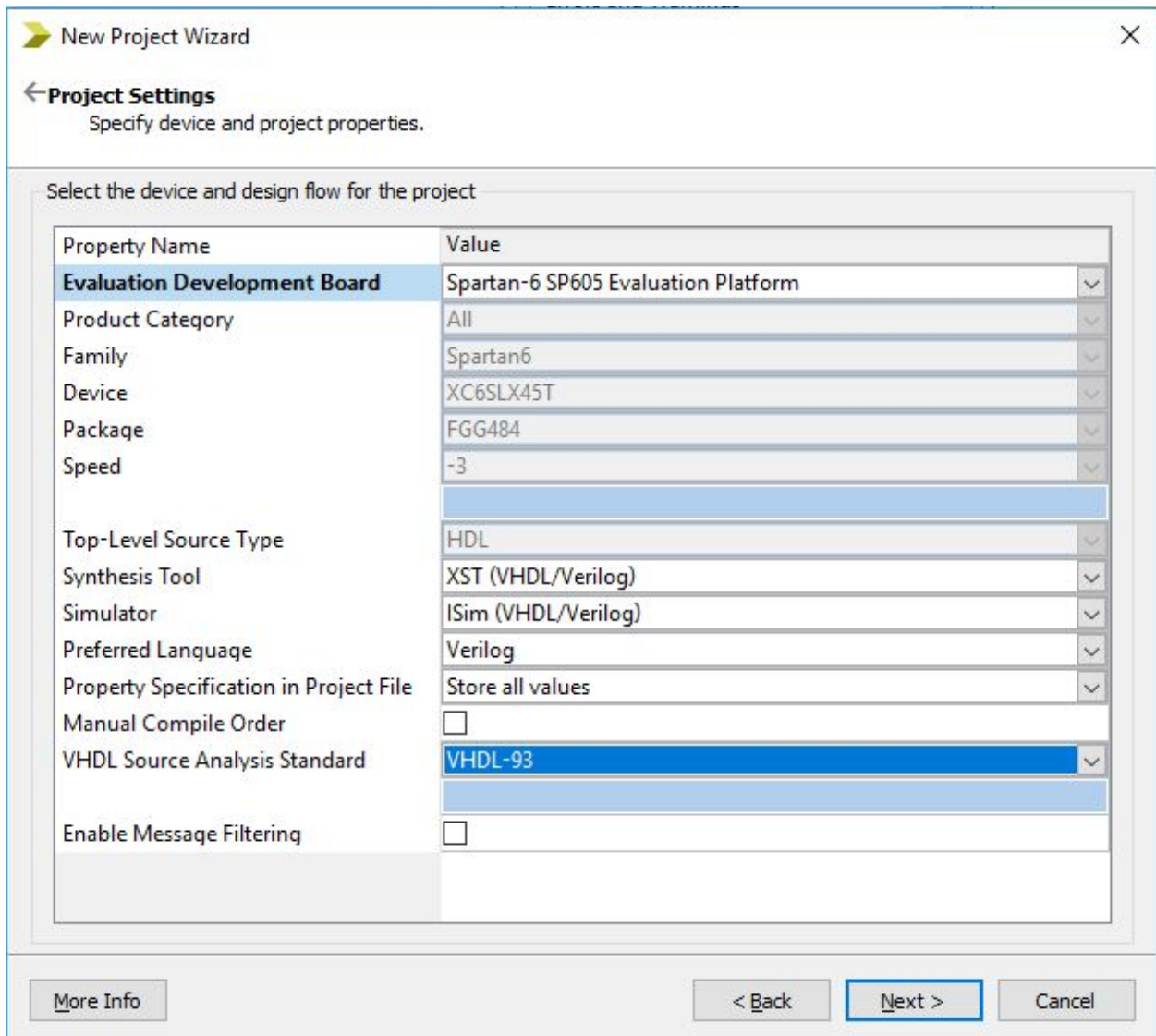
Description:

Select the type of top-level source for the project

Top-level source type:

More Info Next > Cancel

3) In the next screen select the Evaluation Development Board to be the Spartan-6 SP605 Evaluation Platform and change the VHDL Source Analysis to be VHDL-200X and select the preferred language to be Verilog. Then select finish on the next screen.

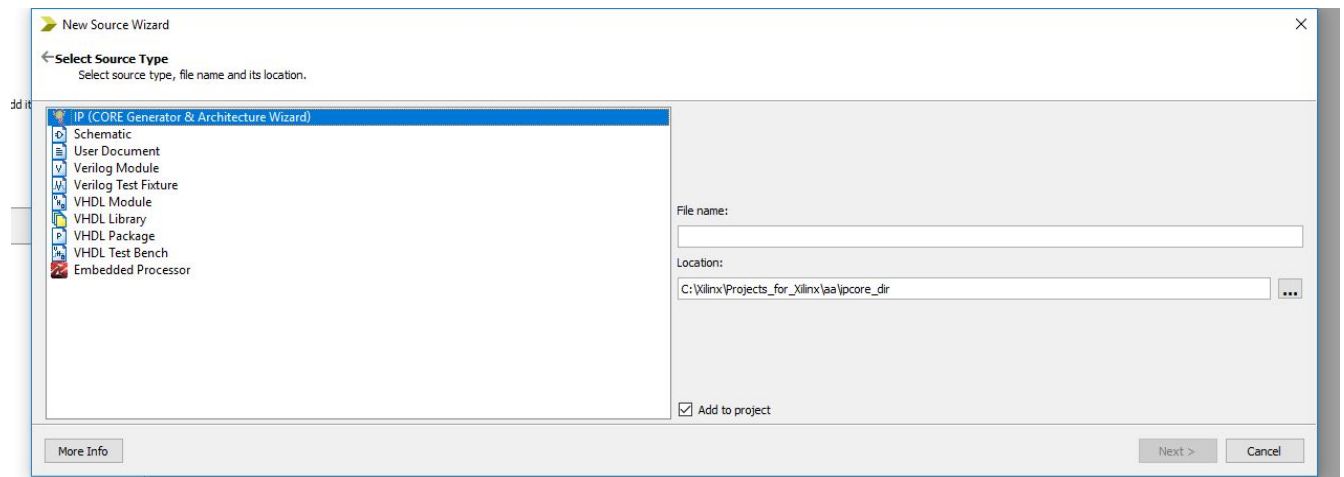


The image shows the 'New Project Wizard' dialog box, specifically the 'Project Settings' step. The title bar says 'New Project Wizard' with a close button. Below the title bar, there's a back arrow and the text 'Project Settings' followed by 'Specify device and project properties.' The main area is titled 'Select the device and design flow for the project'. It contains a table with two columns: 'Property Name' and 'Value'. The properties are: Evaluation Development Board (Spartan-6 SP605 Evaluation Platform), Product Category (All), Family (Spartan6), Device (XC6SLX45T), Package (FGG484), Speed (-3), Top-Level Source Type (HDL), Synthesis Tool (XST (VHDL/Verilog)), Simulator (ISim (VHDL/Verilog)), Preferred Language (Verilog), Property Specification in Project File (Store all values), Manual Compile Order (checkbox), VHDL Source Analysis Standard (VHDL-93), and Enable Message Filtering (checkbox). The 'Next >' button is highlighted with a blue border. There are also 'More Info', '< Back', and 'Cancel' buttons.

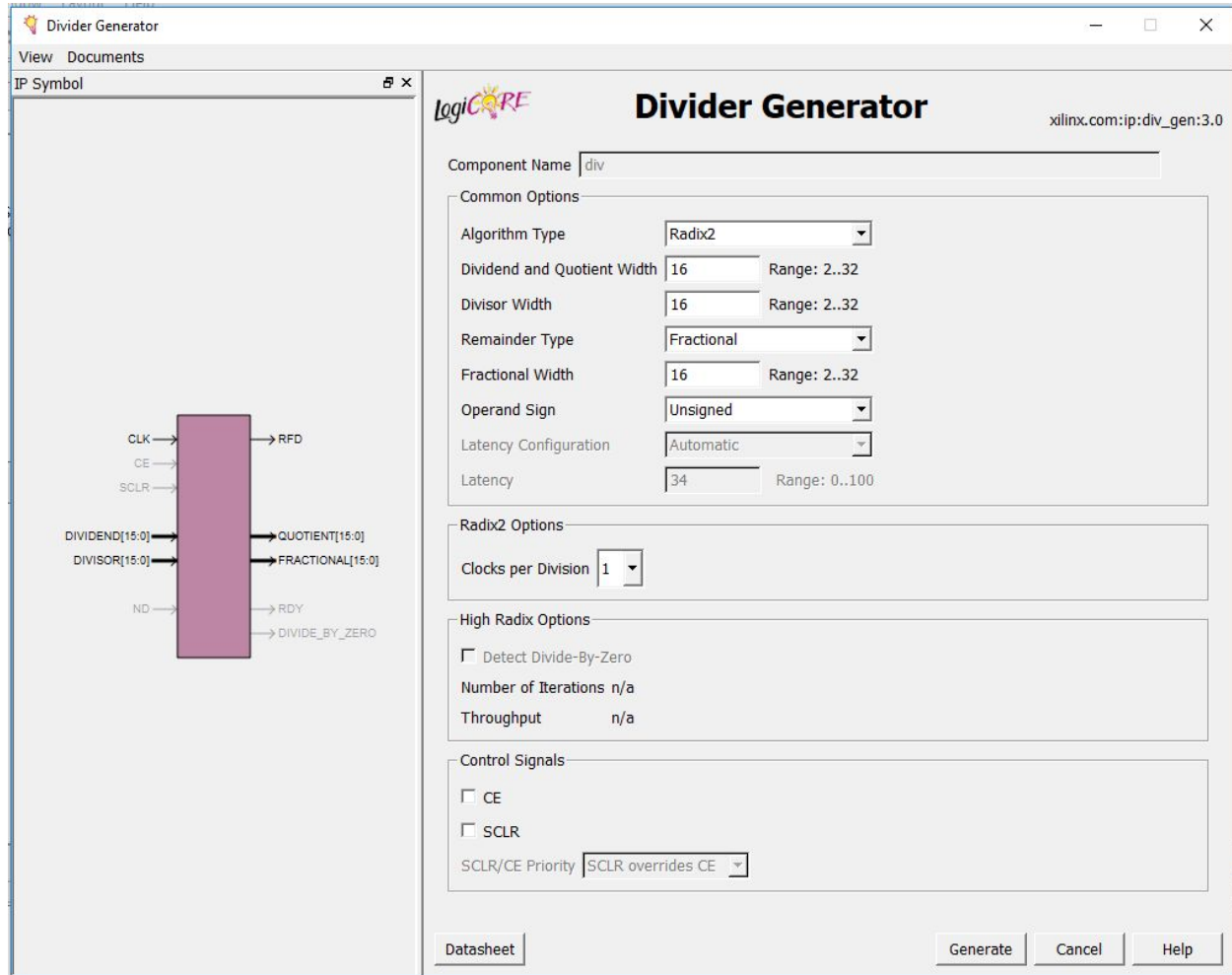
| Property Name                          | Value                               |
|--|-------------------------------------|
| Evaluation Development Board           | Spartan-6 SP605 Evaluation Platform |
| Product Category                       | All                                 |
| Family                                 | Spartan6                            |
| Device                                 | XC6SLX45T                           |
| Package                                | FGG484                              |
| Speed                                  | -3                                  |
| Top-Level Source Type                  | HDL                                 |
| Synthesis Tool                         | XST (VHDL/Verilog)                  |
| Simulator                              | ISim (VHDL/Verilog)                 |
| Preferred Language                     | Verilog                             |
| Property Specification in Project File | Store all values                    |
| Manual Compile Order                   | <input type="checkbox"/>            |
| VHDL Source Analysis Standard          | VHDL-93                             |
| Enable Message Filtering               | <input type="checkbox"/>            |

More Info      < Back      **Next >**      Cancel

4) In the design panel, right-click the xc6slx45t-3fpg484 and select add new source, then select new verilog module and give the module a name.



5) Once complete, select new source again and select IP\_Core generator and find the “Divider Generator 3.0” and select it. Next you must define it. Select the Algorithm to be “Radix2,” the dividend and quotient width to be 16, the Divisor width to be 16, the remainder type to be fractional, the fractional width to be 16, the operand to be unsigned, the clocks per division to be 1, and then select generate.



6) Once complete, copy and paste this code into the verilog module (found in the HDL Instantiation Template of the IP core):

```
div divider (
.clk(clk), // input clk
.rfd(rfd), // output rfd
.dividend(sample), // input [15 : 0] dividend
.divisor(count), // input [15 : 0] divisor
.quotient(quotient), // output [15 : 0] quotient
.fractional(fractional)); // output [15 : 0] fractional
```

This code is similar to, but not exactly, a function in other coding languages, the names in the parentheses are user defined ports, while the `####` are the names of ports of the ip core. This code is modified to fit the names of the ports that will be used later in the walkthrough.

7) Next copy and paste this code into the module counter\_program in the verilog module (these are the arguments, or inputs)

```
input clk,  
input sig,  
output rfd,  
output [15:0] quotient,  
output [15:0] fractional,  
output khz_clk,  
output hz_clk
```

These will be the ports used in the program. clk represents the onboard clock, sig represents the input signal, rfd is the rfd output for the divider, quotient is the output for the divider (for the ones and beyond), fractional is the output for the divider (tenths and beyond), khz\_clk is the indicator for the khz divider being engaged, and the hz\_clk is the indicator for the hz divider being engaged.

8) Next copy and paste this code:

```
reg [15:0] count = 16'b0000000000000000;  
reg [15:0] tempc = 16'b0000000000000000;  
reg [15:0] count2 = 16'b0000000000000000;  
reg [15:0] tempc2 = 16'b0000000000000000;  
reg [15:0] sample = 16'b0000000000000000;  
reg [15:0] temple = 16'b0000000000000000;  
reg khz_clk = 1'b0;  
reg hz_clk = 1'b0;  
reg [15:0] count3 = 16'b0000000000000000;  
reg [15:0] tempc3 = 16'b0000000000000000;
```

These are the instantiation of many Verilog reg, for use in the code. count (and temp) keep track of the positive edges of the onboard clk, count2 (and tempc2) will be used later for both clock dividers, but it is first used in the khz clock divider code, sample (and temple) count the positive edges of the input signal, khz\_clk and hz\_clk are used as indicators to show which divider is being used, and count3 (and tempc3) are used for the hz\_clk divider.

9) Next copy this code:

always @ (posedge sig)

begin

if (count != 16'b1111111111111111 )

begin

temple = sample + 1;

sample = temple;

end

else

begin

temple = temple;

sample = sample;

end

if (fractional <= 16'b000000000010000 && count == 16'b1111111111111111 && khz\_clk == 1'b1 && hz\_clk != 1'b1)

begin

sample = 16'b0000000000000000;

temple = 16'b0000000000000000;

end

else

begin

if (fractional <= 16'b000000000010000 && count == 16'b1111111111111111 && khz\_clk != 1'b1 && hz\_clk == 1'b1)

begin

sample = 16'b0000000000000000;

temple = 16'b0000000000000000;

end

else

begin

sample = sample;

temple = temple;

end

end

if (khz\_clk == 1'b1 && hz\_clk == 1'b1)

begin

sample = 16'b0000000000000000;

temple = 16'b0000000000000000;

end

else

begin

sample = sample;

temple = temple;

```
end  
end
```

This code is used to count the positive edges of the input signal based upon the counts of the onboard clock of the board, the counter will stop counting and hold its value after a certain number of counts of the onboard clock.

This code is to reset the value of the sample and temporary variable if/when a clock divider has been engaged

10) Next copy and paste this code

```
always @ (posedge clk)
```

```
begin//count, check if count is complete and engage dividers
```

```
if (count != 16'b1111111111111111 && sample != 16'b1111111111111111 &&  
sample!=16'b0000000000000000 && khz_clk != 1'b1 && hz_clk != 1'b1)
```

```
begin
```

```
    tempc = count + 1;
```

```
    count = tempc;
```

```
end
```

```
else
```

```
begin
```

```
    if (fractional <= 16'b0000000000010000 && sample!=16'b0000000000000000 && count ==  
16'b1111111111111111 && khz_clk != 1'b1 && hz_clk != 1'b1)
```

```
begin
```

```
    tempc = 16'b0000000000000000;
```

```
    count = 16'b0000000000000000;
```

```
    khz_clk = 1'b1;
```

```
end
```

```
else
```

```
begin
```

```
    if (fractional <= 16'b0000000000010000 && sample!=16'b0000000000000000 && count ==  
16'b1111111111111111 && khz_clk == 1'b1 && hz_clk != 1'b1)
```

```
begin
```

```
    tempc = 16'b0000000000000000;
```

```
    count = 16'b0000000000000000;
```

```
    khz_clk = 1'b0;
```

```
    hz_clk = 1'b1;
```

```
end
```

```
else
```

```
begin
```

```
    if (fractional <= 16'b0000000000010000 && sample!=16'b0000000000000000 && count ==  
16'b1111111111111111 && khz_clk != 1'b1 && hz_clk == 1'b1)
```

```
begin
```



```

    tempc = 16'b0000000000000000;
    count = 16'b0000000000000000;
    khz_clk = 1'b1;
    hz_clk = 1'b1;
end
else
begin
    tempc = tempc;
    count = count;
end
end
end
end
//engage hz clk
if (khz_clk != 1'b1 && hz_clk == 1'b1)
begin
    if (count2 != 16'b0000001111101000)
    begin
        tempc2 = count2 + 1;
        count2 = tempc2;
    end
    else
    begin
        tempc2 = tempc2;
        count2 = count2;
    end
    if (count2 == 16'b0000001111101000 && count != 16'b1111111111111111 )
    begin
        tempc2 = 16'b0000000000000000;
        count2 = 16'b0000000000000000;
        tempc3 = count3 + 1;
        count3 = tempc3;
        if (count3 == 16'b0000001111101000 && count != 16'b1111111111111111 )
        begin
            tempc3 = 16'b0000000000000000;
            count3 = 16'b0000000000000000;
            tempc = count + 1;
            count = tempc;
        end
        else
        begin
            tempc = tempc;
            count = count;
        end
    end
end

```

```

    tempc3 = tempc3;
    count3 = count3;
end
end
else
begin
    tempc2 = tempc2;
    count2 = count2;
    tempc = tempc;
    count = count;
end
end
else
begin
    tempc = tempc;
    count = count;
end
// engage khz clk
if (khz_clk == 1'b1 && hz_clk != 1'b1)
begin
    if (count2 != 16'b00000001111101000)
    begin
        tempc2 = count2 + 1;
        count2 = tempc2;
    end
    else
    begin
        tempc2 = tempc2;
        count2 = count2;
    end
    if (count2 == 16'b00000001111101000 && count != 16'b1111111111111111)
    begin
        tempc2 = 16'b0000000000000000;
        count2 = 16'b0000000000000000;
        tempc = count + 1;
        count = tempc;
    end
    else
    begin
        tempc2 = tempc2;
        count2 = count2;
        tempc = tempc;
        count = count;
    end
end

```

```

    end
end
else
begin
    tempc = tempc;
    count = count;
end
end
endmodule

```

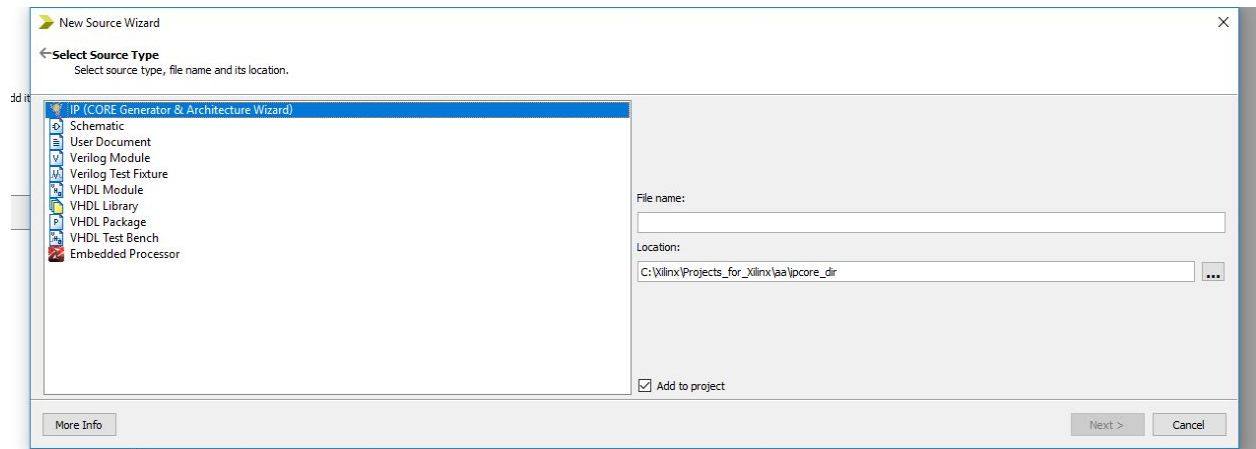
This code is what counts the positive edges of the onboard clock, but only if the clock dividers are not engaged and sample is not equal to the highest value of count (1111111111111111).

This code is what determines when the clock dividers are engaged, it does this by checking whether or not the fractional answer is less than the 16'b00000000000010000, sample is not 0, count reached its highest attainable value, and if the dividers are engaged. All three if statements have many of the same components, however, the main difference between each is the condition the value of clock dividers are 0 or 1. If the other conditions are met and the dividers are off, it engages the khz\_clk. If the other conditions are met and the khz\_clk is on and the hz\_clk is off, it turns off the khz\_clk and turns on hz\_clk. Finally, if the other conditions are met, then it turns on both khz\_clk and hz\_clk. If all of these conditions are not met, then the value of count and its temporary variables are kept.

This code is the clock divider that makes the code count at a frequency of 27 hertz. First it checks whether or not hz\_clk is on and whether or not khz\_clk is off. Then it engages count2 to count to 1000, once there, it adds one to count3 until it reaches 1000, after count3 reaches 1000, it adds one to count. After count reaches (1111111111111111) it stops counting and holds count's value.

This code is the clock divider that makes the code count at a frequency of 27 khz. First it checks if khz\_clk is on and hz\_clk is off, then if this condition is met, count2 will begin counting until it reaches a value of 1000, then it adds one to count until it reaches its maximum value (1111111111111111).

11) Now, choose new source, select implementation constraints file, and name it:



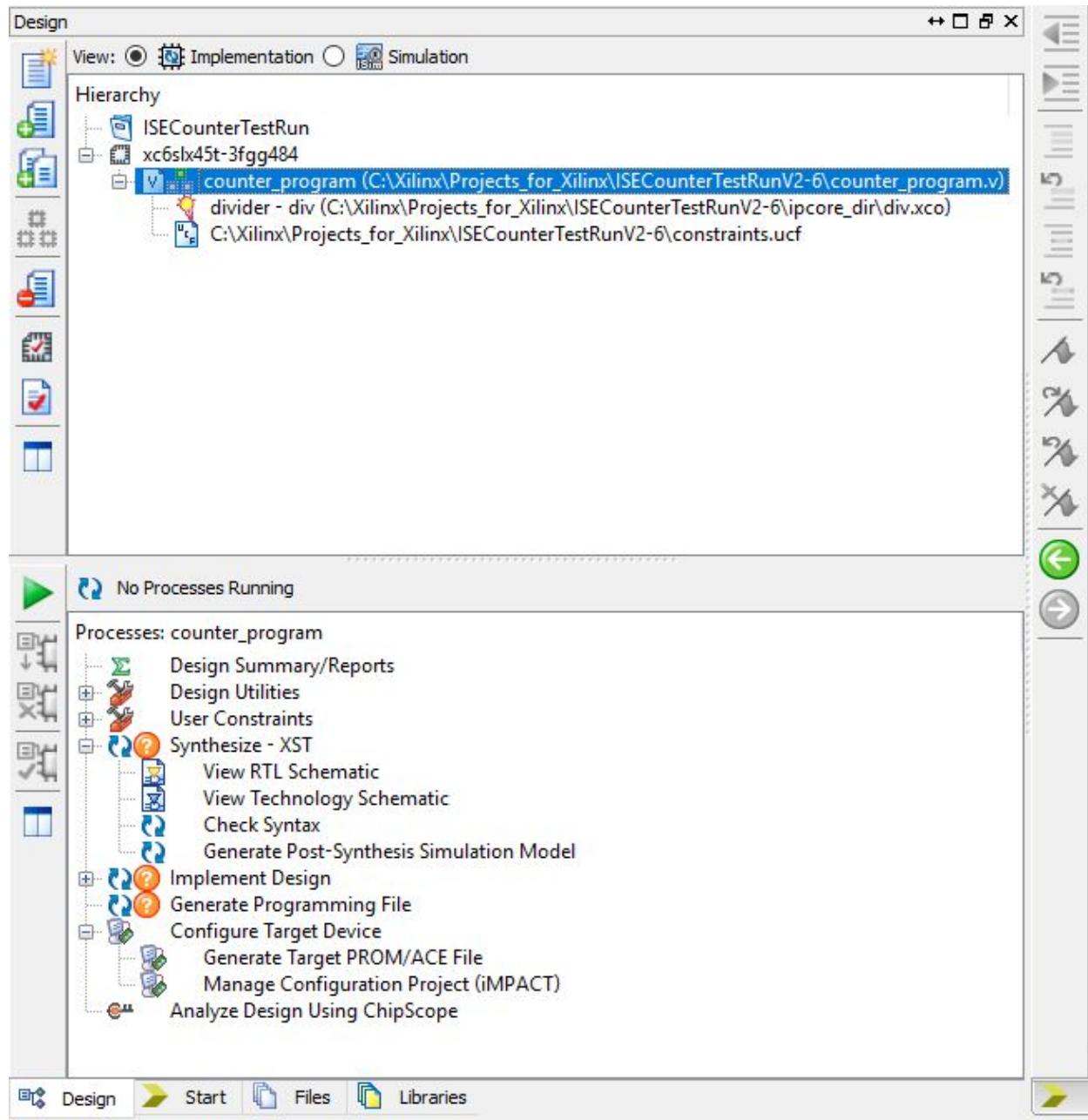
Then paste this code into it:

```
#NET "quotient[0]" LOC = "D17";
#NET "quotient[1]" LOC = "AB4";
#NET "quotient[2]" LOC = "D21";
#NET "quotient[3]" LOC = "W15";
#NET "quotient[4]" LOC = "G7";
#NET "quotient[5]" LOC = "H6";
#NET "quotient[6]" LOC = "D1";
#NET "quotient[7]" LOC = "R7";
//NET "quotient[0]" LOC = "G7";
//NET "quotient[1]" LOC = "H6";
//NET "quotient[2]" LOC = "D1";
//NET "quotient[3]" LOC = "R7";
NET "khz_clk" LOC = "V19";
NET "hz_clk" LOC = "B3";
NET "clk" LOC = "AB13";
NET "sig" LOC = "A3";
NET "sig" CLOCK_DEDICATED_ROUTE = FALSE;
//NET "quotient[0]" LOC = "D17";
//NET "quotient[1]" LOC = "AB4";
//NET "quotient[2]" LOC = "D21";
//NET "quotient[3]" LOC = "W15";
//NET "fractional[0]" LOC = "G7";
//NET "fractional[1]" LOC = "H6";
//NET "fractional[2]" LOC = "D1";
//NET "fractional[3]" LOC = "R7";
NET "fractional[0]" LOC = "D17";
```

```
NET "fractional[1]" LOC = "AB4";  
NET "fractional[2]" LOC = "D21";  
NET "fractional[3]" LOC = "W15";  
NET "fractional[4]" LOC = "G7";  
NET "fractional[5]" LOC = "H6";  
NET "fractional[6]" LOC = "D1";  
NET "fractional[7]" LOC = "R7";
```

This code implements the ports in the verilog module to the hardware on the Xilinx board.

12) Select your top verilog module and double click configure target device in the processes section of the design panel.



13) Once complete, a prompt will appear where you will press ok and impact will open up. From there. Once impact opens and the Xilinx board is set up and plugged into the computer, select boundary scan and then initialize chain. Once complete select program in the side panel and test the board. (**In impact, double-clicking the device and assigning the BIT file is necessary**).

14) Due to hardware limitations, even though the divider has 16 bit accuracy for both the quotient and remainder, only 8 bits are visible at a time, so in order to see all 32 bits, you must change the constraints file to see either 8 bits of the quotient or 8 bits of the remainder.

Currently the LEDs and pin-headers are programmed to show bits 0-7 (LSB) of the remainder, you can see the other 8-bits by changing each of the array places to the numbers 8-15 (MSB). To view the quotient comment the “fractional[#]” code and uncomment the “quotient[#]” code. After you make any changes, you must redo mapping and routing the components and generating the file, so you must double-click the configure target device in the design panel once more and repeat step 13 if you did not save your project previously. If you did save and you see two chips on the boundary scan screen, then press program and test your program.

```

module counter_program(
    input clk,
    input sig,
    output rfd,
    output [15:0] quotient,
    output [15:0] fractional,
    output khz_clk,
    output hz_clk
);
//declarations
reg [15:0] count = 16'b0000000000000000;
reg [15:0] tempc = 16'b0000000000000000;
reg [15:0] count2 = 16'b0000000000000000;
reg [15:0] tempc2 = 16'b0000000000000000;
reg [15:0] sample = 16'b0000000000000000;
reg [15:0] temple = 16'b0000000000000000;
reg khz_clk = 1'b0;
reg hz_clk = 1'b0;
reg [15:0] count3 = 16'b0000000000000000;
reg [15:0] tempc3 = 16'b0000000000000000;
div divider (
    .clk(clk), // input clk
    .rfd(rfd), // output rfd
    .dividend(sample), // input [15 : 0] dividend
    .divisor(count), // input [15 : 0] divisor
    .quotient(quotient), // output [15 : 0] quotient
    .fractional(fractional)); // output [15 : 0] fractional

always @ (posedge sig)
begin // count or reset
    if (count != 16'b1111111111111111 )
    begin
        temple = sample + 1;
        sample = temple;
    end
    else
    begin
        temple = temple;
        sample = sample;
    end
    end
    if (fractional <= 16'b0000000000010000 && count == 16'b1111111111111111 && khz_clk ==
1'b1 && hz_clk != 1'b1)
    begin

```



```

sample = 16'b0000000000000000;
temple = 16'b0000000000000000;
end
else
begin
if (fractional <= 16'b0000000000010000 && count == 16'b1111111111111111 && khz_clk !=
1'b1 && hz_clk == 1'b1)
begin
sample = 16'b0000000000000000;
temple = 16'b0000000000000000;
end
else
begin
sample = sample;
temple = temple;
end
end
if (khz_clk == 1'b1 && hz_clk == 1'b1)
begin
sample = 16'b0000000000000000;
temple = 16'b0000000000000000;
end
else
begin
sample = sample;
temple = temple;
end
end
end

```

```

always @ (posedge clk)
begin//count, check if count is complete and engage dividers
if (count != 16'b1111111111111111 && sample != 16'b1111111111111111 &&
sample!=16'b0000000000000000 && khz_clk != 1'b1 && hz_clk != 1'b1)
begin
tempc = count + 1;
count = tempc;
end
else
begin
if (fractional <= 16'b0000000000010000 && sample!=16'b0000000000000000 && count ==
16'b1111111111111111 && khz_clk != 1'b1 && hz_clk != 1'b1)
begin
tempc = 16'b0000000000000000;

```

```

    count = 16'b0000000000000000;
    khz_clk = 1'b1;
end
else
begin
    if (fractional <= 16'b00000000000010000 && sample!=16'b0000000000000000 && count ==
16'b1111111111111111 && khz_clk == 1'b1 && hz_clk != 1'b1)
    begin
        tempc = 16'b0000000000000000;
        count = 16'b0000000000000000;
        khz_clk = 1'b0;
        hz_clk = 1'b1;
    end
    else
    begin
        if (fractional <= 16'b00000000000010000 && sample!=16'b0000000000000000 && count ==
16'b1111111111111111 && khz_clk != 1'b1 && hz_clk == 1'b1)
        begin
            tempc = 16'b0000000000000000;
            count = 16'b0000000000000000;
            khz_clk = 1'b1;
            hz_clk = 1'b1;
        end
        else
        begin
            tempc = tempc;
            count = count;
        end
    end
end
end
//engage hz clk
if (khz_clk != 1'b1 && hz_clk == 1'b1)
begin
    if (count2 != 16'b0000001111101000)
    begin
        tempc2 = count2 + 1;
        count2 = tempc2;
    end
    else
    begin
        tempc2 = tempc2;
        count2 = count2;
    end
end

```

```

end
if (count2 == 16'b00000001111101000 && count != 16'b1111111111111111 )
begin
    tempc2 = 16'b0000000000000000;
    count2 = 16'b0000000000000000;
    tempc3 = count3 + 1;
    count3 = tempc3;
    if (count3 == 16'b00000001111101000 && count !=16'b1111111111111111 )
    begin
        tempc3 = 16'b0000000000000000;
        count3 = 16'b0000000000000000;
        tempc = count + 1;
        count = tempc;
    end
end
else
begin
    tempc = tempc;
    count = count;
    tempc3 = tempc3;
    count3 = count3;
end
end
else
begin
    tempc2 = tempc2;
    count2 = count2;
    tempc = tempc;
    count = count;
end
end
else
begin
    tempc = tempc;
    count = count;
end
end
// engage khz clk
if (khz_clk == 1'b1 && hz_clk != 1'b1)
begin
    if (count2 != 16'b00000001111101000)
    begin
        tempc2 = count2 + 1;
        count2 = tempc2;
    end
end

```

```

else
begin
    tempc2 = tempc2;
    count2 = count2;
end
if (count2 == 16'b00000001111101000 && count != 16'b1111111111111111)
begin
    tempc2 = 16'b0000000000000000;
    count2 = 16'b0000000000000000;
    tempc = count + 1;
    count = tempc;
end
else
begin
    tempc2 = tempc2;
    count2 = count2;
    tempc = tempc;
    count = count;
end
end
else
begin
    tempc = tempc;
    count = count;
end
end
endmodule

```

```

#NET "quotient[0]" LOC = "D17";
#NET "quotient[1]" LOC = "AB4";
#NET "quotient[2]" LOC = "D21";
#NET "quotient[3]" LOC = "W15";
#NET "quotient[4]" LOC = "G7";
#NET "quotient[5]" LOC = "H6";

```

```
#NET "quotient[6]" LOC = "D1";
#NET "quotient[7]" LOC = "R7";
//NET "quotient[0]" LOC = "G7";
//NET "quotient[1]" LOC = "H6";
//NET "quotient[2]" LOC = "D1";
//NET "quotient[3]" LOC = "R7";
NET "khz_clk" LOC = "V19";
NET "hz_clk" LOC = "B3";
NET "clk" LOC = "AB13";
NET "sig" LOC = "A3";
NET "sig" CLOCK_DEDICATED_ROUTE = FALSE;
//NET "quotient[0]" LOC = "D17";
//NET "quotient[1]" LOC = "AB4";
//NET "quotient[2]" LOC = "D21";
//NET "quotient[3]" LOC = "W15";
//NET "fractional[0]" LOC = "G7";
//NET "fractional[1]" LOC = "H6";
//NET "fractional[2]" LOC = "D1";
//NET "fractional[3]" LOC = "R7";
NET "fractional[0]" LOC = "D17";
NET "fractional[1]" LOC = "AB4";
NET "fractional[2]" LOC = "D21";
NET "fractional[3]" LOC = "W15";
NET "fractional[4]" LOC = "G7";
NET "fractional[5]" LOC = "H6";
NET "fractional[6]" LOC = "D1";
NET "fractional[7]" LOC = "R7";
```