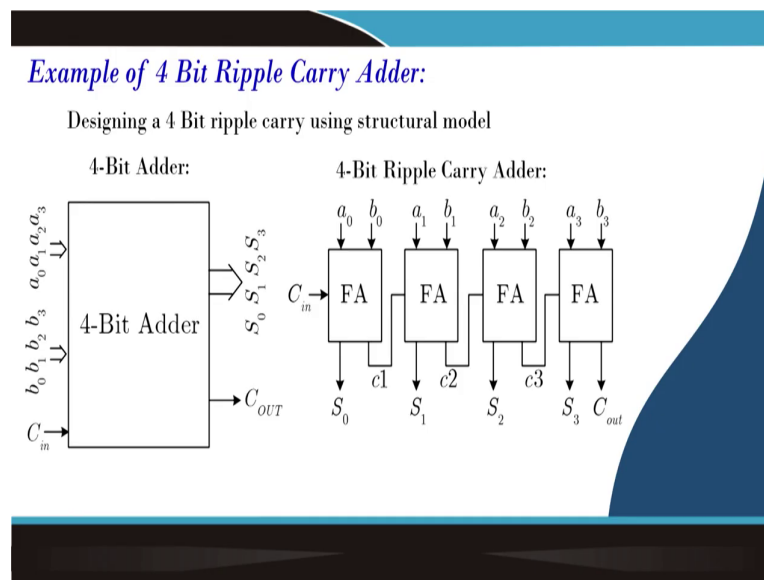


Digital Control in Switched Mode Power Converters and FPGA - based Prototyping
Prof. Santanu Kapat
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Module - 07
Introduction to Verilog and Simulation Using Xilinx Webpack
Lecture - 66
Simulation of Verilog-HDL-based Design using Xilinx Webpack - II

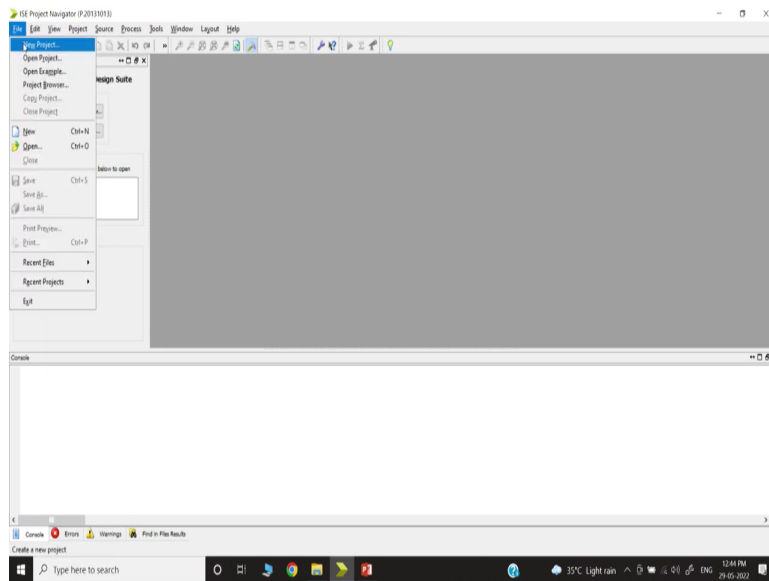
Welcome. So, this is the continuation of the previous lecture. Here, we want to show using the Xilinx ISE simulator, we want to demonstrate.

(Refer Slide Time: 00:34)



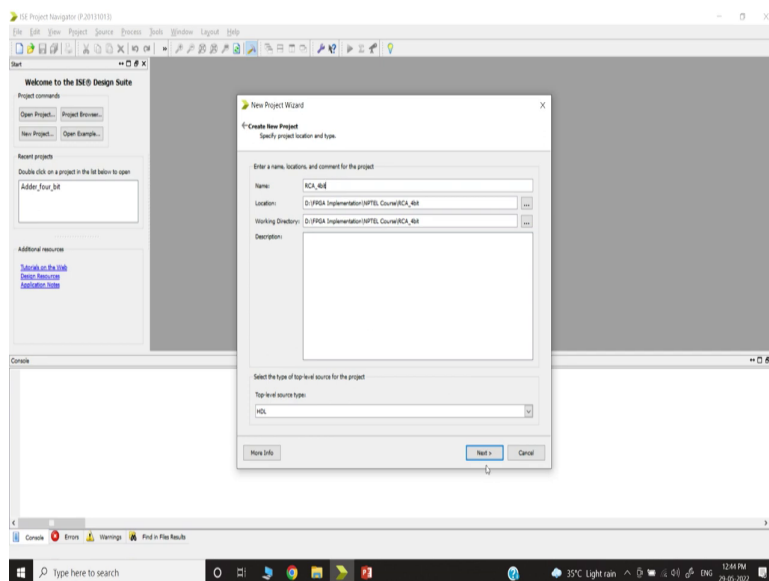
So, we want to consider one 4-bit ripple carry adder as an example here. So, we want to implement using Verilog HDL. So, let us go to our Verilog block.

(Refer Slide Time: 00:45)



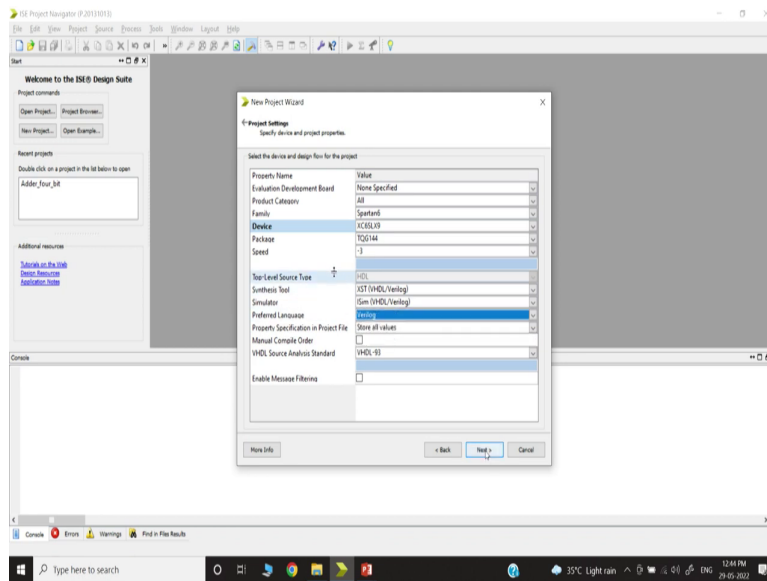
And, you know we are going to implement this Verilog. So, let us open a project, I mean what we have discussed already.

(Refer Slide Time: 00:56)



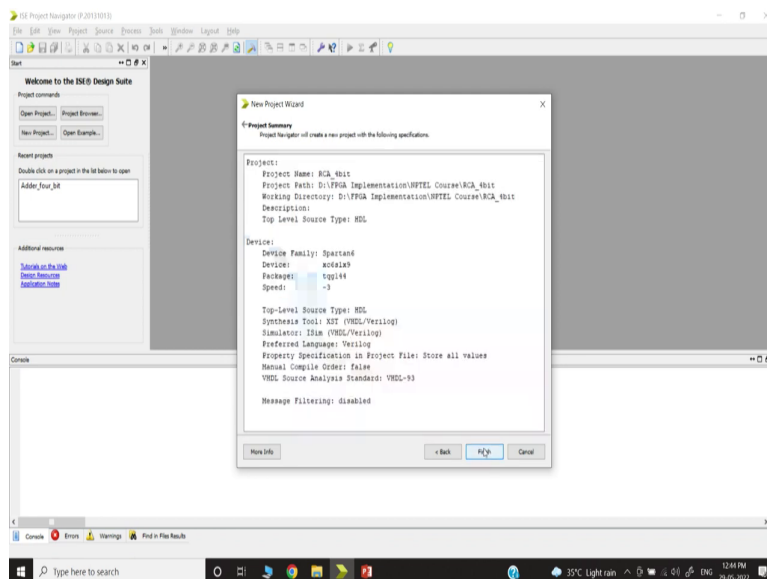
So, in this, we will create a new project and here the name of the project is I will say 4 ripple carry adder, ripple carries adder it is like a 4-bit, 4-bit ripple carries adder. So, this is my project name.

(Refer Slide Time: 01:13)



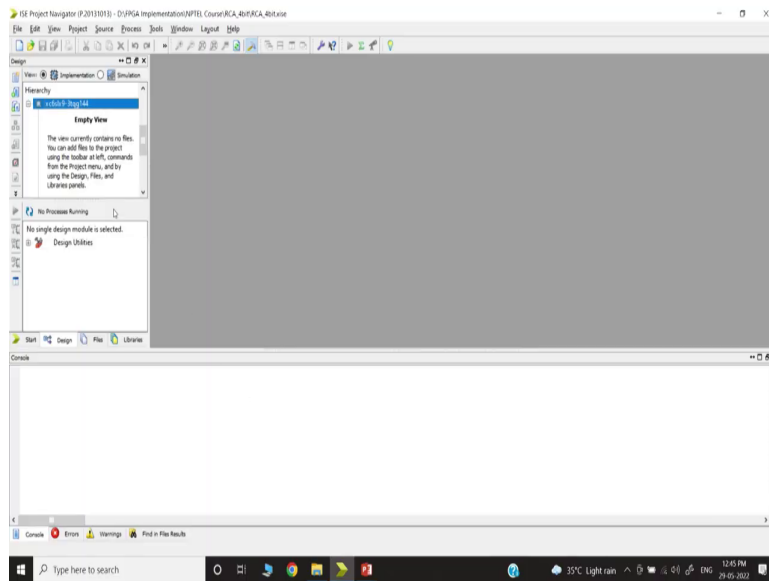
Next, I will go to this Spartan device and we are talking about the Spartan6 ok. And, this is a package we will be using, I have discussed and you can take VHDL or Verilog, but we will be using Verilog. All other blocks keep it as it is.

(Refer Slide Time: 01:30)



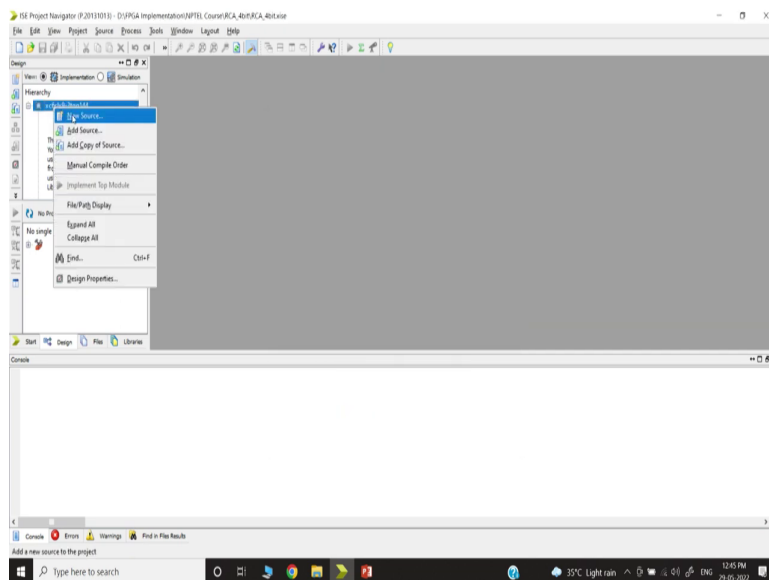
Then, this is the description of the project, the summary you can double check because even though we are simulating here; that means, in the simulation, you can arbitrarily take any device. But, we want to make sure that when you do the harder implementation, the device must match, the device as well as the package ok, and also the device family.

(Refer Slide Time: 01:53)



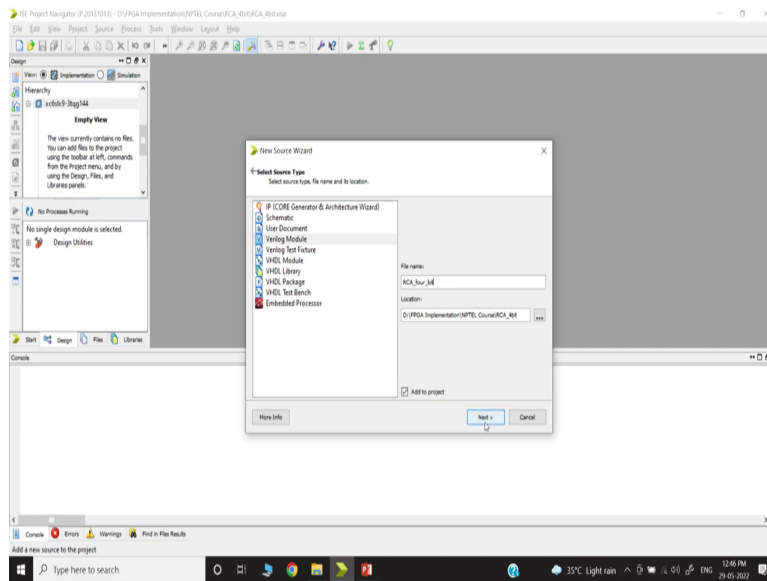
Next, now the project is created. So, you can see this project is like an IC, where we want to implement a 4-bit adder. So, in the 4-bit adder, if you go back to our presentation; so, we need vector data, 4-bit a, 4-bit b, carry in scalar, some vector 4-bit, and carry out scalar. So, here let us know you know just a minute.

(Refer Slide Time: 02:21)



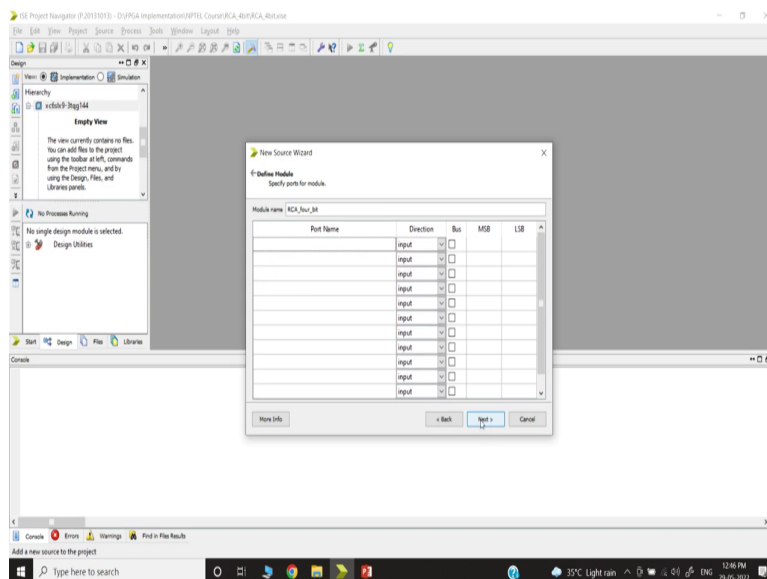
So, if you right-click, New Source.

(Refer Slide Time: 02:22)



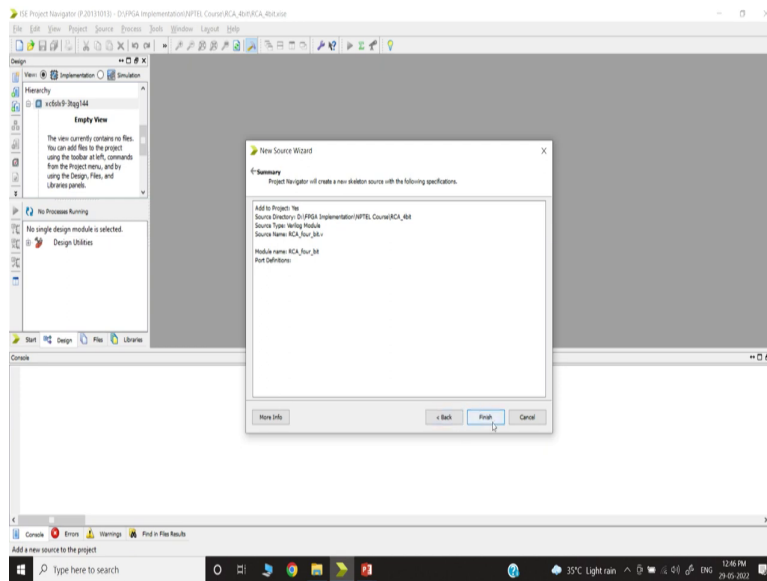
So, we want to create a Verilog module and we call it an RCA. So, RCA four-bit, four-bit ripple carry adder. So, this is a Verilog file name and it is different from our location, the project name.

(Refer Slide Time: 02:44)



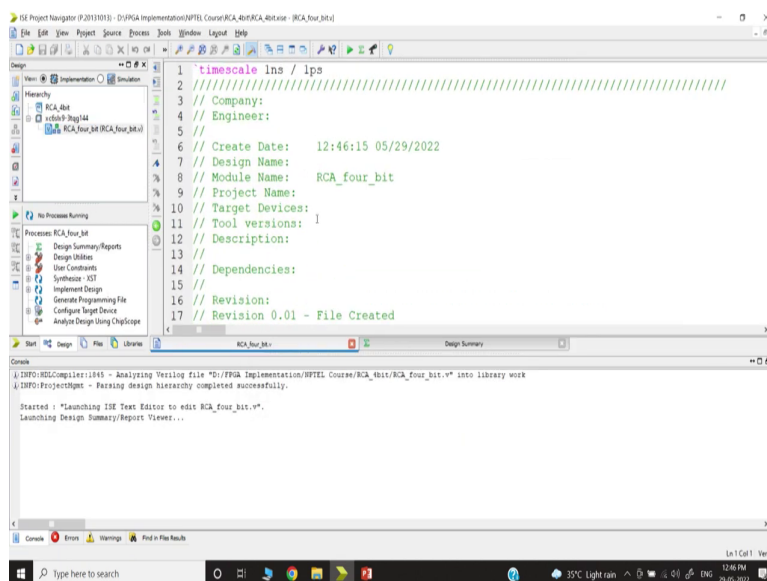
Next, I am not specifying any input-output.

(Refer Slide Time: 02:47)



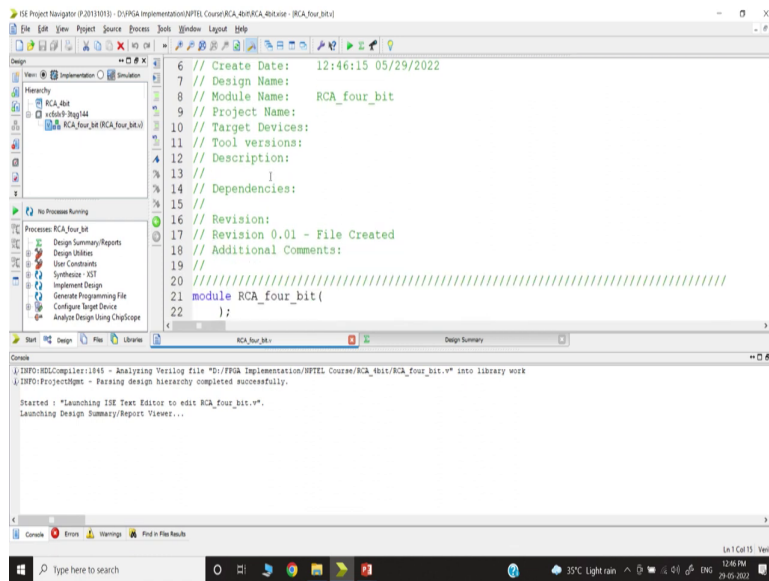
So, you can go ahead and this is a dot v file created.

(Refer Slide Time: 02:50)



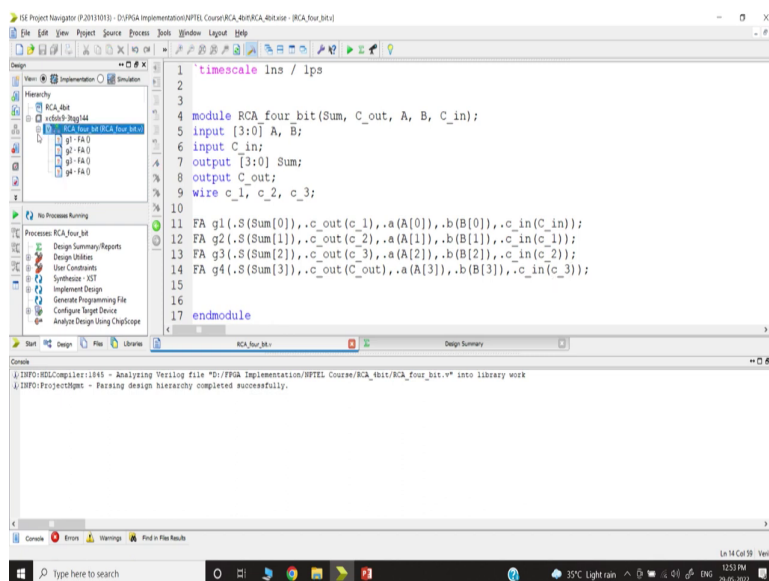
Now, I want to zoom in. So, I have discussed the time scale here, it is just for simulation purposes. So, it is 1 nanosecond the time. So, any unit of time you provide will be just multiplied by you know 20 nanoseconds and 30 units, which will be multiplied by 1 nanosecond.

(Refer Slide Time: 03:07)



And, these are all comments.

(Refer Slide Time: 03:11)



So, you can remove this line ok. So, you can remove this line with no issue. So, you can remove this line. Now, this is a four-bit ripple carry adder. In this ripple carry adder, if you look at the port; that means, if you go to our presentation, and I am taking a 4-bit data input b 4-bit data input, C in, C out S; that means, we are writing we can sum we can write sum like Sum, then we can write C out, then we can write A, B, C in.

So, this is the four-bit ripple, ripple carries adder has four-bit data input A and B and the sum will be four-bit carryout and carry in are scalar. Then, we have to define the what is the input.

So, our input is like initially, we are defining the vector input which is a four-bit number and those are A comma B.

The A and B both are four-bit numbers. Another input is the scalar, we have to specify which is nothing but our C in. Another output will be vector output will be our sum which will be Sum and then another output is C out ok. So, this is a port declaration. We have declared which are input, output, vector, and so on.

Now, if you go to the diagram, you see there are three internal c_1 c_2 c_3 . So, those things will make a wire; that means, we will make sorry c_1 c_2 c_3 ; that means, we will define wire like a c_1 , c_2 , and c_3 . There are three wires. Next, what we can see in the diagram?

That means, if we look at the diagram first, the first full adder will have a carrying input sum; that means, we have to go; that means, let us instantiate. That means, we have a full adder, assuming that we will make a full adder block, full adder then the instant name may be gate 1. Now, inside the full adder, we are assuming there will be a sum, the full adder will have a sum.

Full adder will also have to carry out C_0 , just to make you know you should not confuse, then full adder will have a data a, data b, and full adder will also have a carry, we are writing in small. So, here it is. So, here you can write c out, c small. So, this is for full adder carry-in carry out ok. Now, this is a structure of the full adder and we want to create a full adder circuit and we are making a dot; that means, we are calling by name, not by sequence.

So, we are just making a name, we will internally connect. This name must be consistent with the actual full adder that we are going to build. Now, we want to connect the local variable here; that means, the local variable here. So, local variable here, local variable. So, if you take the first gate, what is the sum out of the first full adder? If you go to the first full adder, the first full adder sum out is the S_0 ; that means, it will be we have already defined a sum that will be Sum 0.

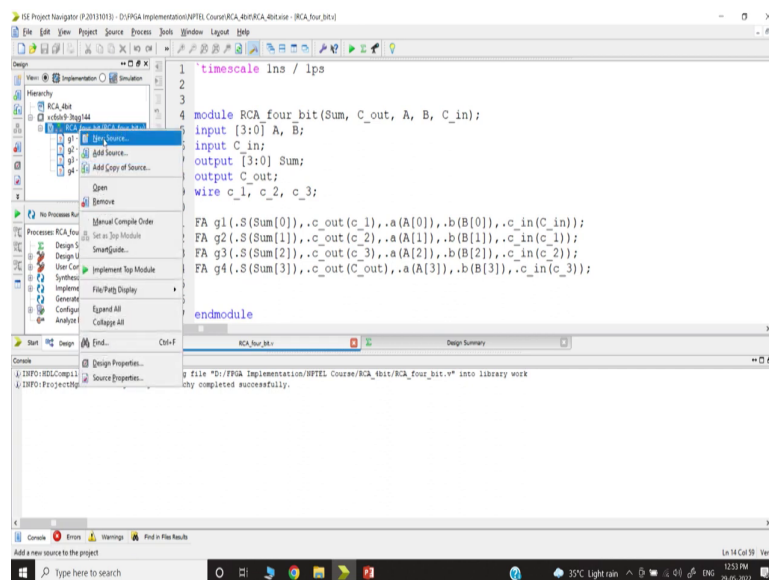
Because, here we mention Sum 0 is the LSB and Sum 3 is the MSB, Sum 0. What will be c out of this full adder? It will be the wire connection of c_1 ; that means, let us copy c_1 . What will be that first data? It will be A_0 ; that means, it will be A_0 LSB. Then, what will it will be B_0 , and, the carry-in will be the same as the actual carry which is here; that means, we will mention this one as the carry-in. So, this is the local value.

Now, we have made it, next I will just copy and paste this, next line. Here, we have to change the second gate; that means, the second full adder. See, the second full adder naturally will be the first S Sum 1, this will be bit 1, bit 1. And, this will be if you go to that block, you will find the second case the carry-in is the carry-out of the first, and its carry-out will be c 2 in the wire. So that means, its carry-in will be c 1 which is the local variable which is the wire variable, and its carry-out will be c 2 ok.

Then, we will copy paste copy-paste, and make it. So, g 3, then we will make Sum 2, this will be c 3, this will be A 2, this will be B 2, this will be c 2. Why c 3? Because, if you go to the diagram c 3 is the out and c 2 is the in, c 3 is the out, and c 2 is the in. Again, I am copying it here. So, 4 full adders are connected in sequence. So, it will be the fourth one.

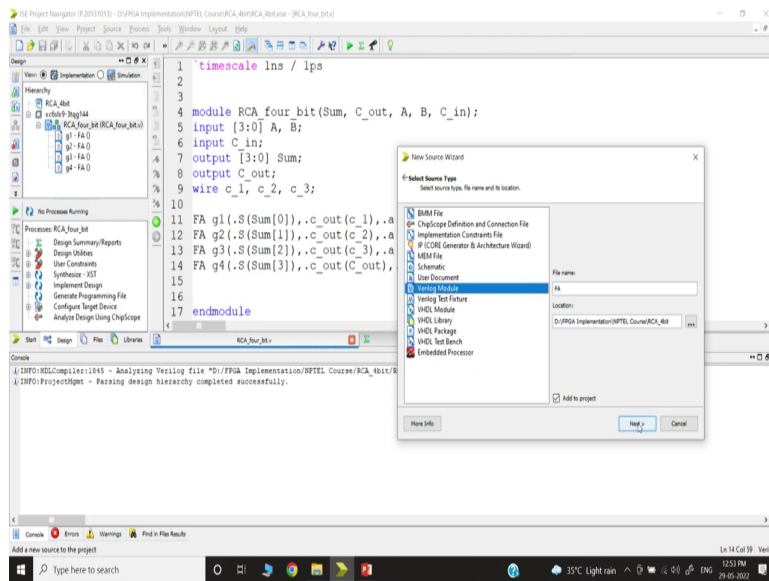
The fourth one will be Sum 3, which is the MSB, it will be the actual carry-out because this will be the output of the fourth full adder will be the actual carry out. And, this will be the third that is the MSB, MSB and this will be the c 3. So, now this circuit is complete, but it is incomplete because it needs a full adder, you see there is a full adder. So, you have to define.

(Refer Slide Time: 09:44)



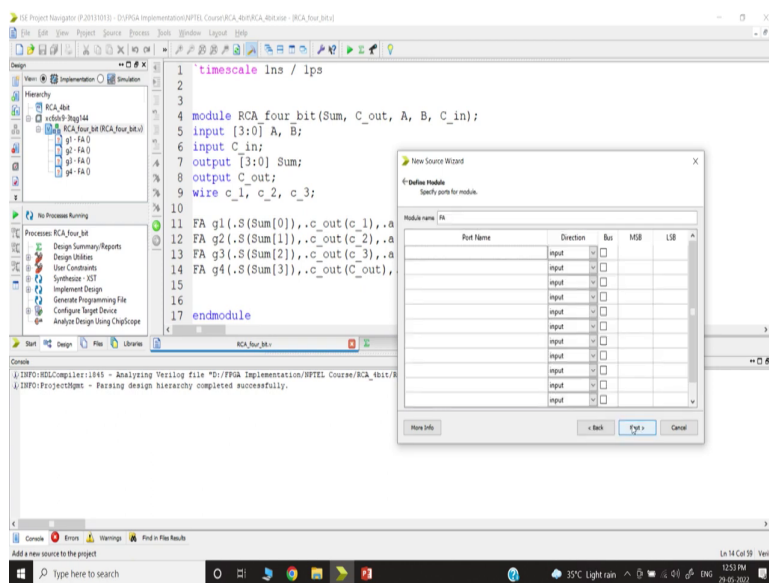
That means, now we are going for a circuit.

(Refer Slide Time: 09:46)

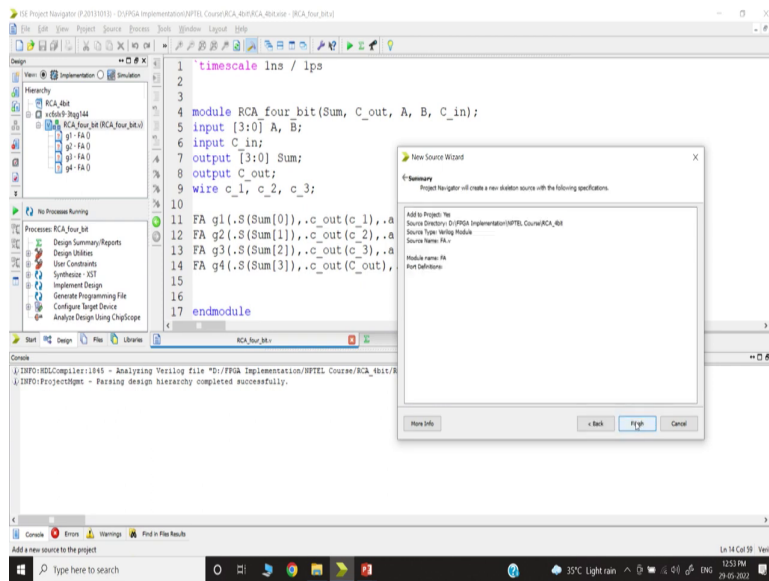


And we have to keep the same name; that means, we are creating a Full Adder.

(Refer Slide Time: 09:57)

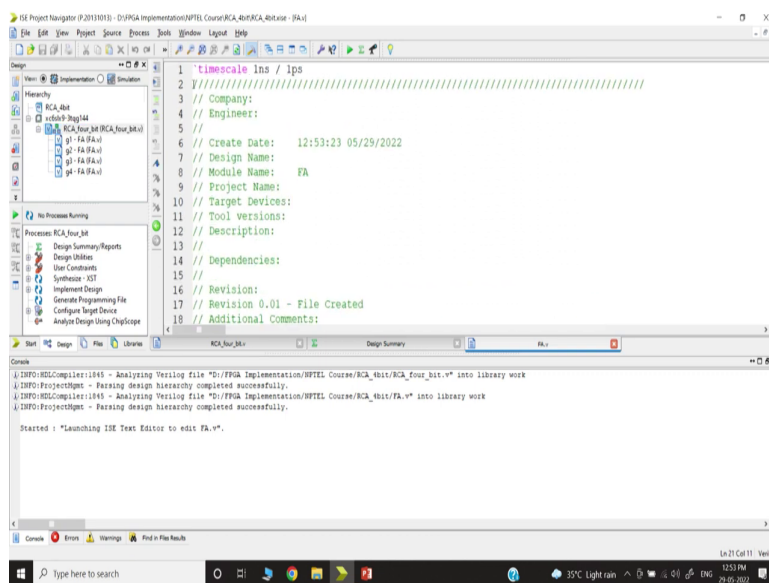


(Refer Slide Time: 09:57)



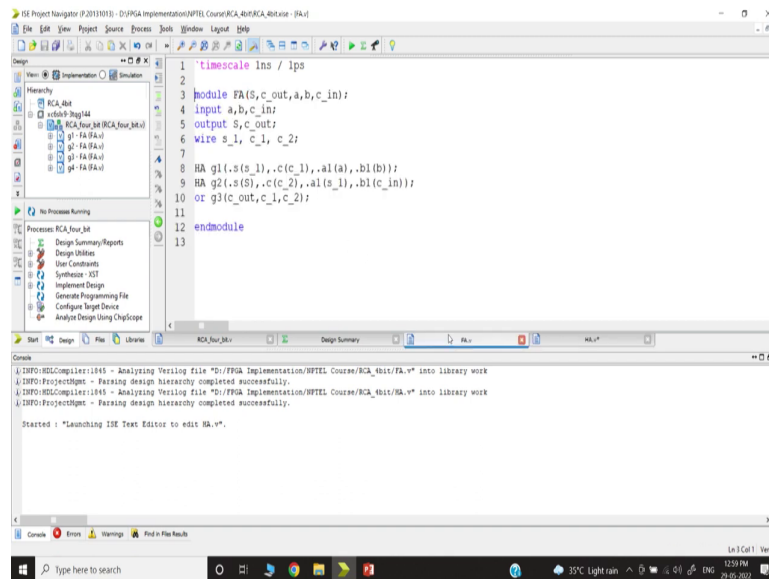
And, this is a Verilog module. So, full adder. Now, we have created a full adder.

(Refer Slide Time: 09:58)



Again, we can remove this line, a redundant line.

(Refer Slide Time: 10:05)



So, we have to make sure that we should name the same thing. We cannot change the name; that means, we just copy and paste this and if we go here we will use only we will use this S, not internal. So, we will remove this. We will use this c dot, remove this. Then, you will use this a, remove this and we will use b, we will remove this, will you c in, remove it.

Now, these are our names. So, the name must be consistent. So, this is the name. We have to declare the port. So, here all are scalar. So, there is no problem, with input. So, input is what? a comma b comma c in. And what is the output? Output is S comma c out. So, the name must be consistent. Now, we have to write the code. What is the full adder? Again, a full adder consists of 2 half adders.

(Refer Slide Time: 11:27)

Connecting Ports to External Signals by ordered list

Connecting by ordered list

```

module FA(S,C,a,b,c_in);
    ...
    HA ha1(s_1,c_1,a,b);
    HA ha2(S,c_2,s_1,c_in);
    ...
endmodule

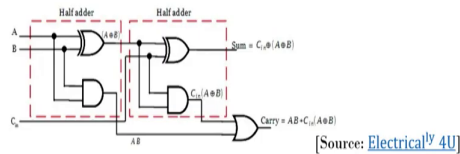
```

Half Adder Module

```

module HA(S,C,a,b);
    input a,b;
    output S,C;
    xor g1(S,a,b);
    and g2(C,a,b);
endmodule

```



That means if you go to our block, the next level. If you look at this diagram; that means, half adder, so this is a half adder and half adder will make. So, half adder input a and b are the input, the output is I am taking some small s and small c, small s, small c, small a, small b ok. So, I am creating a half adder; that means, half adder I am calling.

Half adder, let us say we are using 1 instance, where we are using sum, again I am using a dot-bracket I will connect the link with that. Then, c the half adder c, comma dot half adder a, you know you can say a local variable. So, it may be the same, or I will just avoid the a1 dot, I will say dot b1 ok. So, this will take half adder and will have 2 inputs, and 2 outputs. So, this is done.

Now, you have to link. So, we will link with that. First half adder if you check, if you go the first adder half adder inputs are the same as that a, b. So that means, we will go a, b, inputs are a, b; that means, we will simply connect local variable a, local variable b. And, first half adder, what is the carry in; sorry carry out?

So, some of this first half adder; means, there will be s, which means internal variable. So, we call it s 1, the sum of first that is a wire connection and c 1 is the output. So that means, we have to define wire here s 1 comma c 1. There will be c 2, s 2 may not be needed because the sum will be the actual sum.

So; that means, it will be what? It will be c 1 and this will be s 1, the local variable. Next, we will copy-paste. We will go next. Here, we will make g 2. Now, if you go to the diagram, the

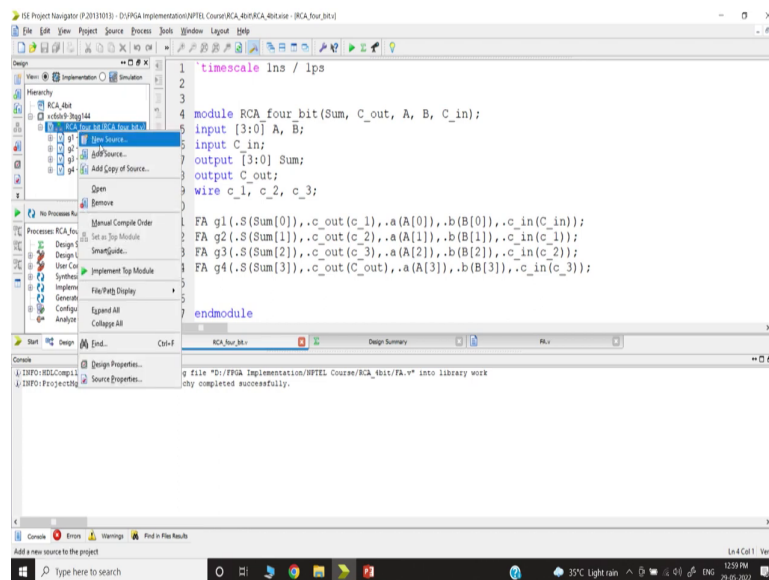
second half adder input will be the sum 1 and the carry-in; that means, the input will be sum 1 which is the s₁, 1 input.

Another will be carried which will be the actual carry-in which means, this carry-in, is the actual carry. Then, what is the output? The output will be if you go output to the second will be the actual output; that means, this output will be the actual sum. So, this output will be the actual sum, and the carry we are mentioning 2, carry 2 output.

Then, if we go further the actual carrier will be or of this carryout, the carry out of the second half adder and first adder half adder; that means, we will use the gate, gate 3. Here, we have to make sure we have to maintain the sequence. The output of the or gate will be c_{out}, c_{out}.

Then, input to the or gates is c₁ comma c₂ that is it. Now, this level module is implemented. Next, we have to create this half adder; that means, if you go inside, you will see there is a half adder also. So, now, with this full adder; that means if you want to synthesize let us say first block; that means, you know because we have to go actually, I am coming from top to bottom. I should have gone from bottom to top, but that is ok.

(Refer Slide Time: 15:37)

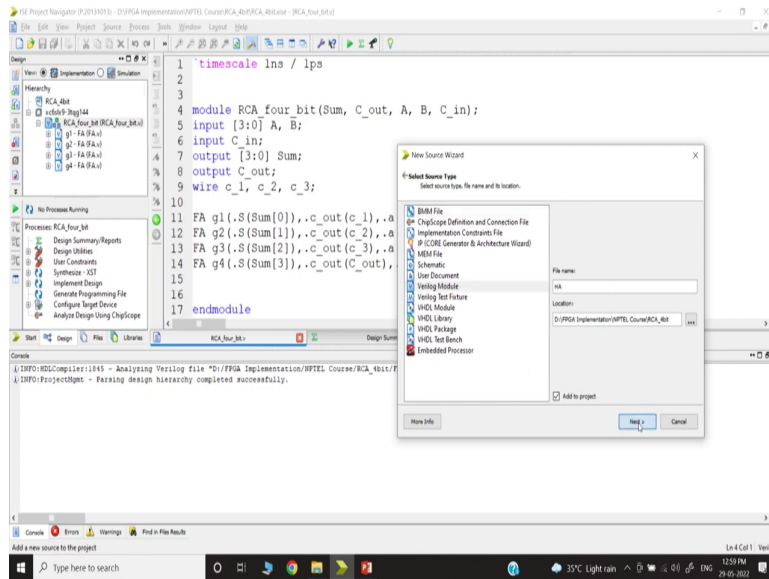


```
1 timescale 1ns / 1ps
2
3
4 module RCA_four_bit(Sum, C_out, A, B, C_in);
5   input [3:0] A, B;
6   input C_in;
7   output [3:0] Sum;
8   output C_out;
9   wire c_1, c_2, c_3;
10
11   FA g1(.S(Sum[0]), .c_out(c_1), .a(A[0]), .b(B[0]), .c_in(C_in));
12   FA g2(.S(Sum[1]), .c_out(c_2), .a(A[1]), .b(B[1]), .c_in(c_1));
13   FA g3(.S(Sum[2]), .c_out(c_3), .a(A[2]), .b(B[2]), .c_in(c_2));
14   FA g4(.S(Sum[3]), .c_out(C_out), .a(A[3]), .b(B[3]), .c_in(c_3));
15
16 endmodule
```

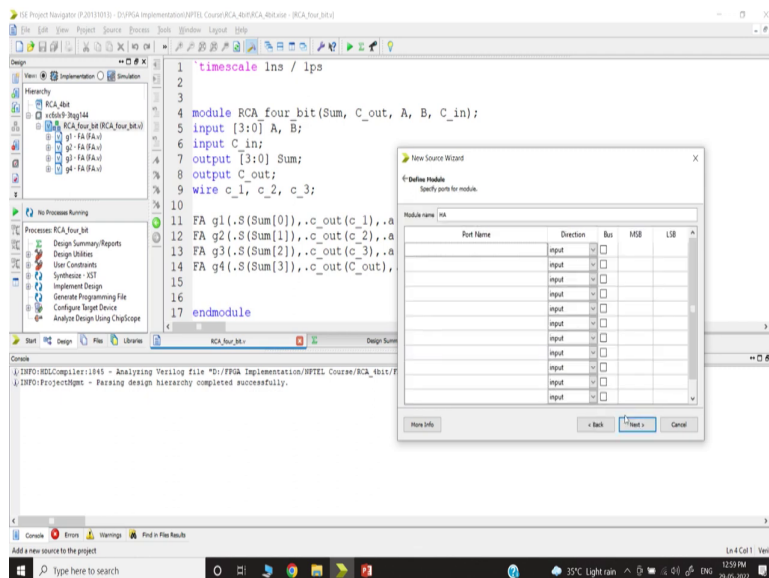
INFO: HDLCompiler: File "D:/FPGA Implementation/NPTEL Course/RCA_bit/FA.v" into library work chy completed successfully.

Now, I want to create another module or I can just create another module. You can click anywhere, New Source.

(Refer Slide Time: 15:44)

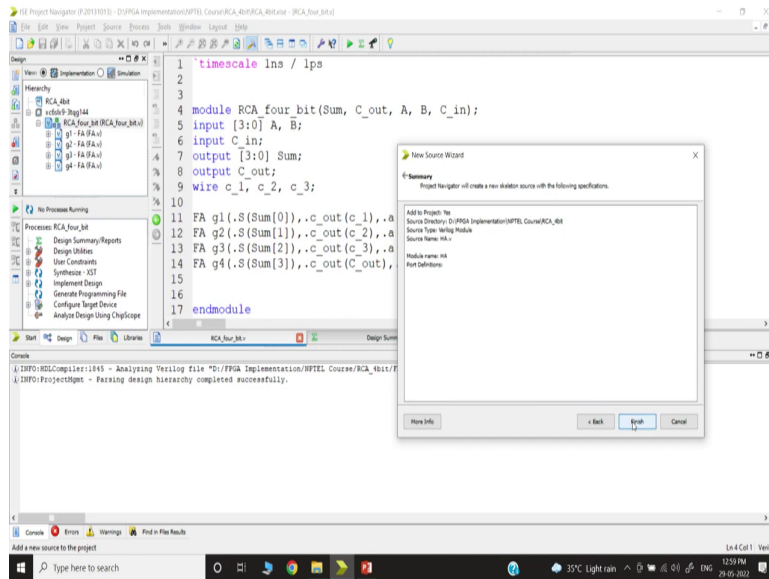


(Refer Slide Time: 15:51)

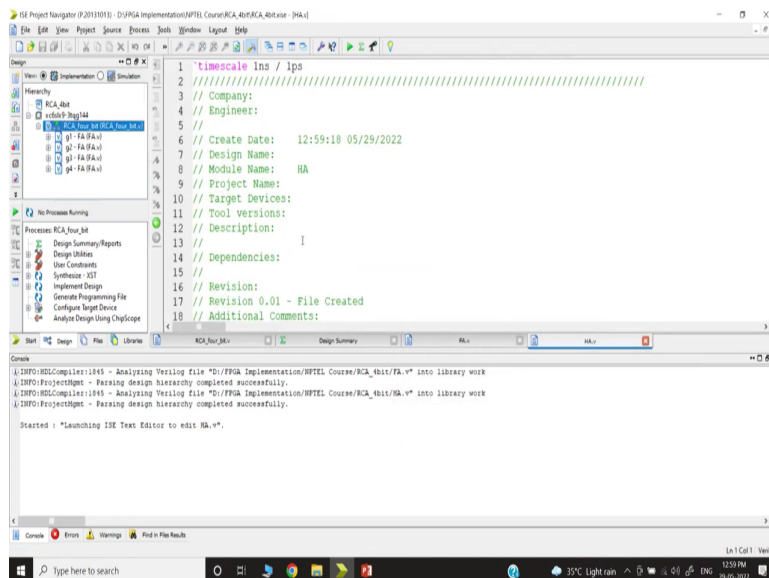


I am telling you this is my half-adder. It should be a Verilog Module ok.

(Refer Slide Time: 15:52)

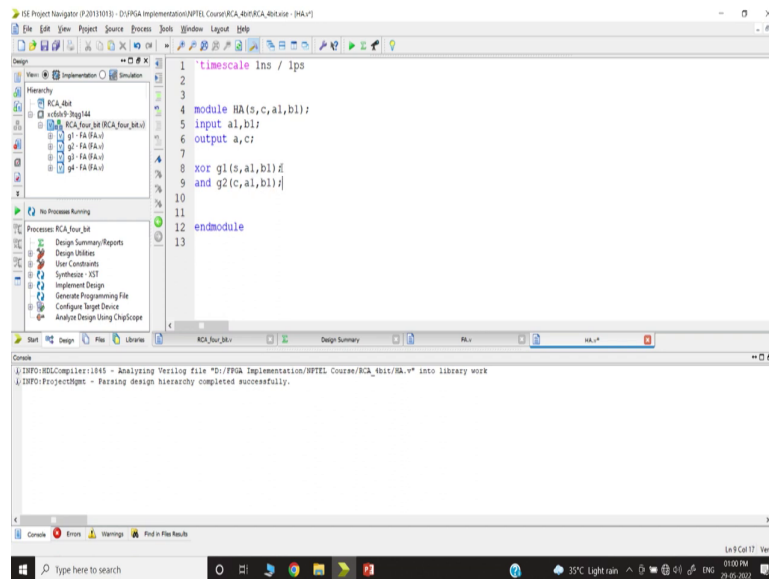


(Refer Slide Time: 15:53)



So, in the half adder, again I can remove this redundant part; what is there in the full adder?

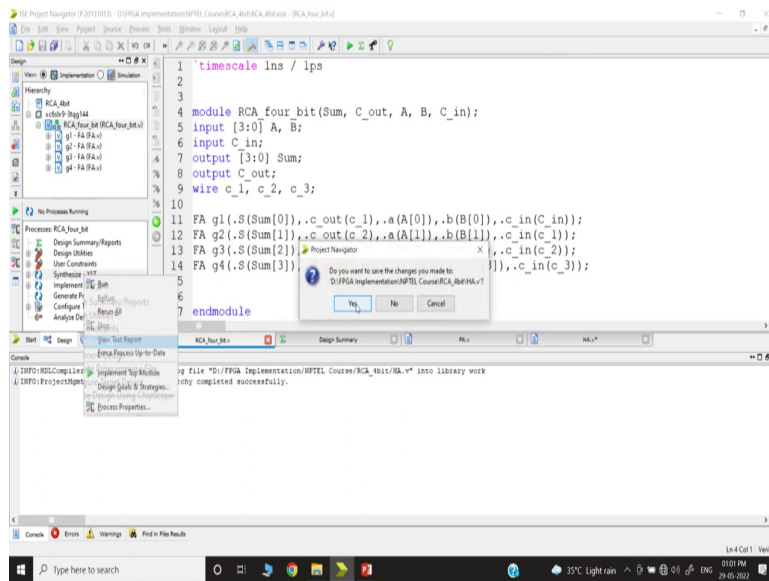
(Refer Slide Time: 16:00)



So, if you go to the full adder, the half adder consists of $s = a \oplus b$, $c = a \cdot b$. So, it has to be $s = a \oplus b$. So, we have to make sure that we make it consistent; that means, $s = a \oplus b$; that means, this dot symbol $s = a \oplus b$ ok. And, what is the input? At the a comma b . And, what is the output? The s comma c . You may or may not give any space because it will be ignored. Now, what is the half-adder circuit?

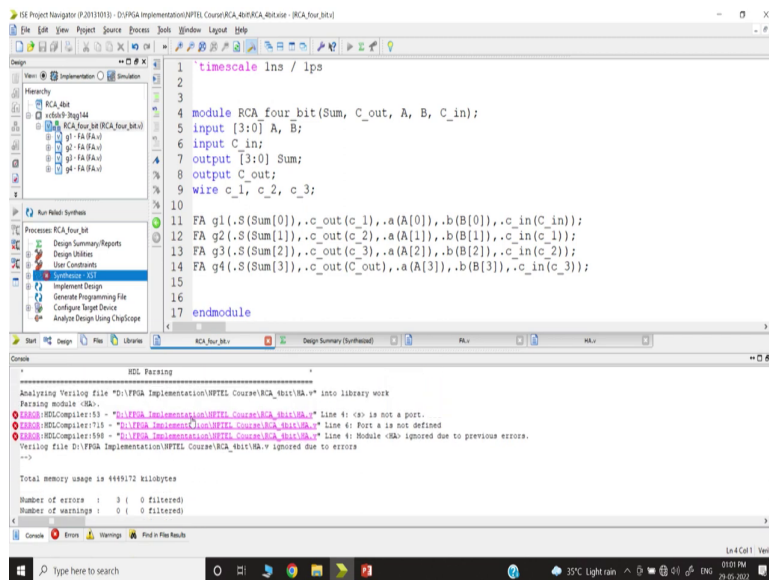
Again, if you go to the half adder, you can see here half adder is nothing but xor for the sum and the carry; that means, I can get xor gate 1, which will be my sum comma a comma b and or sorry and gate 2 which will be my carry out; that means, c comma a comma b . So, that is it, looks like we are all set. Now, let us try to first synthesize.

(Refer Slide Time: 17:46)



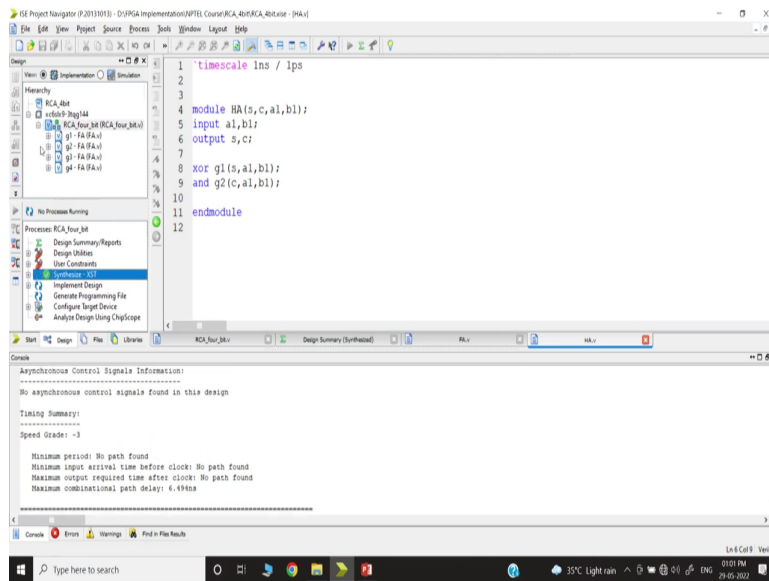
So, we synthesize this.

(Refer Slide Time: 17:52)



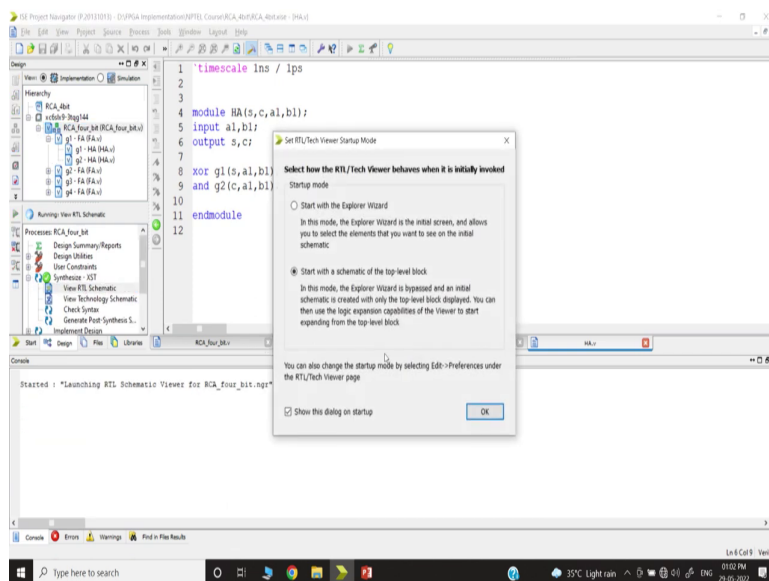
Yes, we will synthesize. Let us see because ok. So, there may be some port line.

(Refer Slide Time: 18:00)



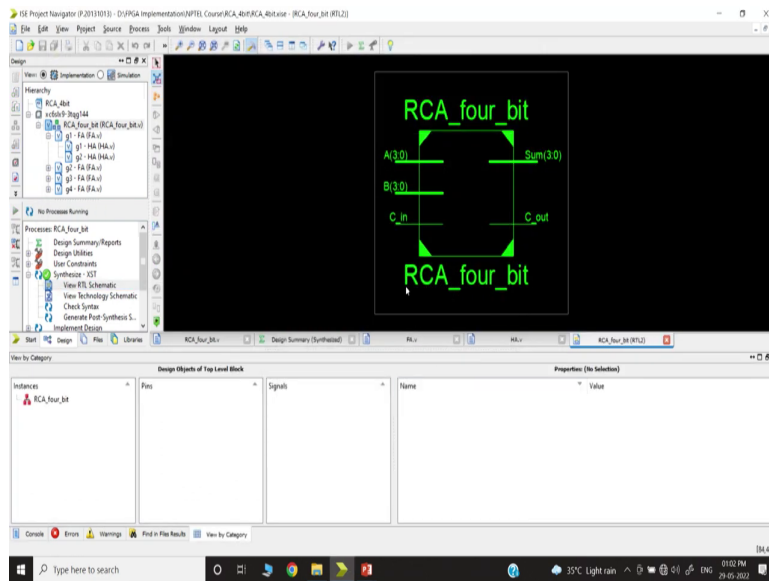
We have to check where it is, half adder line 4, module, end module not defined; oh I made a mistake. So, sorry that was a typo. So, now, it is synthesized.

(Refer Slide Time: 18:33)



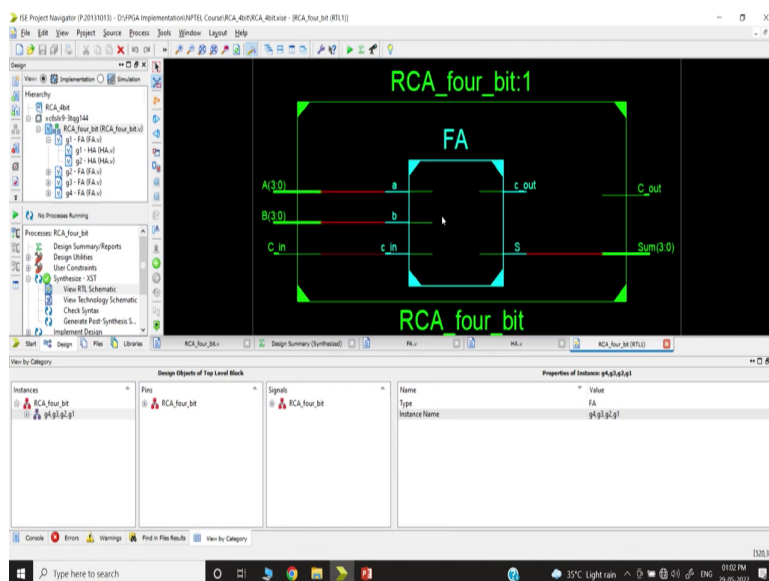
And, you give you see each block. So, first I want to see the RTL block; that means, you go to the RTL block, View the RTL block.

(Refer Slide Time: 18:41)



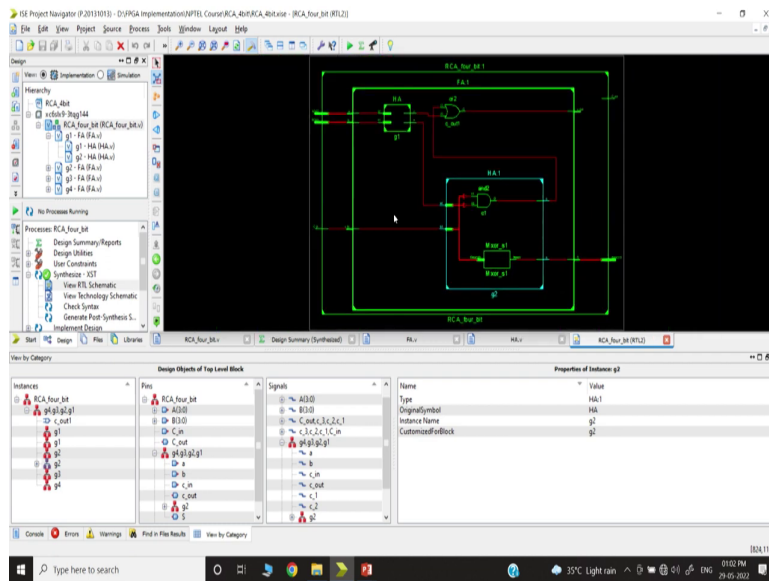
So, is a ripple carry adder.

(Refer Slide Time: 18:43)



And, if you go inside each block; that means, you know if I again open the RTL block. This is a four-bit ripple carry adder. Go inside, you will have such a full adder.

(Refer Slide Time: 18:58)



Then, you have a half adder and so on. So, it is designed. If you go further half adder, you will find that half adder consists of you know all these blocks. So that means, we are all set, we have designed our half-adder and all these RTLs are ready.

(Refer Slide Time: 19:20)

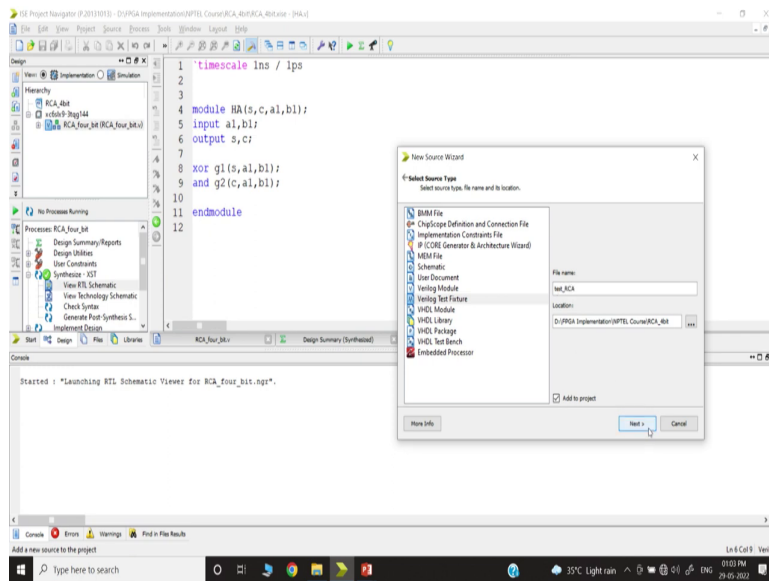
```

1 timescale 1ns / 1ps
2
3
4 module HA(s,c,a1,b1):
5   input a1,b1;
6   output s,c,r;
7   xor g1(s,a1,b1);
8   and g2(c,a1,b1);
9
10  endmodule

```

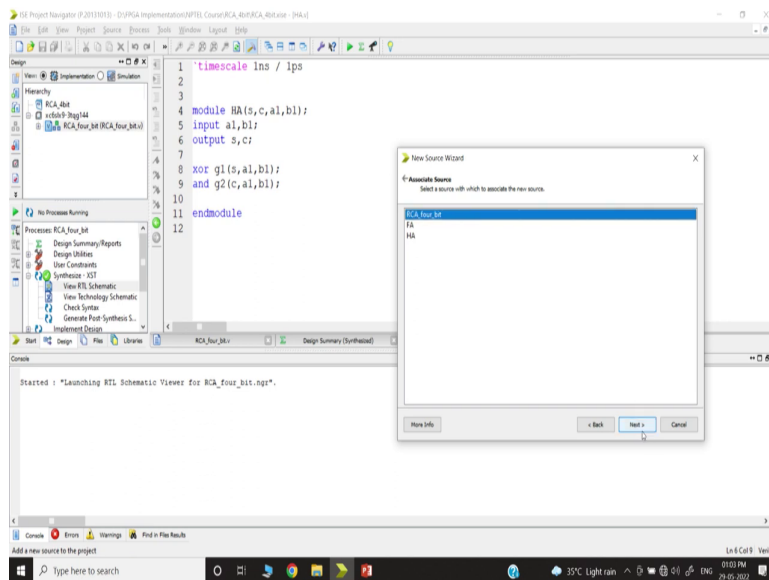
And, it looks fine. Now, you have to create a test signal. Now, we are creating a test signal. So, we have to test the circuit.

(Refer Slide Time: 19:31)



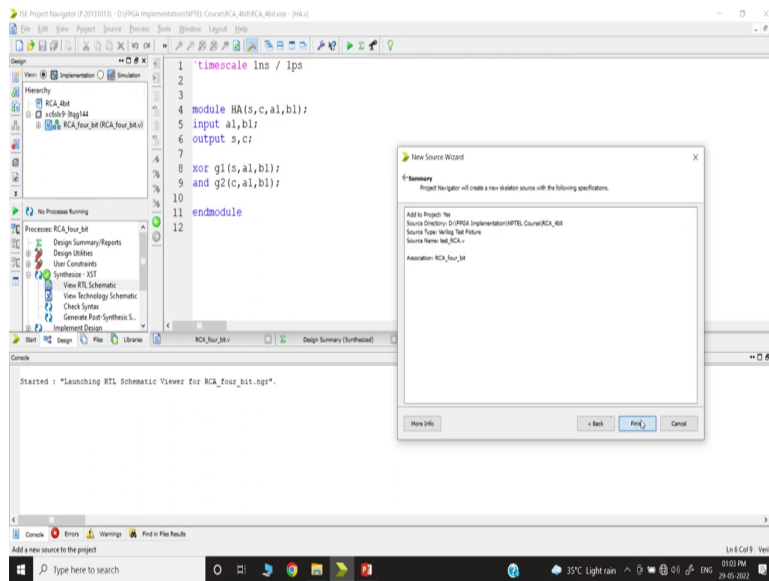
So, again you go to New Source. Then, you go to a Test Fixture and I get that test RCA.

(Refer Slide Time: 19:43)



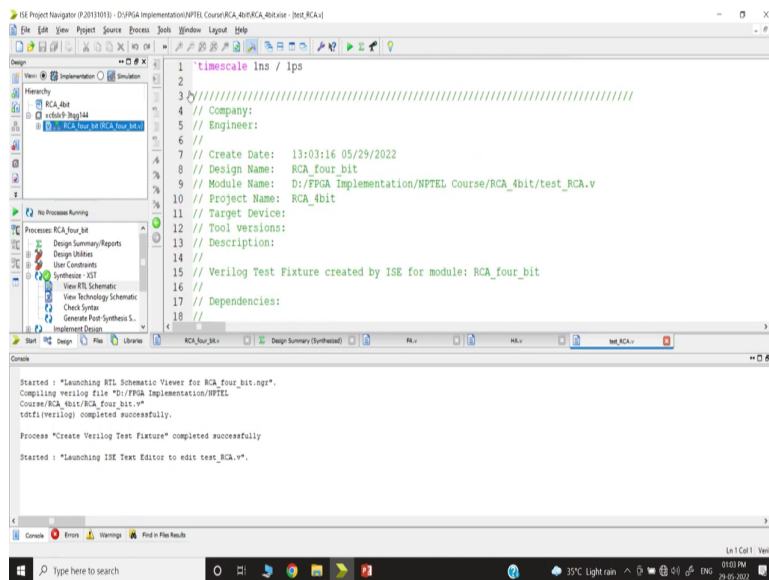
We want to test the RCA. There are many options. You can test separately half adder and full adder RCA.

(Refer Slide Time: 19:49)



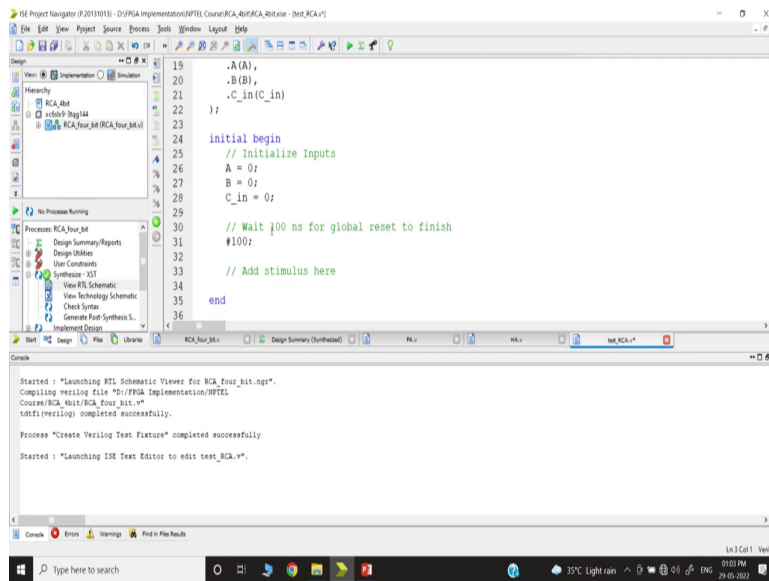
So, I have to test the full adder and full four-bit RCA.

(Refer Slide Time: 19:51)



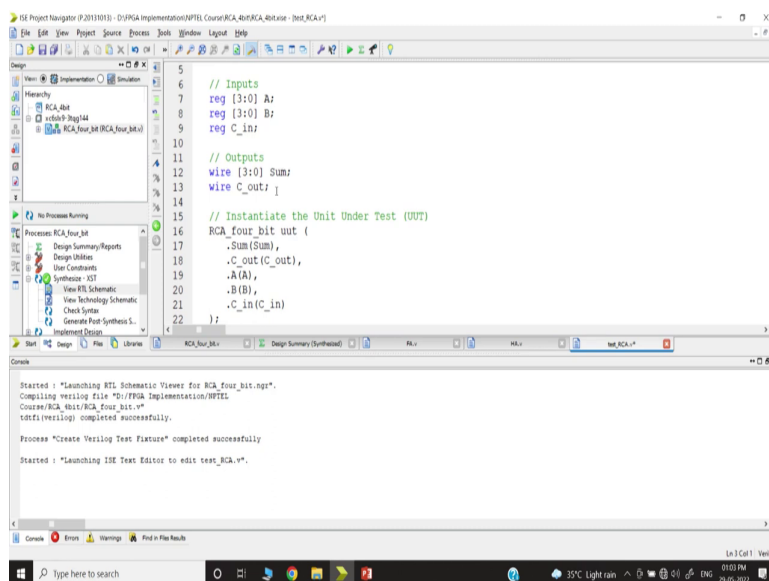
Now, again there will be a lot of redundant lines.

(Refer Slide Time: 19:59)



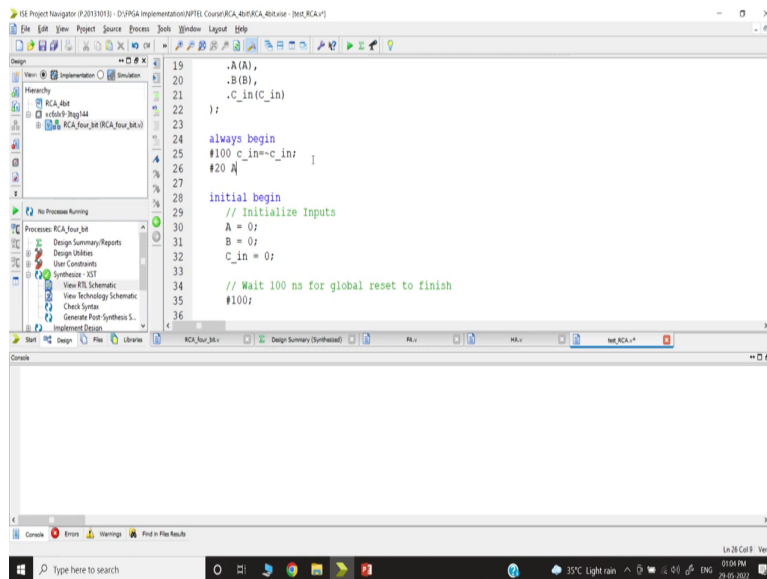
So, I can remove that. So, now, here there is data, you have to define the data.

(Refer Slide Time: 20:07)



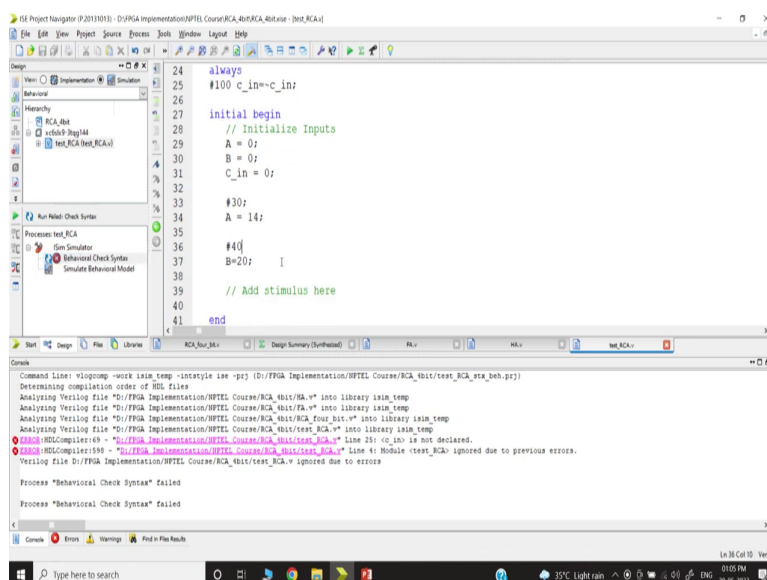
And, these are the standard you know you can initialize A, B etcetera. Now, I want to create; that means, I want to change the carry-in signal, I want to create like a clock.

(Refer Slide Time: 20:23)



I want to define a clock for the carry-in; that means, I will make an always block for the carry-in. So, my carry-in will be; that means after let us say 100 nanoseconds, 100-time unit, I want to toggle this c in, c it will be equal to c in. I want to toggle. And I want to change; that means, my data; that means, I also want to change; if you want to give more begin; that means, this is 1 block. Then, another block I want to create, 20 nanosecond duration I want to check A data; that means, my A; that means, ok. So, here always fine.

(Refer Slide Time: 21:17)

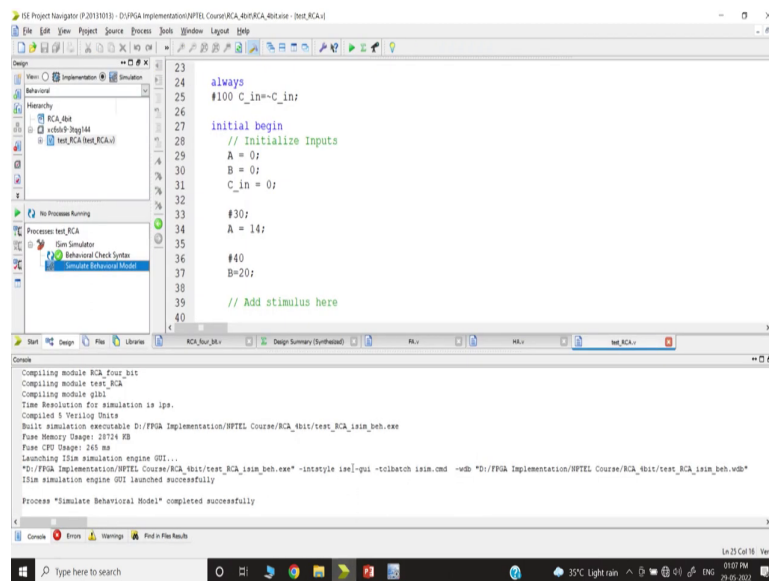


I want to initialize. So, initialize A B. Now, what I want to make, after let us say 30 nanoseconds, 30-time unit, 30-time unit I want to change the data; that means, I want to make

A equal to maybe I will take because by default if you do not specify, it will take decimal. It is a 4-bit data. So, you can use let us say you know 14 and B to be 10, then after ok.

So, I may not change even B, after another let us say 40 nanoseconds, I want to change B to be 20, and so on. So, let us check whether we can do it. So, if the test fixture; is so, we have to first check the behavioral model, Behavioral Syntax. So, I think c is not declared, c in [FL] we have changed the notation.

(Refer Slide Time: 22:36)

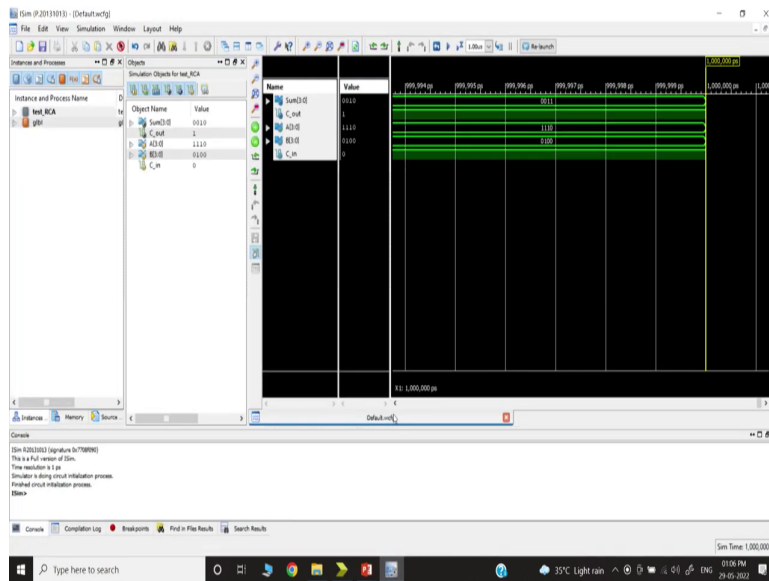


```
23
24 always
25 #100 C_in=~C_in;
26
27 initial begin
28 // Initialize Inputs
29 A = 0;
30 B = 0;
31 C_in = 0;
32
33 #30;
34 A = 14;
35
36 #40;
37 B=20;
38
39 // Add stimulus here
40
```

```
Compiling module RCA_4bit
Compiling module test_RCA
Compiling module q1b3
Time Resolution for simulation is 1ps.
Compiled 5 Verilog Units
Built simulation executable D:/FPGA Implementation/NPTEL Course/RCA_4bit/test_RCA_isim_beh.exe
Fuse Memory Usage: 29724 KB
Fuse CPU Usage: 245 ms
Launching ISim simulation engine GUI...
"D:/FPGA Implementation/NPTEL Course/RCA_4bit/test_RCA_isim_beh.exe" -inststyle ism[-gui -tclbatch isim.cmd -vdb "D:/FPGA Implementation/NPTEL Course/RCA_4bit/test_RCA_isim_beh.vdb"]
ISim simulation engine GUI launched successfully
Process "Simulate Behavioral Model" completed successfully
```

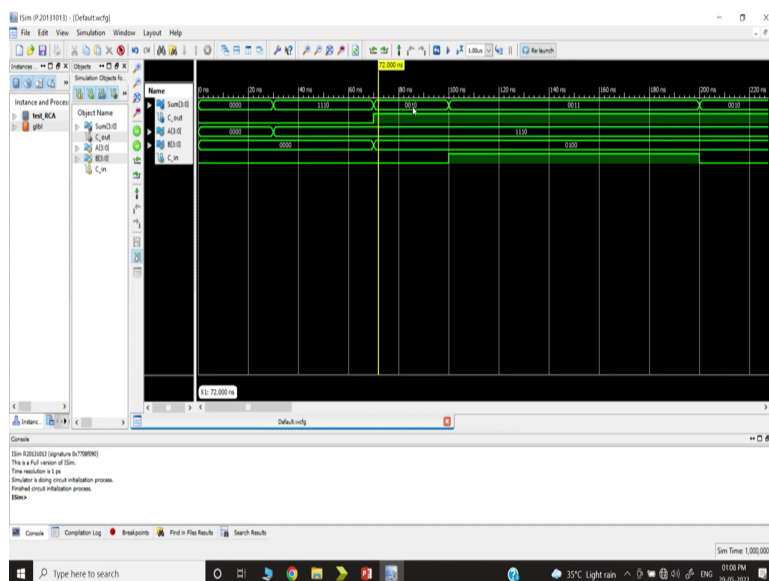
It should be capital C in sorry. So, you have to be very careful about the notation, yes now it is coming.

(Refer Slide Time: 22:48)



And, we are ready to go for level simulation.

(Refer Slide Time: 22:52)



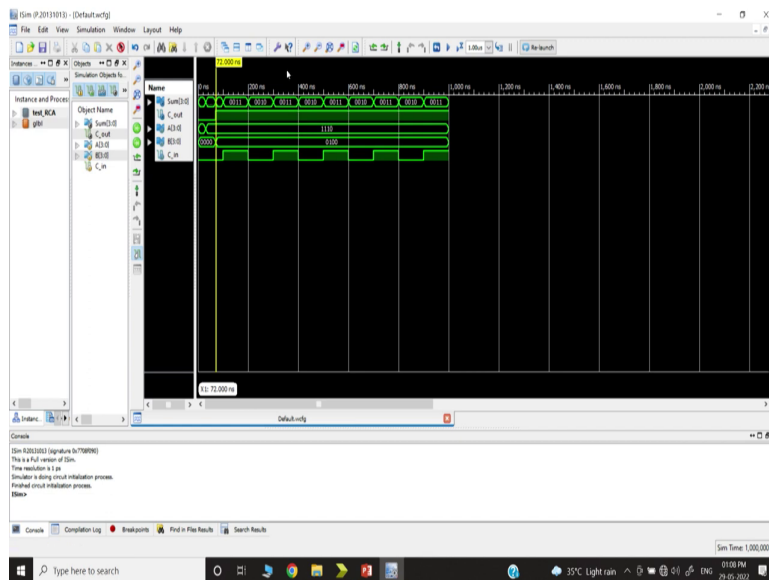
So, now, we are trying to check what happened here. So, let us say we are going to check the simulation. So, you can see our carry signal, I mean if I specify just a minute, I want to show. So, this is our carry and you can see I am changing the carry-in periodically ok, as a result, the sum must change ok. So, you can see initially I started with you know if I go all 0 0. The carry-in was 0.

So, Sum was 0, everything was 0. There was no carryout. Now, at this time I have loaded with some value; that means, I have changed my B to Sum 0 1 0 because 0 1 0 0. Because

what was the value I inserted? If you go 0 1 0; that means, I have loaded 14. So, if you take 14 so, you can get the corresponding binary number. So, it is showing. As a result, the Sum, what will be the Sum? The Sum that will be carried out is 1, sorry your carry-in is 0, B actually is changed to 0 1 0 0 and oh sorry A has changed here.

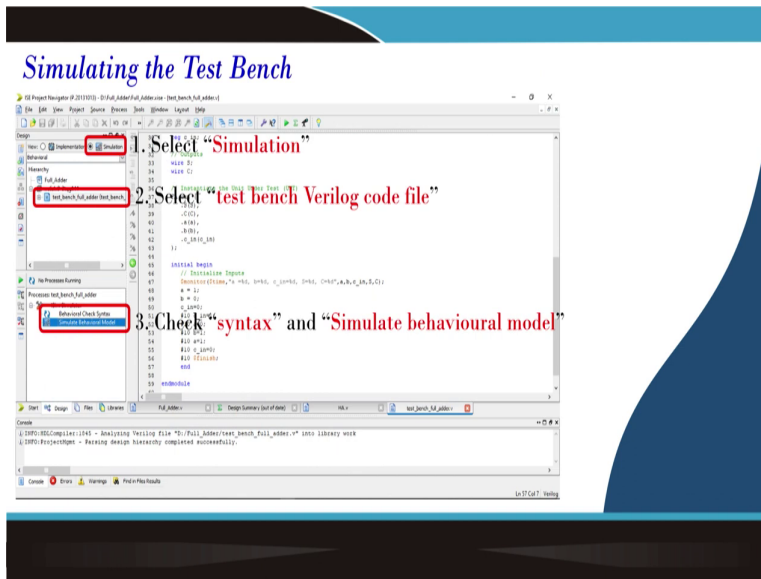
So, A has become 1 1 1 0 and since there is no carry in B is 0. So, Sum is the same as carrying that A. So, Sum is equal to A, because B is equal to 0, and carry in equal to 0. At this time, if you see here, we have now inserted carry-in, also we have changed the B. So, as a result, this value will be, what is A? 1 1 1 0 plus 0 1 0 0. So, the Sum will be 0 0 1 0 and there is a carryout.

(Refer Slide Time: 25:05)



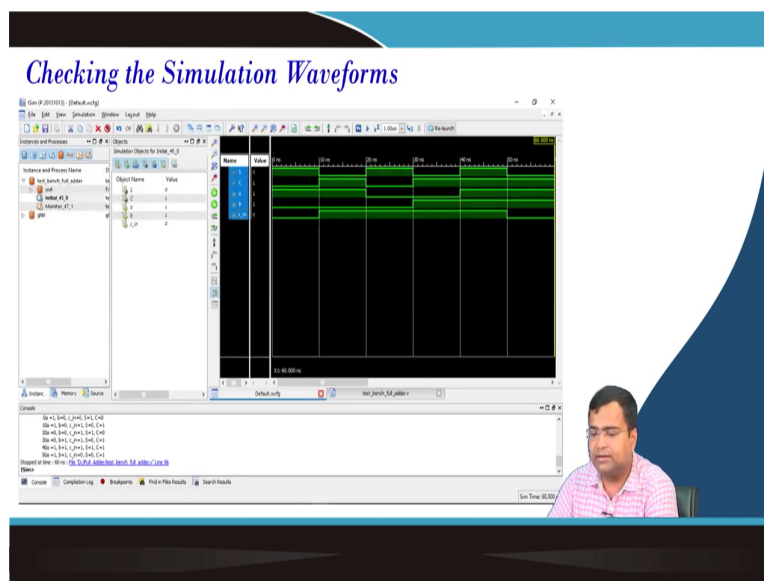
So, in that way you can check whether the logical logically whether it is correct or not. So, you can get the updated value. Now, since well are making 1 and 0, accordingly carry in and carry out. So, the sum will be computed; that means, we can successfully implement our 4-bit ripple carry adder and by that way; that means, we have learned that how to you know we have learned how to simulate.

(Refer Slide Time: 25:30)



And, we have how to simulate behavioral simulation, how check the timing diagram, and can customize the timing.

(Refer Slide Time: 30:38)



And, we can we have checked the simulation waveform, by that way you can check any logic. You can check your half adder circuit, you can check your full adder circuit, and you can check the full ripple carry adder.

(Refer Slide Time: 25:51)

Verilog HDL Simulation using Xilinx ISE Webpack

Let us start with Verilog HDL examples



So; that means, we have discussed; let us start we have already discussed that example of the 4-bit ripple carry adder.

(Refer Slide Time: 25:54)

Summary

- Steps for simulation using Xilinx ISE simulator
- Example case studies using Xilinx Webpack



IIT Kharagpur

In summary, we have discussed the step for simulation using the Xilinx ISE simulator. And, we have also discussed an example case study using Xilinx Webpack. I hope in the subsequent lecture when you go to Q format and all, we will show some more case studies.

And, again the interested participant can check the simulation in the Xilinx ISE tool. But, again it is not mandatory for the exam as well from an assignment point of view. But, it is important to know the Verilog coding, which will be part of the syllabus, that is for today.

Thank you very much.