CprE 381, Computer Organization and Assembly Level Programming

Lab 1 Report

Student Name : Rafat Momin

Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the lab document for the context of the following questions.

[Part 1.c] Think of three more cases and record them in your lab report.

[Part 1.e] For labels 1, 7, 22, and 28, specify where (VHDL file and line number) these values are located – some will be found in more than one place. Also attempt to explain the functionality of each label as it occurs in the code

[Part 1.g.v] In your lab report, include a screenshot of the waveform. Describe, in plain English, any differences between what you expected and what the simulation showed.

[Part 1.h] In your lab report, include a screenshot of the waveform. Describe, in plain English, how your waveform matches the expected result (e.g., reference the specific cycles and times). In your submission zip file, provide the completed *TPU_MV_Element.vhd* file in a folder called 'MAC'.

[Part 3.a] Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 2:1 mux. Include this in your lab report.

[Part 3.d] In your lab report, include a screenshot of the waveform. Make sure to label the screenshot with which module it is testing.

[Part 3.e] Again, in your lab report, include a labeled screenshot of the waveform showing the dataflow mux implementation working.

[Part 4] Include a waveform screenshot and corresponding description demonstrating it is working correctly.

[Part 5.b] Include a waveform screenshot and description in your lab report.

[Part 6.a] A full adder takes three single-bit inputs and produces two single-bit outputs – a sum and carry for the addition of the three input bits. Draw the truth table, Boolean equation, and Boolean circuit equivalent (using only two-input gates) that implements a 1-bit full adder. Include this in your report.

[Part 6.c] Then draw a schematic of the intended design, including inputs and outputs and at least the 0, 1, N-2, and N-1 stages. Include this in your report.

[Part 6.d] Include an annotated waveform screenshot in your write-up.

[Part 7.a] Draw a schematic (don't use a schematic capture tool) showing how an N-bit adder/subtractor with control can be implemented using only the three main components designed in earlier parts of this lab (i.e., the N-bit inverter, N-bit 2:1 mux, and N-bit adder). How is the 'nAdd_Sub' bit used? Include this in your report.

[Part 7.c] Provide multiple waveform screenshots in your write-up to confirm that this component is working correctly. What test-cases did you include and why?

```
-- Test case 3:
    -- Perform average example of an input activation of 3 and a partial sum of 25. The weight is still 10.
    s_iX   <= x"04";  -- 3 in hexadecimal
    s_iW   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
    s_iLdW <= '0';    -- Make sure we don't continue to load.
    s_iY   <= x"19";  -- 25 in hexadecimal
    wait for gCLK_HPER*2;
    wait for gCLK_HPER*2;
-- Test case 4:
    -- Perform average example of an input activation of 3 and a partial sum of 25. The weight is still 10.
    s_iX   <= x"05";  -- 3 in hexadecimal
    s_iW   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
    s_iLdW <= '0';    -- Make sure we don't continue to load.
    s_iY   <= x"19";  -- 25 in hexadecimal
    wait for gCLK_HPER*2;
    wait for gCLK_HPER*2;
-- Test case 5:
    -- Perform average example of an input activation of 3 and a partial sum of 25. The weight is still 10.
    s_iX   <= x"06";  -- 3 in hexadecimal
    s_iW   <= x"00";  -- Not strictly necessary, but this makes the testcases easier to read
    s_iLdW <= '0';    -- Make sure we don't continue to load.
    s_iY   <= x"19";  -- 25 in hexadecimal
    wait for gCLK_HPER*2;
    wait for gCLK_HPER*2;

    wait;
  end process;
```

Part 1c

I added 3 more test cases in my testbench file.

1e
Label 1:
File: TPU_MV_Element.vhd

The clock signal `iCLK` is first declared in the entity port declaration.

- **Functionality: T**his label refers to the clock input for the processing element, which is critical for controlling the timing of all synchronous operations within the module.

Label 7 :

- **File:** Adder.vhd
  oC: `oC` is declared on line 16 in the adder entity where it is defined as an output port.
- **Functionality**: Label 7 likely refers to the actual addition operation that produces the output `oC`

Label 22 :

- **File:** TPU_MV_Element

  The s_w is first instantitated within the vhdl file in line 77.

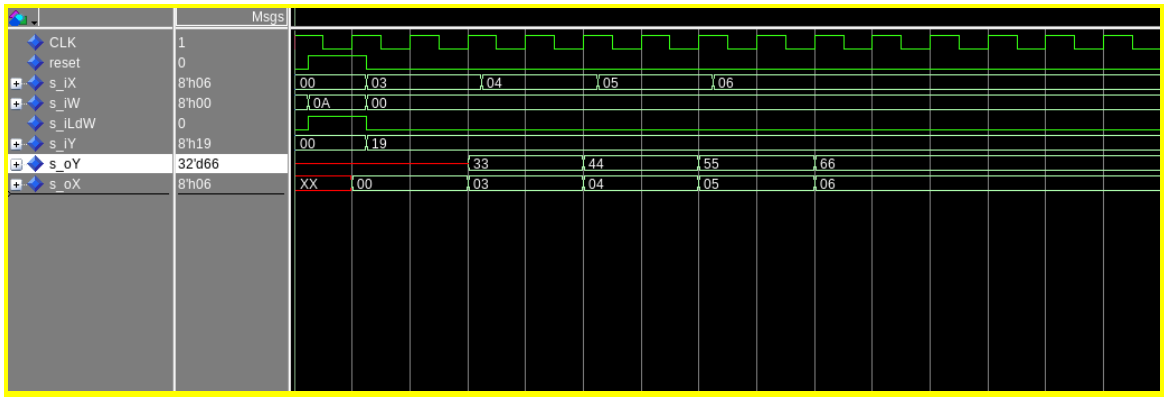
- **Functionality:** The signal `s_W` would be used within the module to store weight values, potentially coming from a weight register (`g_Weight`) that latches weights based on a load control signal.

Label 28 :

- **File:** `TPU_MV_Element.vhd`

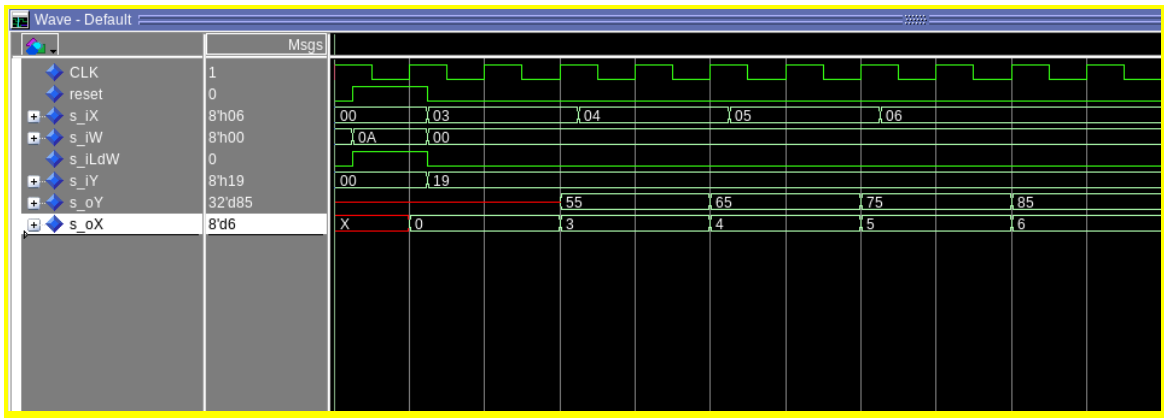
  Near the outputs of complex operations or at the end of processing chains.

Functionality: Represents the final or intermediate output from a processing stage, potentially acting as a stored result or a state value passed on for additional computation or output.

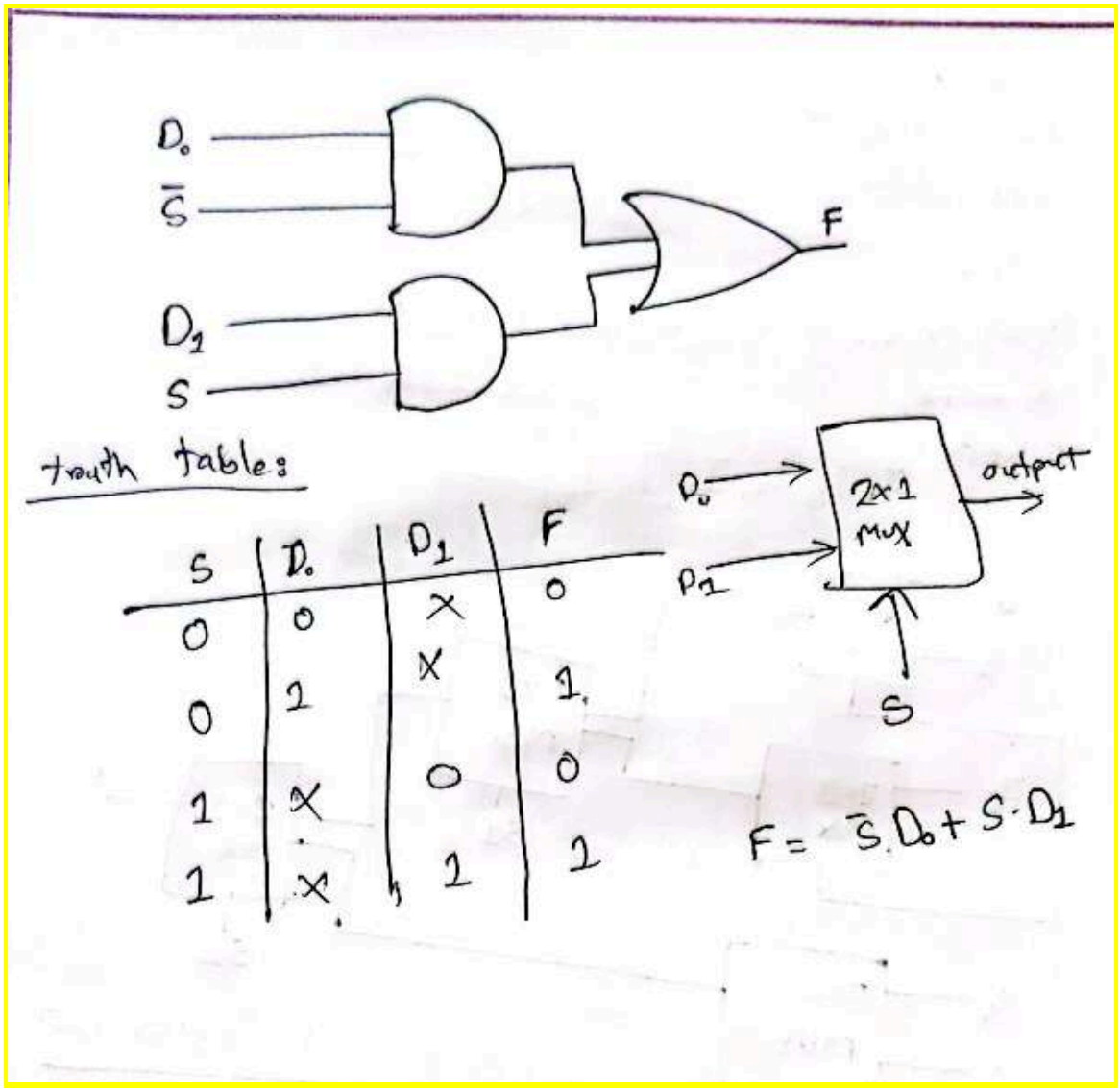


Part 1.g.v

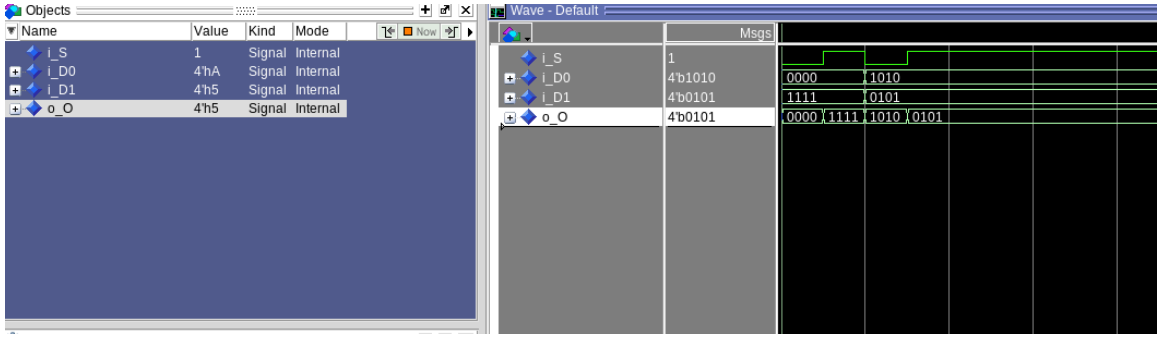
I added three more test cases(3,4,5). I changed the value of s_iX to 4,5,6 initially. Afterward, I got these values 65,75,85. Because the multiplier weight value is 10, 4,5,6 multiplies with 10, and the adder adds the value 25. While starting the simulation, I got an unexpected value instead of 55. Because inside the TPU_MV_Element.vhd file it was not initialized to the right value. I changed that and then got my expected results.



Part 1.h



Part 3A



Part 3d

Test Case 1: Select `i_D0`

- Initial Condition: `i_S = '0'` (select signal for `i_D0`), `i_D0 = "0000"`, `i_D1 = "1111"`.
- Expectation: When `i_S` is '0', the multiplexer should select `i_D0`. Thus, `o_O` should be equal to `i_D0` which is "0000".
- Waveform Annotation: Highlight the portion of the waveform where `i_S` changes to '0' and verify that `o_O` changes to "0000" simultaneously. Annotate this change to illustrate that the multiplexer is correctly selecting `i_D0`.

Test Case 2: Select `i_D1`

- Condition: After 10 ns, `i_S` changes to '1' (select signal for `i_D1`).
- Expectation: With `i_S` now '1', `o_O` should switch to "1111", matching the value of `i_D1`.
- Waveform Annotation: Mark the timeline where `i_S` switches to '1'. Show that `o_O` updates to "1111" at this point, confirming the mux's response to the select signal.

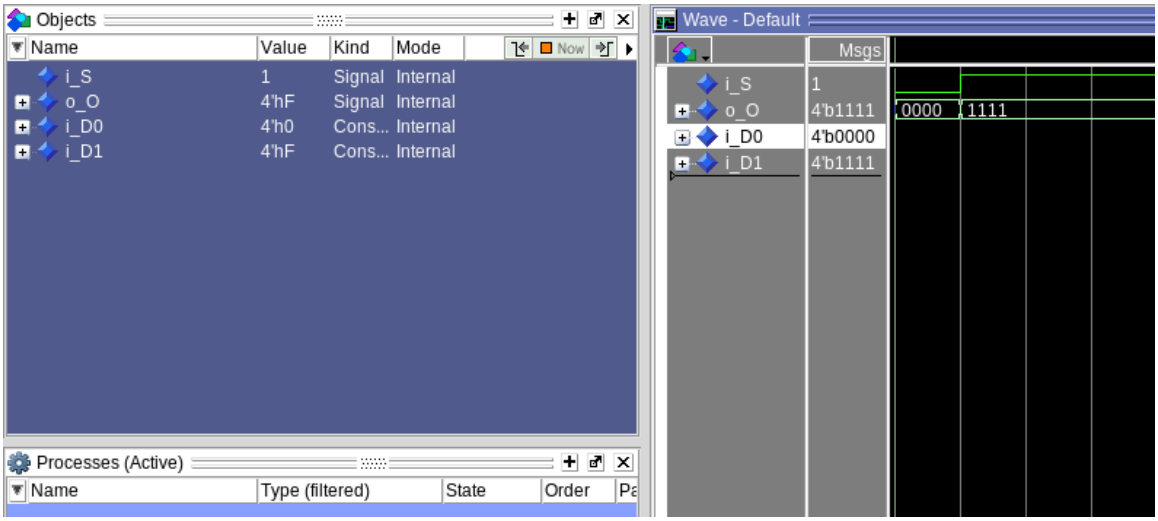
Test Case 3: Change Inputs While Selector is '0'

- Condition: Inputs `i_D0` and `i_D1` change to "1010" and "0101", respectively, while `i_S` remains '0'.
- Expectation: Since `i_S` is still '0', `o_O` should reflect the new value of `i_D0`, which is "1010".
- Waveform Annotation: Illustrate the change in `i_D0` and `i_D1` and ensure `o_O` follows `i_D0`'s change to "1010". This demonstrates that the mux continues to hold the correct selection despite changes in inputs.

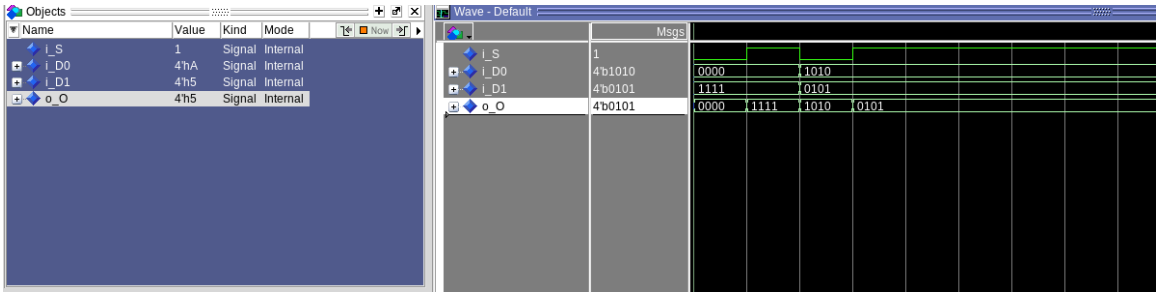
Test Case 4: Toggle Selector After Changing Inputs

- Condition: `i_S` toggles to '1' after the input changes.
- Expectation: As `i_S` switches to '1', `o_O` should now match `i_D1`'s value, "0101".
- Waveform Annotation: Display the point where `i_S` switches back to '1'. Confirm that `o_O` changes to "0101", indicating the mux is dynamically responding to both selector and input changes.

Part3e



Part3e



Test Case 1: Select `i_D0`

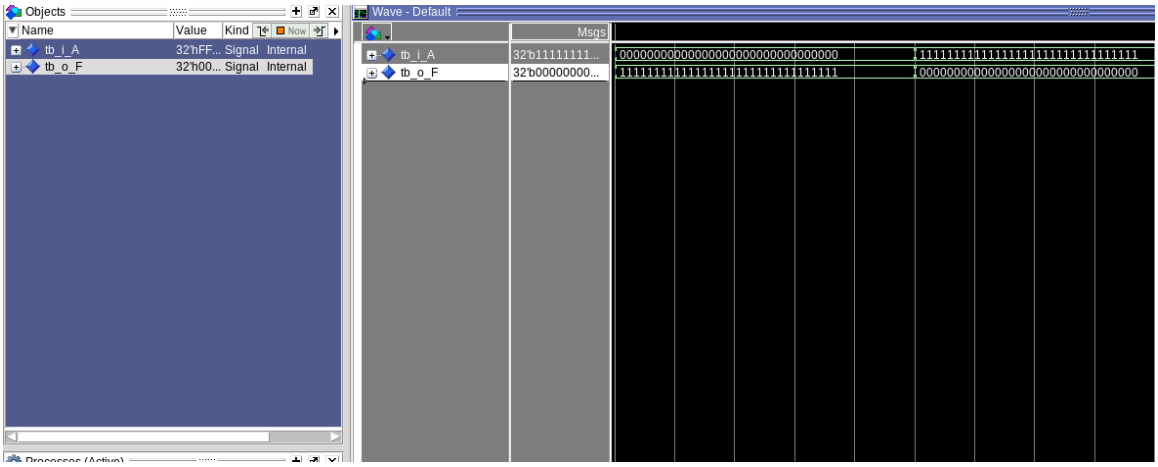
- Scenario Setup:
  - Selector Signal (`i_S`): Initialized to '0' to select the first input, `i_D0`.
  - Input Values: `i_D0 = "0000"` and `i_D1 = "1111"`.
- Expected Output: When `i_S` is '0', the output `o_O` should match the value of `i_D0`, which is "0000".
- Waveform Annotation: Highlight the segment where `i_S` is '0'. Annotate the waveform to show that `o_O` correctly reflects "0000", confirming that the multiplexer is selecting the first input (`i_D0`). The annotation could read: "Selector is 0, output matches i_D0: 0000."

Test Case 2: Select `i_D1`

- Scenario Setup:
    - Selector Change: After 10 ns, `i_S` changes to '1' to select the second input, `i_D1`.
- Expected Output: With `i_S` set to '1', `o_O` should switch to match `i_D1`, which is "1111".
- Waveform Annotation: Mark the instance when `i_S` transitions to '1'. Note on the waveform that `o_O` updates to "1111" at this point. This annotation helps illustrate the multiplexer's dynamic response to the select signal changing. You might note: "Selector changes to 1, output updates to match i_D1: 1111."

Part4

The waveform screenshot provided illustrates the correct functionality of the N-bit 2:1 multiplexer, designed using a structural VHDL approach with a generate for statement. The testbench effectively validates the multiplexer's ability to select between two N-bit input vectors (`i_D0` and `i_D1`) based on the value of the selection signal (`i_S`).



Part5b

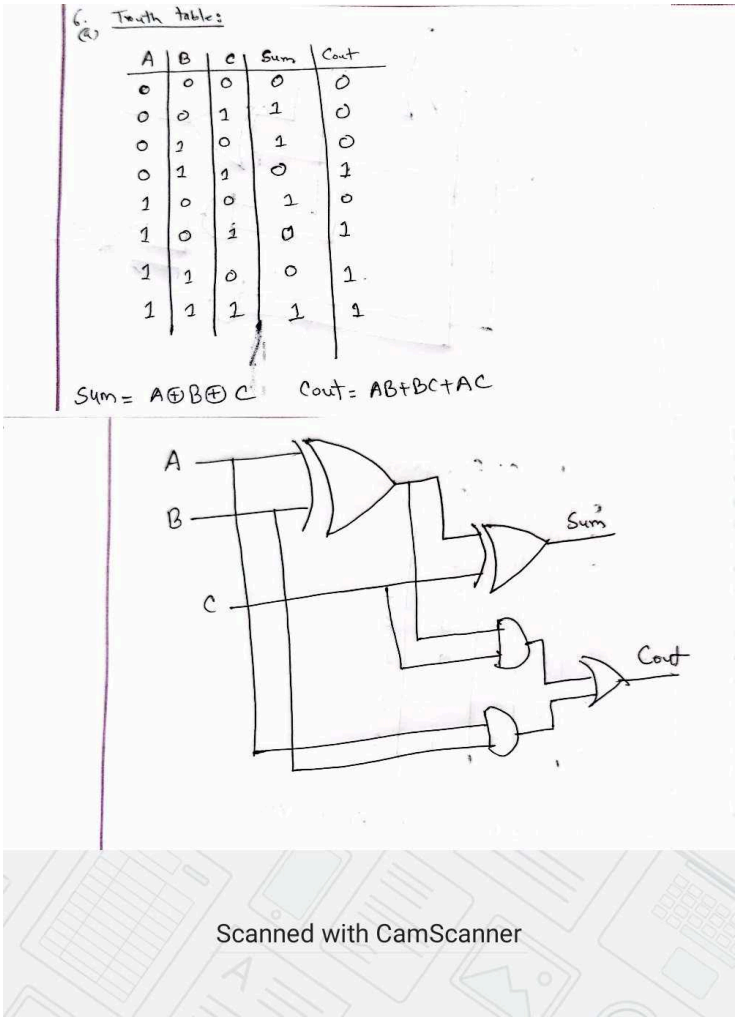
Test Case 1: All Zeros Input

- **Scenario Setup**:
    - **Input (`i_A`)**: Initialized to all zeros ("`00000000000000000000000000000000`").
- **Expected Output**: When `i_A` is all zeros, the output `o_F` should be all ones ("`11111111111111111111111111111111`").
- **Waveform Annotation**: Highlight the portion of the waveform where `i_A` is all zeros. Note on the waveform that `o_F` correctly reflects all ones, confirming that the ones complementor is functioning as expected. The annotation might read: "Input is all zeros, output correctly shows all ones as per ones complement operation."

Test Case 2: All Ones Input

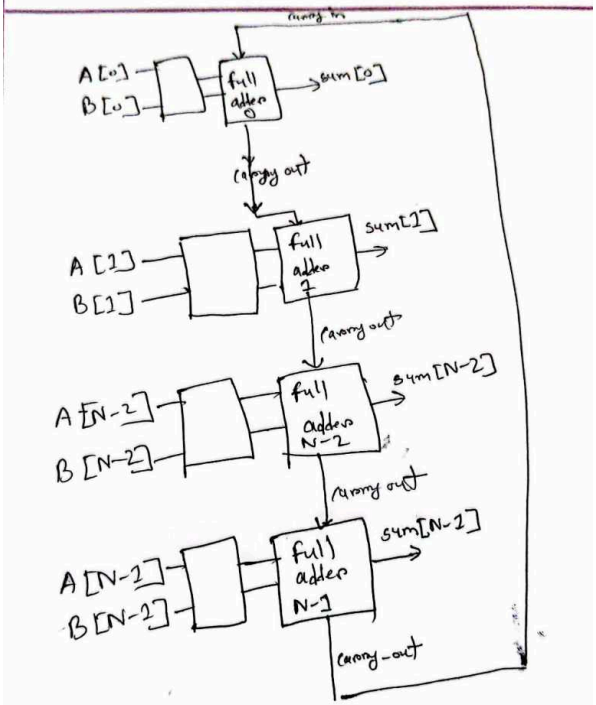
- **Scenario Setup**:

    - **Input (`i_A`)**: Changed to all ones (`"11111111111111111111111111111111"`).
- **Expected Output**: With `i_A` set to all ones, `o_F` should switch to all zeros (`"00000000000000000000000000000000"`).
- **Waveform Annotation**: Mark the instance when `i_A` transitions to all ones. Note on the waveform that `o_F` updates to all zeros at this moment. This annotation helps illustrate the ones complementor's response to the input change. You could note: "Input changes to all ones, output correctly flips to all zeros as expected."

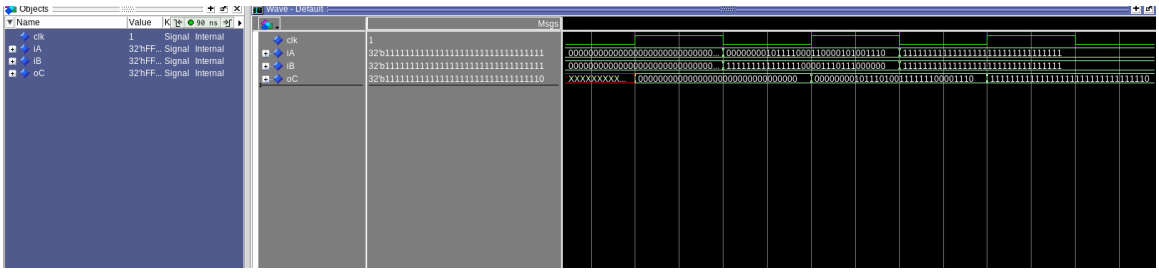


Part 6a

(c) stage 0 (0th bit)
stage 1 (1st bit)
stage 2 N-2
stage N-1 (Last Bit)



Part 6c



Part6d
Initial State (0ns to 20ns):

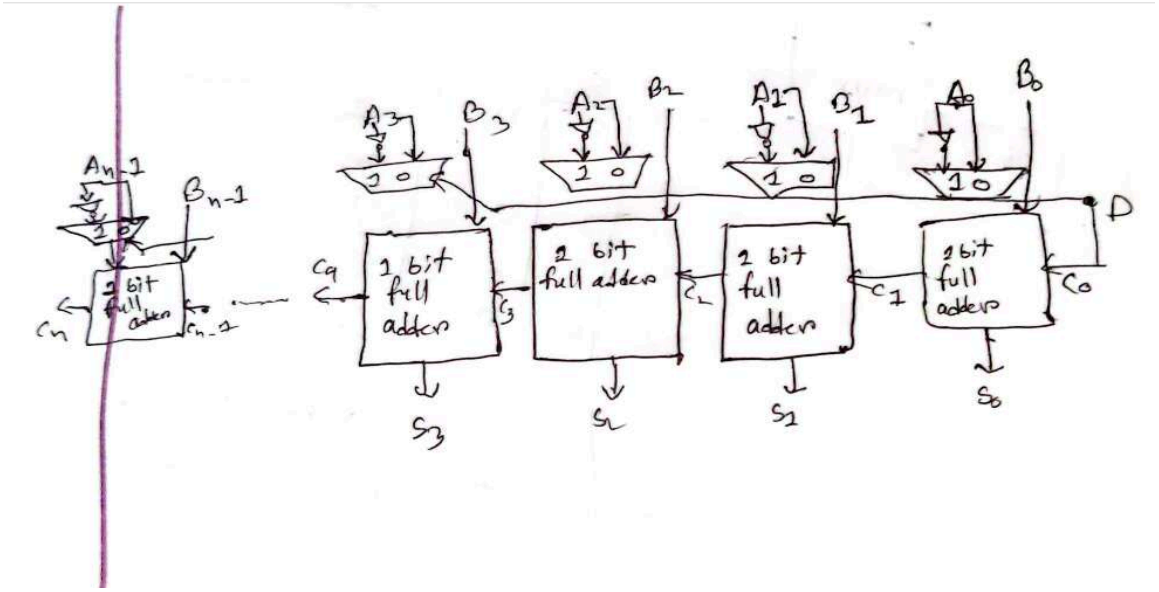
- Inputs: `iA` and `iB` are both set to zero (`00000000000000000000000000000000`).
- Output: `oC` should also be zero (`00000000000000000000000000000000`), reflecting the correct sum of the inputs.
- Clock: Observes a toggle from '0' to '1' every 10ns, simulating a typical clock signal.

First Input Change (20ns to 40ns):

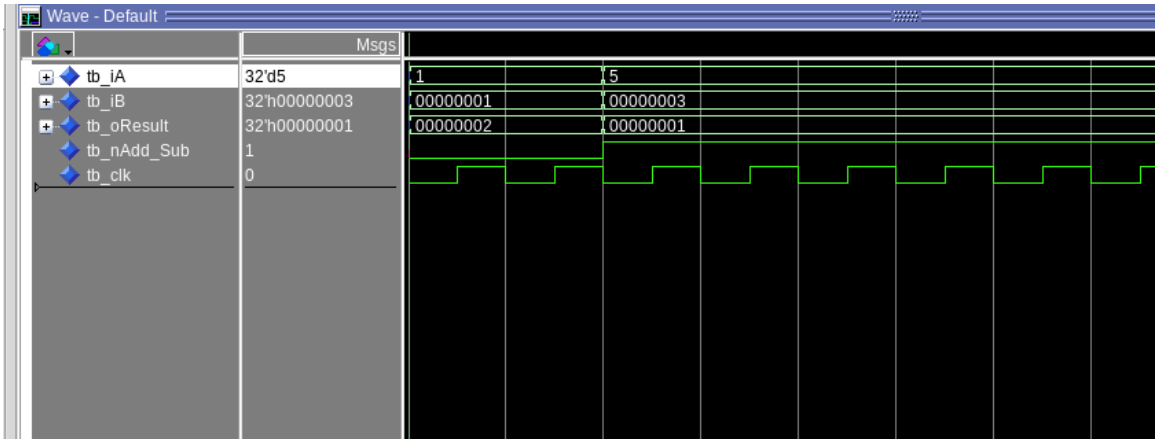
- Inputs:
  - `iA` is set to `00000000000010111101110011010010` (binary for 12345678).
  - `iB` is set to `11111111111111111111000001001000` (binary for -123456 using two's complement).
- Output: `oC` should display the sum of these two numbers in binary, showcasing the adder's capability to handle both positive and negative values.

Overflow Condition (40ns onwards):

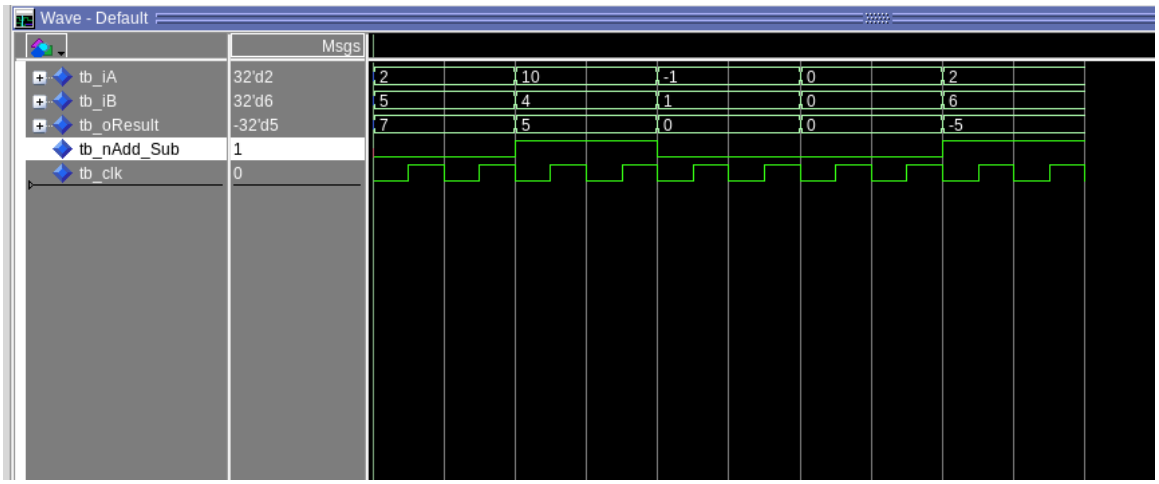
- Inputs: Both `iA` and `iB` are set to maximum negative values (`11111111111111111111111111111111`), intending to test the overflow condition.
- Output: `oC` will reflect the overflow condition, potentially showing either the maximum or minimum possible value, depending on how overflow is managed within the Adder design.



Part7a



Part7c



Part7c(in decimals)

Waveform Annotation for Testbench 1 (Binary Format)

In this waveform from the first testbench (`adder_sub_tb.vhd`), we observe the following sequence:

1. **Initial State:**
   - `iA` and `iB` are both initialized to binary `00000000000000000000000000000001`, which corresponds to the decimal value 1.
   - The `nAdd_Sub` signal is set to '0', indicating an addition operation.
   - The operation results in `oResult` being `00000000000000000000000000000010`, binary for 2, which is the correct output for the addition of 1 and 1.
2. **Subsequent State:**
   - The inputs `iA` and `iB` transition to `00000000000000000000000000000101` (binary for 5) and `00000000000000000000000000000011` (binary for 3) respectively.
   - The `nAdd_Sub` signal switches to '1', triggering a subtraction operation.
   - The resulting `oResult` correctly displays `00000000000000000000000000000010` (binary for 2), validating the subtraction of 3 from 5.

Waveform Annotation for Testbench 2 (Decimal Format)

The waveform from the second testbench (`adder2_sub2_tb2.vhd`) shows these key sequences:

1. **Test 1: Addition (0ns to 20ns)**
   - Inputs: `iA = 2`, `iB = 5`
   - Operation: Addition (`nAdd_Sub = 0`)
   - Expected and observed result: `oResult = 7` which correctly represents the addition of 2 and 5.
2. **Test 2: Subtraction (20ns to 40ns)**
   - Inputs: `iA = 10`, `iB = 4`
   - Operation: Subtraction (`nAdd_Sub = 1`)
   - Expected and observed result: `oResult = 6` which correctly captures the result of subtracting 4 from 10.
3. **Test 3: Overflow Addition (40ns to 60ns)**
   - Inputs: `iA = 0xFFFFFFFF`, `iB = 1` (attempting to simulate overflow)
   - Operation: Addition (`nAdd_Sub = 0`)
   - Expected and observed result: `oResult = 0` due to overflow, which is correctly handled by the simulator.
4. **Test 4: Zero Addition (60ns to 80ns)**
   - Inputs: `iA = 0`, `iB = 0`
   - Operation: Addition (`nAdd_Sub = 0`)
   - Expected and observed result: `oResult = 0`, correctly reflecting the addition of zero and zero.

5. **Test 5: Negative Result Subtraction (80ns to 100ns)**
    - Inputs: `iA = 2`, `iB = 6`
    - Operation: Subtraction (`nAdd_Sub = 1`)
    - Expected and observed result: `oResult = -4`, correctly showing the subtraction of 6 from 2 yielding a negative result.