Lab 5: Structural Modeling Using Regular Structure

The basic idea of iterative (or regular) circuits is to decompose a function into a set of simpler functions that can cascaded together. The chain of cascaded circuits incrementally computes the desired function. We will use 2-bit to represent 3 possible comparison results. 00(a=b), 01(a<b), 10(a>b), and 11(not use). Denote (c_in1, c_in0) to be the comparison result of the previous high-order bit and denote (c_out1,c_out0) be the current comparison result. The truth table of bit comparison is given in TABLE 1.

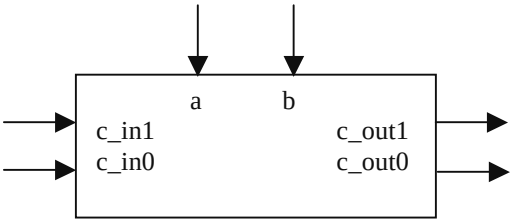


Figure 1. Bit_compare block diagram

TABLE 1. Bit Comparison truth Table

c_in1	c_in0	a	b	c_out1	c_ou0	comment
0	0	0	0	0	0	a=b
0	0	0	1	0	1	a<b
0	0	1	0	1	0	a>b
0	0	1	1	0	0	a=b
0	1	0	0	0	1	a<b
0	1	0	1	0	1	a<b
0	1	1	0	0	1	a<b
0	1	1	1	0	1	a<b
1	0	0	0	1	0	a>b
1	0	0	1	1	0	a>b
1	0	1	0	1	0	a>b
1	0	1	1	1	0	a>b
1	1	0	0	X	X	?
1	1	0	1	X	X	?
1	1	1	0	X	X	?
1	1	1	1	X	X	?

The above table can be summarize as follows:

1. If the previous comparison is equal (00), then the next comparison result is set according to the relative values of a and b. That is, 00 if a=b; 01 if a<b; 10 if a>b.
2. If the previous is not equal (01 or 10), then next comparison result is to (01 or 10) irrespective of the values a and b.
3. The previous comparison result of 11 is not being used. It represent a don't care condition.

The simplified expression for the comparison output is given below:

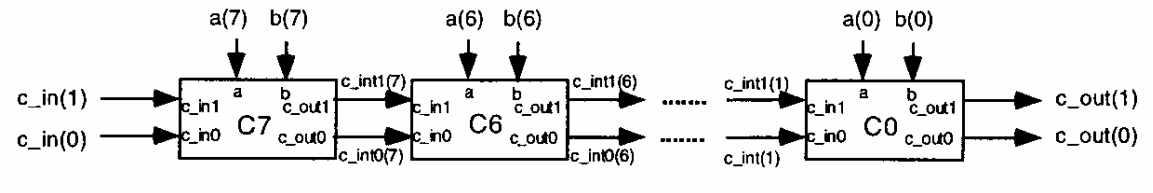
c_out1 = a b' c_in0' + c_in1
c_out0 = a' b c_in1' + c_in0

Bit_Compare Logic Architecture

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;

ENTITY bit_compare IS
        PORT(a, b, c_in1, c_in0: IN BIT;
                c_out1, c_out0:  OUT BIT));
END bit_compare;

Architecture logic of  bit_conpare is
Begin
        c_out1<=(a AND NOT(b) AND NOT(c_in0)) OR c_in1;
        c_out0<=( NOT ( a )AND b AND NOT(c_in1)) OR c_in0;
end logic;
```




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;

ENTITY compare8 IS
        PORT (a, b : IN BIT_VECTOR(7 DOWNTO 0);
                c_in : IN BIT_VECTOR(1 DOWNRO 0);
                c_out : OUT BIT_VECTOR(1 DOWNTO 0));
END compare8;
```

Architecture 1: Abstract Behavior Modelling

```
ARCHITECTURE abstract_behavior of  compare8 IS
BEGIN
        PROCESS(c_in,a, b)
        BEGIN
                IF(c_in="00") THEN
                        IF(a=b) THEN c_out<="00";
                        ELSIF(a<b) THEN c_out<="01";
                        ELSIF(a>b) THEN c_out<="10";
                        END IF;
                ELSIF(c_in="01") THEN c_out<="01";
                ELSIF(c_in="10") THEN c_out<="10";
                END IF;
        END PROCESS;
END abstract_behavior;
```

Architecture 2: Manual Structural Modelling

```
ARCHITECTURE man_struct OF compare8 IS
        COMPONENT bit_compare
```

```
                PORT(a, b, c_in1, c_in0: IN BIT;
                        c_out1, c_out0:  OUT BIT));
        END COMPONENT;

        SIGNAL c_int1, c_int0: BIT_VECTOR(7 DOWNTO 1);
        For all: bit_compare use entity work.bit_compare(logic);
BEGIN
        C7: bit_compare PORT MAP(a(7), b(7), c_in(1), c_in(0), c_int1(7), c_int0(7));
        C6: bit_compare PORT MAP(a(6), b(6), c_int1(7), c_int0(7), c_int1(6), c_int0(6));
        C5: bit_compare PORT MAP(a(5), b(5), c_int1(6), c_int0(6), c_int1(5), c_int0(5));
        C4: bit_compare PORT MAP(a(4), b(4), c_int1(5), c_int0(5), c_int1(4), c_int0(4));
        C3: bit_compare PORT MAP(a(3), b(3), c_int1(4), c_int0(4), c_int1(3), c_int0(3));
        C2: bit_compare PORT MAP(a(2), b(2), c_int1(3), c_int0(3), c_int1(2), c_int0(2));
        C1: bit_compare PORT MAP(a(1), b(1), c_int1(2), c_int0(2), c_int1(1), c_int0(1));
        C0: bit_compare PORT MAP(a(0), b(0), c_int1(1), c_int0(1), c_out(1), c_out(0));
END man_struct;
```

Architecture 3: Generated Structural Modelling

```
ARCHITECTURE gen_struct OF compare8 IS
        COMPONENT bit_compare
                PORT(a, b, c_in1, c_in0: IN BIT; c
                        c_out1, c_out0:  OUT BIT));
        END COMPONENT;
        SIGNAL c_int1, c_int0: BIT_VECTOR(7 DOWNTO 1);
---     FOR ALL: bit_compare USE ENTITY work.bit_compare(logic);
BEGIN
        CASCADE:        --Iteration generate
        FOR i IN 7 DOWNTO 0 GENERATE
        INPUT_CASE:
        IF (i=7) GENERATE
                C7: bit_compare PORT MAP (a(i), b(i), c_in(1), c_in(0), c_int1(i), c_int0(i));
        END GENERATE INPUT_CASE;

        NORMAL_CASE:
        IF(i<=6 AND i>=1) GENERATE
                CX: bit_compare PORT MAP(a(i), b(i), c_int1(i+1), c_int0(i+1), c_int1(i), c_int0(i));
        END GENERATE NORMAL_CASE;

        OUTPUT_CASE:
        IF(I=0) GENERATE
                C0: bit_compare PORT MAP (a(I), b(I), c_int1(I+1), c_int0(I+1), c_out(1), c_out(0));
        END GENERATE OUTPUT CASE;
        END GENERATE CASCADE;
END gen_struct;
```

Assignments

1. Create a testbench to verify that each of the three architectures is working correctly.
2. Synthesis each of the three architectures.
3. Compare the total area of each of the synthesized architectures. The total area is proportional to the circuit complexity.
4. Determine the longest path from input to output of each of the three architectures. This is determined by tracing each of the signal paths from input to output, and counting the number of gates traversed. The longest path is the path with the most gates traversed. This is proportional to propagation delay.