Design of 4-bit full adder using nclaunch Simulator in Cadence software

Aim:

To write a verilog code for 4bit adder and verify the functionality using Test bench.

Tool Required:

Functional Simulation: nclaunch Simulator (nclaunch)

4-bit Adder Design:

To construct a 4-bit adder, need to chain together four 1-bit full adders. Each full adder computes the sum and carry for one bit of the two numbers. The carry-out from one adder feeds into the carry-in of the next adder in the sequence. This process adds the two 4-bit numbers bit by bit, with the carry propagating through each stage, resulting in a final sum and carry-out at the end.

To design a 1-bit full adder, the first step is to create a truth table that represents all possible combinations of the inputs (A, B, and CIN) and the corresponding outputs (Sum(S) and COUT).

Here's the Logic Ic for a 1-bit full adder:

Here's the truth table for a 1-bit full adder:

A	B	CIN	Sum(S)	COUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig 1: Diagram and truth table of full adder

Logic Expressions:

1. Sum (S):
   $S=A \oplus B \oplus CIN$
   Where $\oplus$ represents XOR.

2. Carry out (COUT):
   COUT=(A&B) | (CIN&(A^B))

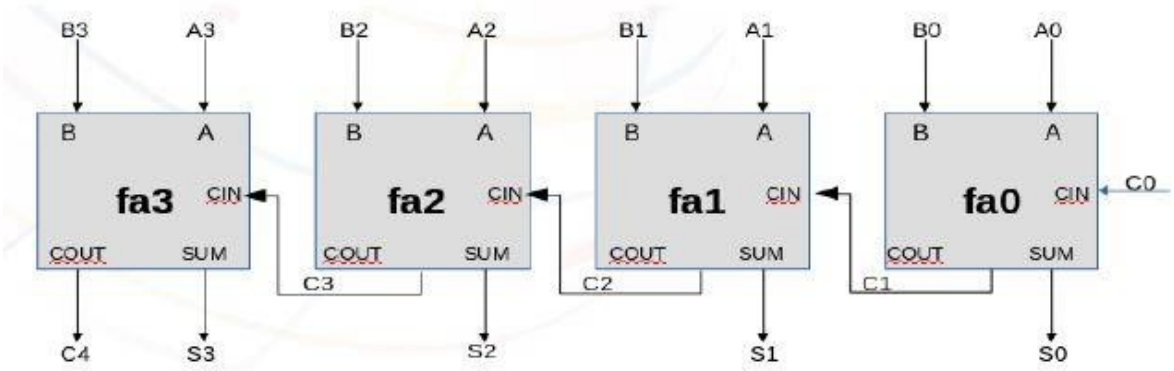

Fig 2: Diagram of 4 Bit Adder

Creating Source Codes

- In the Terminal, type gedit <filename>.v (ex: gedit 4bitadder.v).
- A Blank Document opens up into which the following source code can be typed down.

Note : File name should be with HDL Extension

a) Verify the Functionality

- Three Codes shall be written for implementation of 4-bit Adder as follows,
  - fa.v → Single Bit 3-Input Full Adder [Sub-Module / Function]
  - fa_4bit.v → Top Module for Adding 4-bit Inputs.
  - fa_4bit_test.v → Test bench

```
module full_adder(A,B,CIN,S,COUT);
input A,B,CIN;
output S,COUT;
assign S=A^B^CIN;
assign COUT=(A&B) | (CIN&(A^B));
endmodule

module fulladd_4bit(A,B,C0,S,C4);
input [3:0] A,B;
input C0;
output [3:0] S;
output C4;
wire C1,C2,C3;
full_adder fa0 (A[0],B[0],C0,S[0],C1);
full_adder fa1 (A[1],B[1],C1,S[1],C2);
full_adder fa2 (A[2],B[2],C2,S[2],C3);
full_adder fa3 (A[3],B[3],C3,S[3],C4);
endmodule

module test_4bit;
reg [3:0] A;
reg [3:0] B; reg C0;
wire [3:0] S; wire C4;
```

```
module test_4bit;
reg [3:0] A;
reg [3:0] B; reg C0;
wire [3:0] S; wire C4;
fulladd_4bit dut (A,B,C0,S,C4);
initial
begin
A=4'b0011;B=4'b0011;C0=1'b0;
#10;  A=4'b1011;B=4'b0111;C0=1'b1;
#10; A=4'b1111;B=4'b1111;C0=1'b1;
#10;
end initial
#50 $finish;
Endmodule
```

Fig 3: Verilog code for 4 Bit Adder

Functional Simulation:

- Invoke the cadence environment by type the below commands
  - **tcsh** (Invokes C-Shell)
  - **source /cadence/install/cshrc** (mention the path of the tools)

(The path of cshrc could vary depending on the installation destination)

- After this you can see the window like below

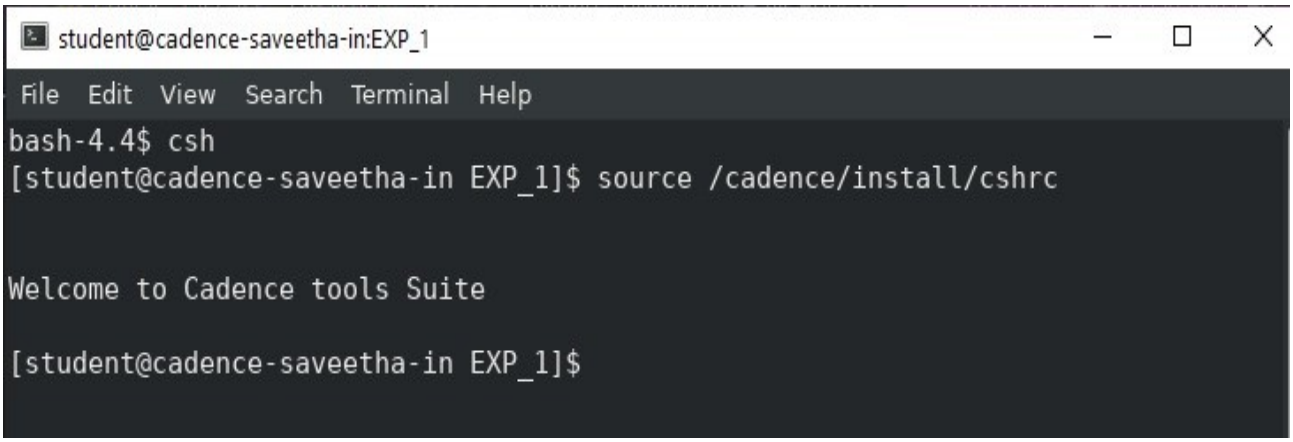

Fig 4: Invoke the Cadence Environment

- To Launch Simulation tool
  - **linux:/> nclaunch -new**& // **"-new"** option is used for invoking NCVERILOG for the first time for any design

or

  - **linux:/> nclaunch**& // On subsequent calls to NCVERILOG

- It will invoke the nclaunch window for functional simulation we can compile,elaborate and simulate it using Multiple Step .

Fig 5 : Invoke the nclaunch

- Select Multiple Step and then select “Create cds.lib File” as shown in below figure
- Click the cds.lib file and save the file by clicking on Save option

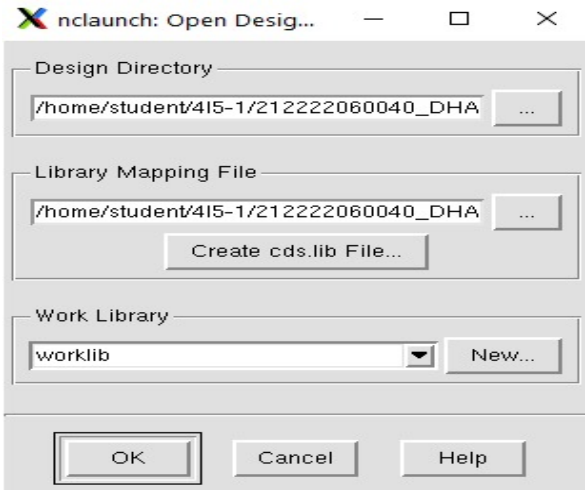

Fig 6: cds.lib file Creation

- Save cds.lib file and select the correct option for cds.lib file format based on the HDL Language and Libraries used.
- Select “Don’t include any libraries (verilog design)” from “New cds.lib file” and click on “OK” as in below figure .

  - We are simulating verilog design without using any libraries
  - A Click “OK” in the “nclaunch: Open Design Directory” window as shown in below figure

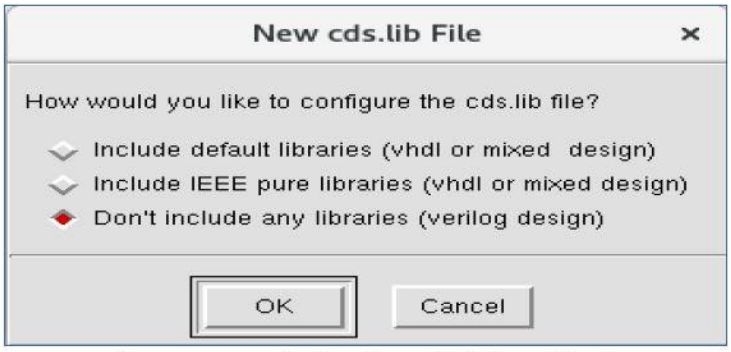

Fig 7: Selection of Don’t include any libraries

- A ‘NCLaunch window’ appears as shown in figure below
- Left side you can see the HDL files. Right side of the window has worklib and snapshots directories listed.
- Worklib is the directory where all the compiled codes are stored while Snapshot will have output of elaboration which in turn goes for simulation .
- To perform the function simulation, the following three steps are involved Compilation, Elaboration and Simulation.

Step 1: Compilation:– Process to check the correct Verilog language syntax and usage

- **Inputs:** Supplied are Verilog design and test bench codes
- **Outputs**: Compiled database created in mapped library if successful, generates report else error reported in log file
- **Steps for compilation:**
  1. Create work/library directory (most of the latest simulation tools creates automatically)
  2. Map the work to library created (most of the latest simulation tools creates automatically)
  3. Run the compile command with compile options

  i.e Cadence IES command for compile: ncverilog +access+rwc -compile fa.v

  – Left side select the file and in Tools : launch verilog compiler with current selection will get enable. Click it to compile the code
  – Worklib is the directory where all the compiled codes are stored while Snapshot will have output of elaboration which in turn goes for simulation

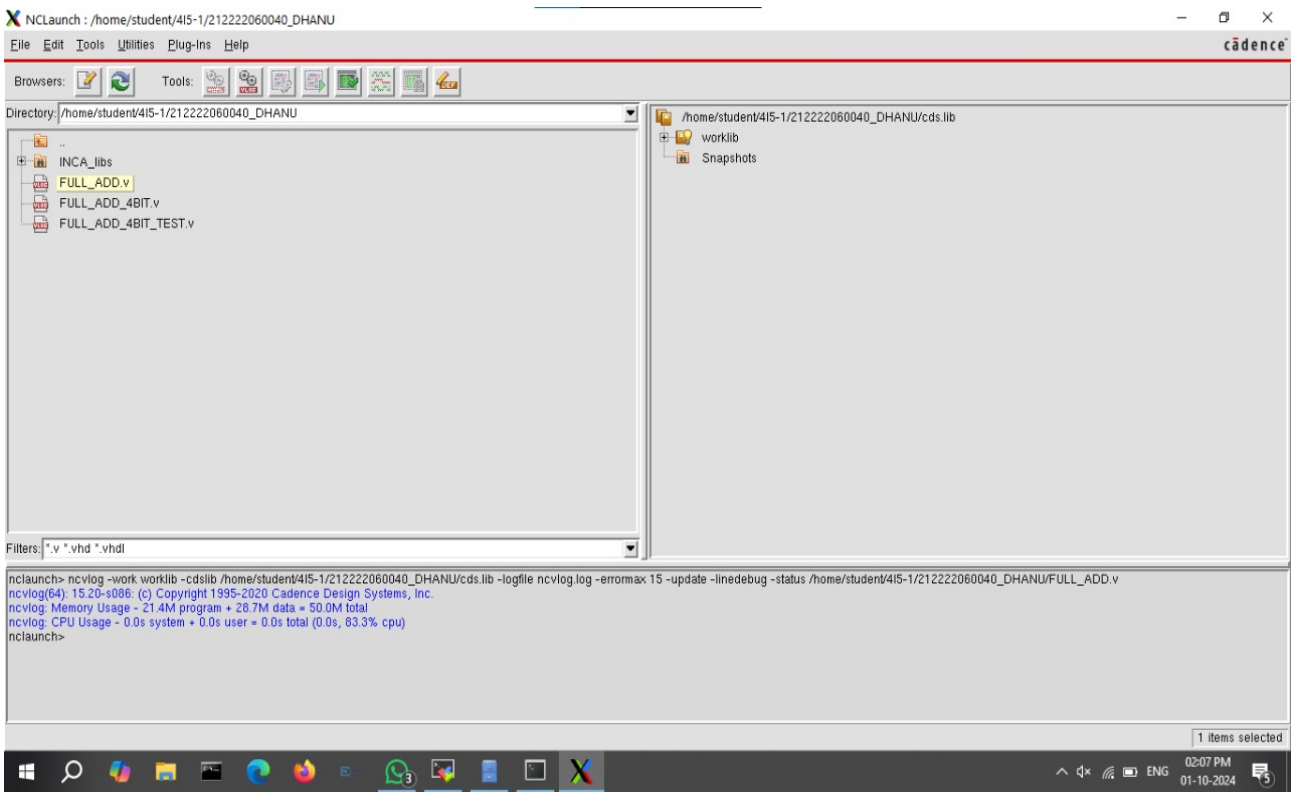

Fig 8(a): Compilation Launch Option

- After compilation it will come under worklib you can see in right side window
- Select the test bench and compile it. It will come under worklib. Under Worklib you can see the module and test-bench.

- The cds.lib file is an ASCII text file. It defines which libraries are accessible and where they are located. It contains statements that map logical library names to their physical directory paths. For this Design, you will define a library called “worklib”

Step 2: Elaboration:– To check the port connections in hierarchical design

- **Inputs:** Top level design / test bench Verilog codes
- **Outputs:** Elaborate database updated in mapped library if successful, generates report else error reported in log file
- **Steps for elaboration** – Run the elaboration command with elaborate options
  1. It builds the module hierarchy
  2. Binds modules to module instances
  3. Computes parameter values
  4. Checks for hierarchical names conflicts
  5. It also establishes net connectivity and prepares all of this for simulation

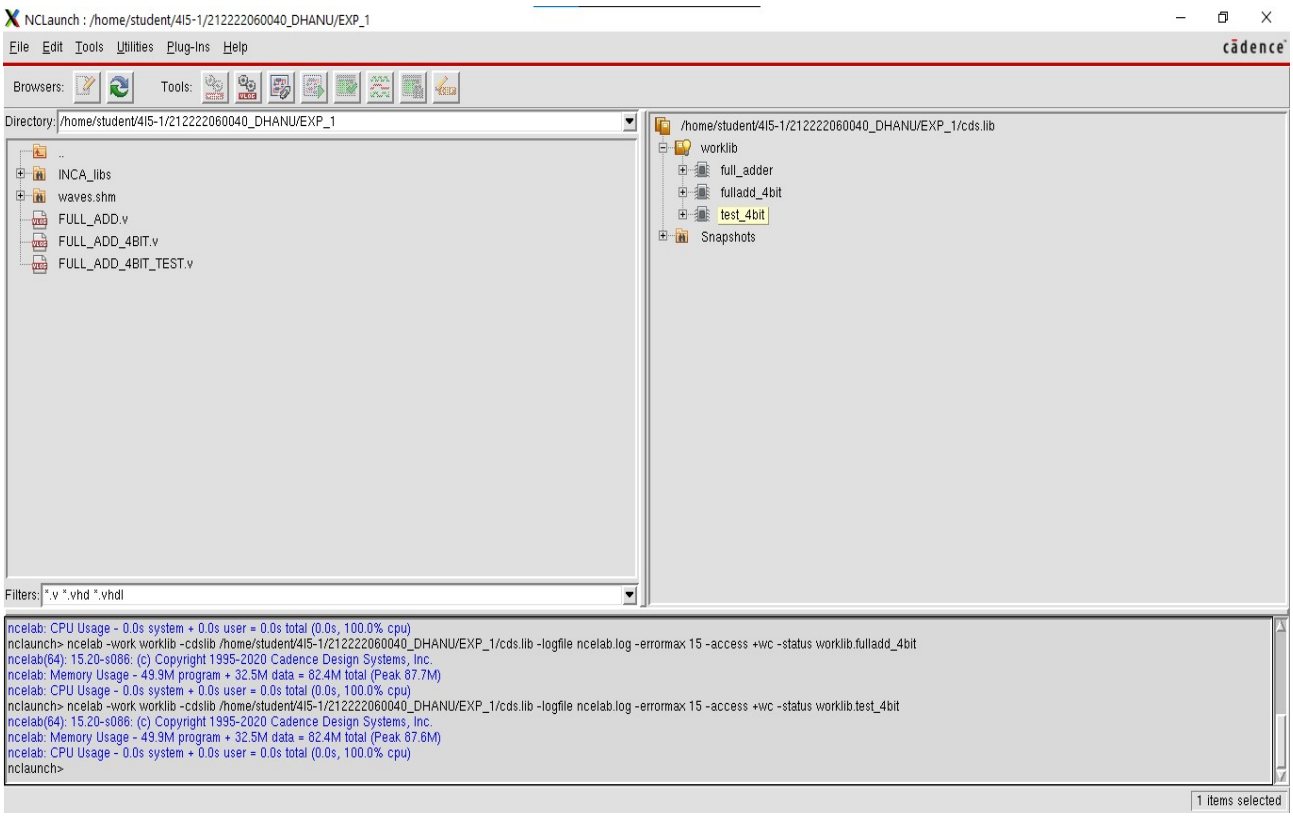

Fig 8(b): Compilation Launch Option

- After elaboration the file will come under snapshot. Select the test bench and elaborate it.

Step 3: Simulation:– Simulate with the given test vectors over a period of time to observe the output behaviour.

- Inputs: Compiled and Elaborated top level module name
- Outputs: Simulation log file, waveforms for debugging
- Simulation allow to dump design and test bench signals into a waveform
- Steps for simulation – Run the simulation command with simulator options

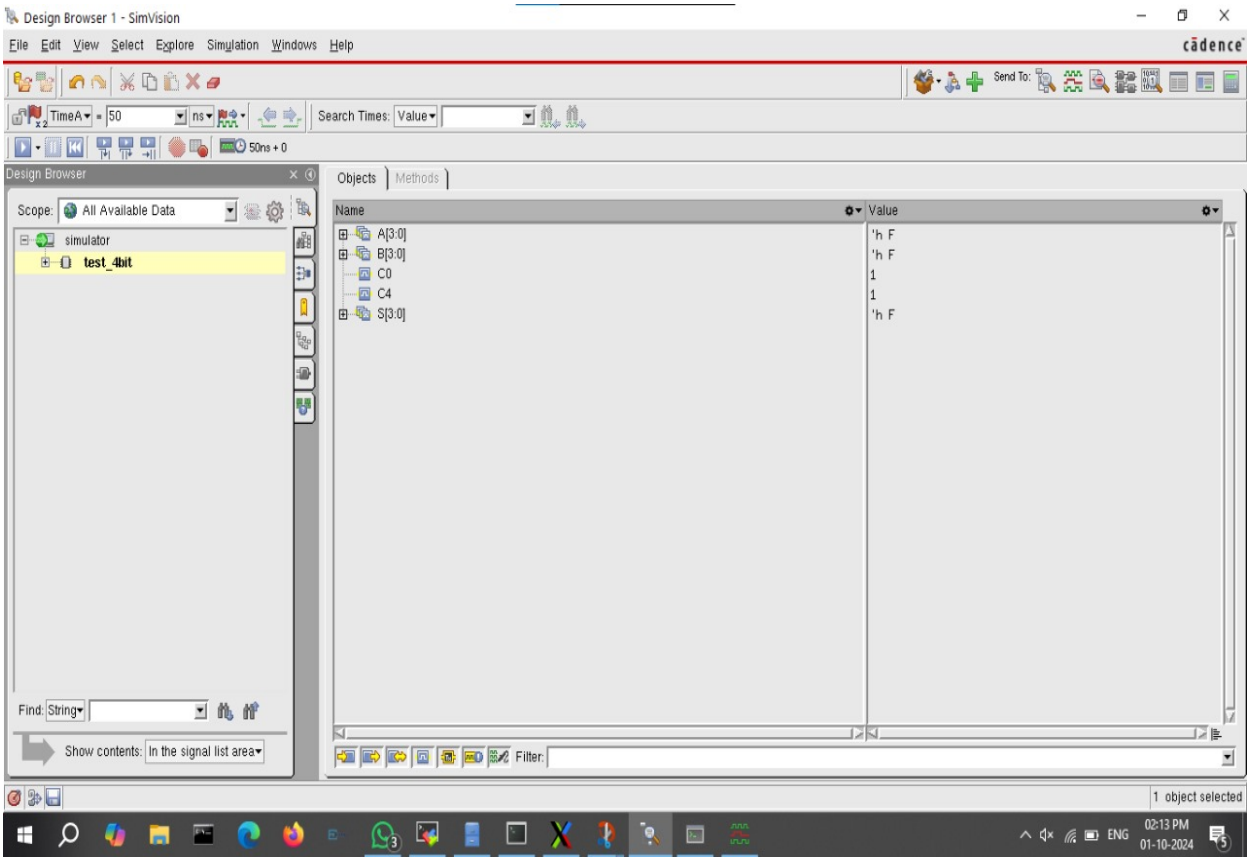

Fig 9: Design Browser window for simulation

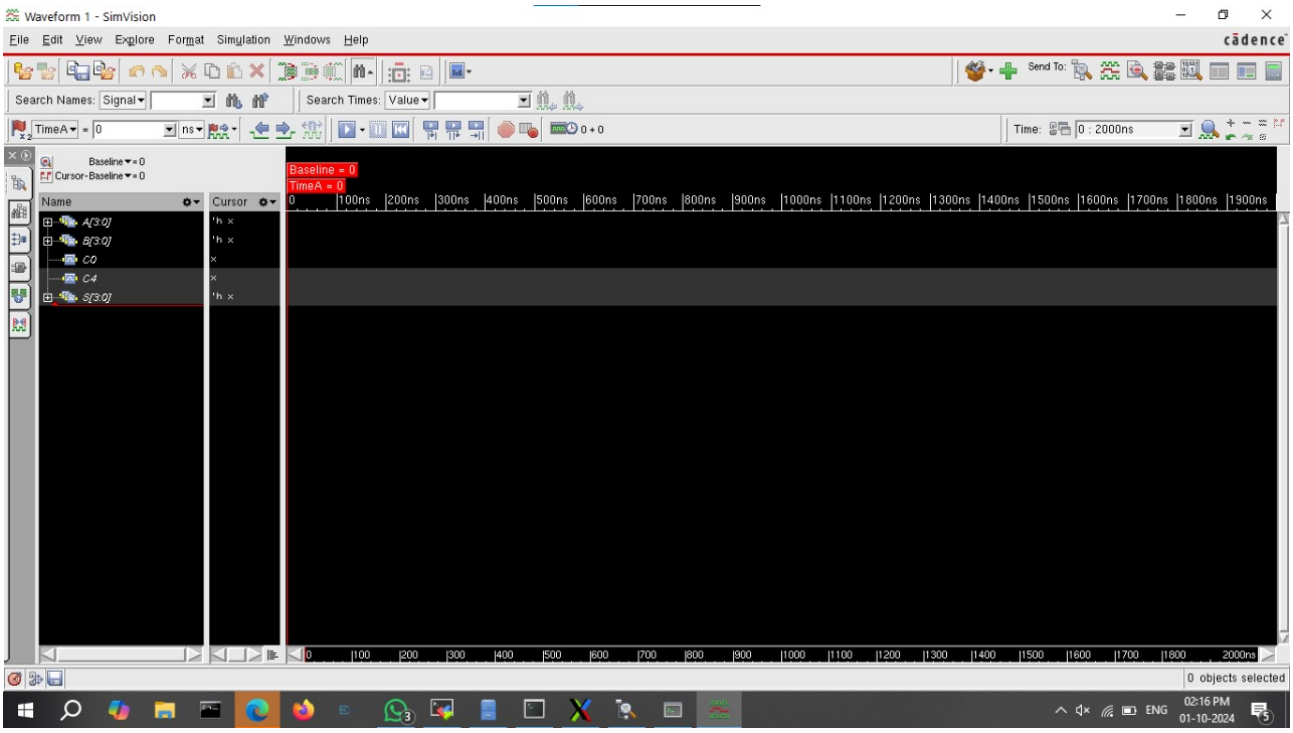

Fig 10: Simulation Waveform Window

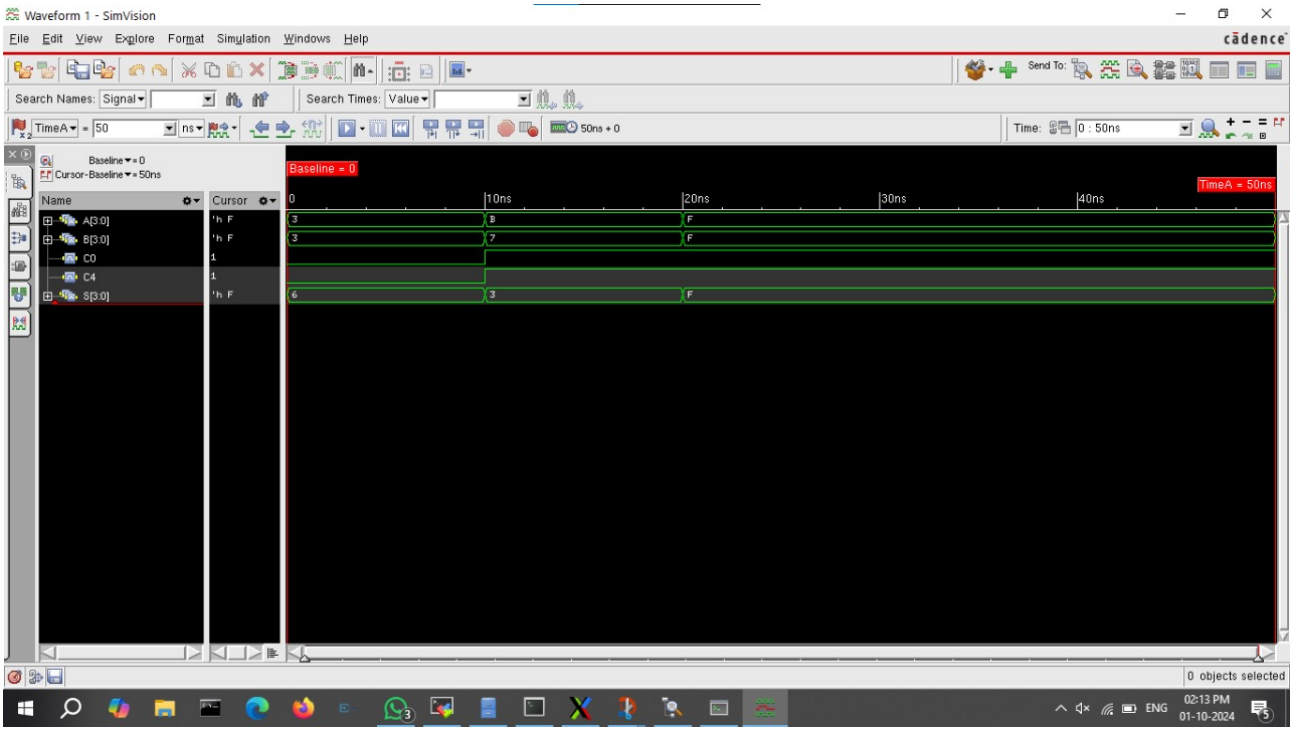

Fig 11: Simulation Waveform Window

Github link : https://github.com/dhanu0503/4-Bit-Adder-nclaunch-simulation