Twos Complement Task 6
Ubaid ur Rehman (Embedded Intern)

Task Given By:
Inam ur Rehman Shahid

Task Assigned Date: May 19, 2024,

Task Finished Date: May 19, 2024

Designing a circuit that computes the twos complement of a 10-bit binary number. Designing Two Design Alternates.

Designing a circuit to compute the two's complement of a 10-bit binary number involves inverting all the bits of the number and then adding 1 to the least significant bit (LSB). Here are two design alternatives for achieving this:

Design Alternative 1: Using Bitwise Inversion and Adder

Components:

- NOT gates (inverters) for each bit
- 10-bit binary adder

Steps:

1. Inversion Stage:
    - Use 10 NOT gates to invert each bit of the 10-bit binary number. This will give you the one's complement of the number.
2. Addition Stage:
    - Use a 10-bit binary adder to add 1 to the result of the inversion stage. This will convert the one's complement to the two's complement.

Circuit Diagram:

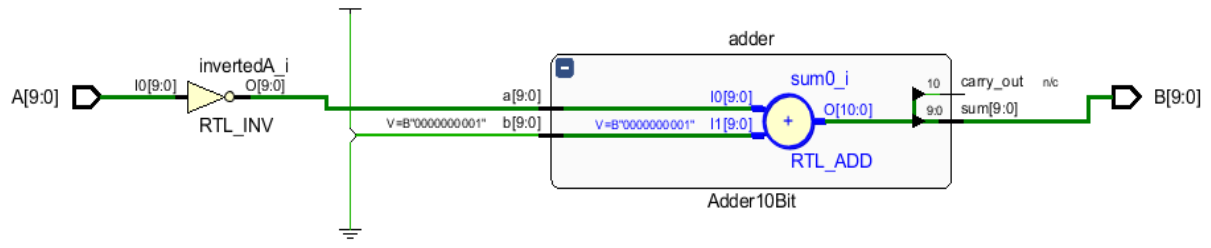


Figure 1: Block Diagram of the Design 1 Circuit

1. Input: 10-bit binary number $A=A9A8A7A6A5A4A3A2A1A0A=A9A8A7A6A5A4A3A2A1A0$
2. Inversion Stage:
    - Inverted bits=$\overline{A9\ A8\ A7\ A6\ A5\ A4\ A3\ A2\ A1\ A0}$ Inverted bits=$A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$
3. Addition Stage:
    - Add 1 to the inverted bits using a 10-bit binary adder.

Circuit Description:

- Each bit of the input number *AA* is fed into a NOT gate.
- The outputs of the NOT gates are then fed into a 10-bit binary adder.
- The second input to the 10-bit binary adder is the binary number 0000000001 (which represents 1).
- The output of the adder is the 10-bit two's complement of the input number.

Time Efficiency Analysis:

1. Inversion Stage:

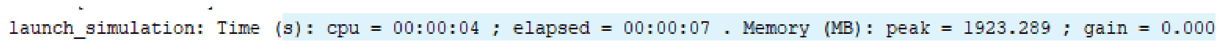
- Each NOT gate has a propagation delay. Assuming a delay of $\mathrm{t}_{\mathrm{total}}$, the total delay for this stage is t_{NOT} because all NOT gates operate in parallel.

2. Addition Stage:

- A 10-bit binary adder typically consists of several stages of full adders. The propagation delay of a ripple carry adder (a common type) is approximately proportional to the number of bits, n, because the carry must propagate through all bits.
- Assuming each full adder stage has a delay of t_{FA}, the total delay for the adder is $10 \cdot t_{\mathrm{FA}}$ for a ripple carry adder.

Total Delay: $t_{total1} = t_{NOT} + 10 \cdot t_{FA}$

Total Time Elapsed:



```
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:07 . Memory (MB): peak = 1923.289 ; gain = 0.000
```

As shown in Picture. The total time taken for simulating Design 1 took 07 seconds. Now, lets see on the Design 2.

Design Alternative 2: Using XOR Gates and Incrementor

Components:

- XOR gates
- 10-bit binary incrementor (a specialized form of an adder)

Steps:

1. XOR Stage:
   - Use 10 XOR gates to invert each bit of the 10-bit binary number by XORing each bit with 1.
2. Incrementor Stage:
   - Use a 10-bit binary incrementor to add 1 to the result from the XOR stage.

Circuit Diagram:

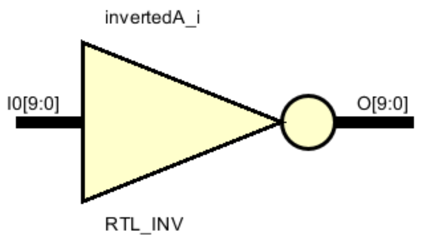


Figure 2: Block Diagram of Design 2

1. Input: 10-bit binary number $\boldsymbol{A}$=$\boldsymbol{A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0}$
2. XOR Stage:
    - Inverted bits=$(A_9 \oplus 1)(A_8 \oplus 1)(A_7 \oplus 1)(A_6 \oplus 1)(A_5 \oplus 1)(A_4 \oplus 1)(A_3 \oplus 1)(A_2 \oplus 1)(A_1 \oplus 1)(A_0 \oplus 1)$
3. Incrementor Stage:
    - Use a 10-bit incrementor to add 1 to the result of the XOR stage.

Circuit Description:

- Each bit of the input number AA is fed into an XOR gate along with a constant 1.
- The outputs of the XOR gates represent the one's complement of the input number.
- The outputs of the XOR gates are then fed into a 10-bit binary incrementor.
- The output of the incrementor is the 10-bit two's complement of the input number.

Time Efficiency Analysis:

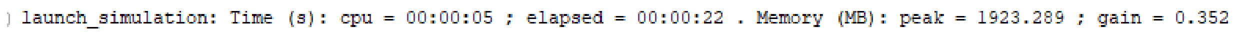
1. XOR Stage

- Each XOR gate has a propagation delay. Assuming a delay of t_{XOR} the total delay for this stage is t_{XOR} because all XOR gates operate in parallel.

2. Incrementor Stage

- A 10-bit binary incrementor can be implemented similarly to a binary adder. If implemented as a ripple carry incrementor, it has the same delay as a ripple carry adder.
- The delay is $10\cdot t_{\mathrm{FA}}$, where t_{FA} is the delay per stage of the incrementor.

Total Delay: $\boldsymbol{t_{\mathrm{total2}}=t_{\mathrm{XOR}}+10\cdot t_{\mathrm{FA}}}$


```
) launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:22 . Memory (MB): peak = 1923.289 ; gain = 0.352
```

As shown in Picture. The total time taken for simulating Design 2 took 22 seconds.

Comparison of Alternatives:

1. Simplicity:
   - The first design is more straightforward, using basic NOT gates and a standard binary adder.
   - The second design uses XOR gates, which might be seen as more complex but essentially perform the same function as the NOT gates in this context.
2. Efficiency:
   - Both designs are similar in terms of the number of gates and components used.
   - The choice between a binary adder and an incrementor might depend on the available components and design preferences.
3. Performance:
   - Both designs will have similar propagation delays since they both involve inversion followed by addition.
   - The performance differences will be minor and typically depend on the specific implementations of the adder and incrementor.
4. Comparison of Delays:
   - NOT gate delay (t_{NOT}) vs XOR gate delay (t_{XOR}): Typically, the delay of an XOR Gate (t_{XOR}) is silighty higher than that of a NOT gate (t_{NOT}) because XOR gates are more complex.
   - Both Designs involve the same adder delay, $10 \cdot t_{\mathrm{FA}}$

Given that $t_{XOR} > t_{NOT}$, the first design (NOT gates and an adder) is likely to be slightly more time efficient than the second design (XOR gates and an incrementor).

Verilog Code for Design 1:

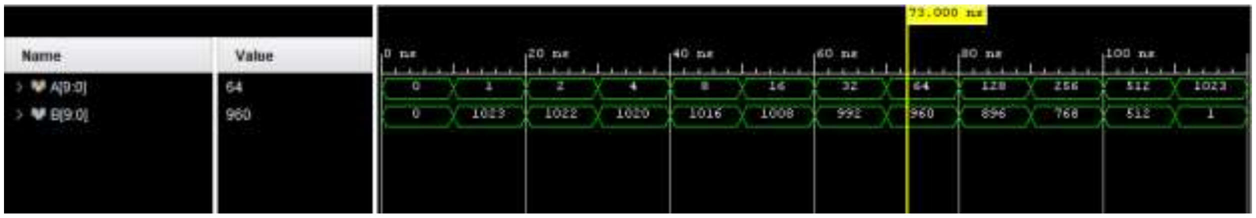
```
`timescale 1ns / 1ps
module TwosComplement_Inversion_Adder (
   input [9:0] A,
   output [9:0] B
);
   wire [9:0] invertedA;
   wire [9:0] one = 10'b0000000001;
   wire carry_out;

   // Invert each bit
   assign invertedA = ~A;

   // 10-bit adder to add 1
   Adder10Bit adder (
      .a(invertedA),
      .b(one),
      .sum(B),
      .carry_out(carry_out)
   );
```

```

endmodule

// 10-bit Adder Module
module Adder10Bit (
   input [9:0] a,
   input [9:0] b,
   output [9:0] sum,
   output carry_out
);
   assign {carry_out, sum} = a + b;
endmodule
```

Verilog Testbench:

```
`timescale 1ns / 1ps
module tb_TwosComplement_Inversion_Adder;
   reg [9:0] A;
   wire [9:0] B;

   // Instantiate the design under test (DUT)
   TwosComplement_Inversion_Adder dut (
      .A(A),
      .B(B)
   );

   initial begin
      // Monitor changes to A and B
      $monitor("Time: %0t | A: %b | B (Two's Complement): %b", $time, A, B);

      // Apply test vectors
      A = 10'b0000000000; #10;
      A = 10'b0000000001; #10;
      A = 10'b0000000010; #10;
      A = 10'b0000000100; #10;
      A = 10'b0000001000; #10;
      A = 10'b0000010000; #10;
      A = 10'b0000100000; #10;
      A = 10'b0001000000; #10;
      A = 10'b0010000000; #10;
      A = 10'b0100000000; #10;
      A = 10'b1000000000; #10;
      A = 10'b1111111111; #10;

      $stop;
   end
endmodule
```

Verified Test Result:



Conclusion:

Design Alternative 1 (using bitwise inversion with NOT gates followed by a binary adder) is generally more time efficient due to the typically lower delay of NOT gates compared to XOR gates. Both designs share the same delay for the addition stage, so the difference in time efficiency primarily hinges on the delay of the inversion stage.