VHDL Exercise – Z0965020

Circuits and Coding

Ripple Adder

```
23  LIBRARY ieee;
24  USE ieee.std_logic_1164.all;
25
26
27  --  Entity Declaration
28
29  ENTITY fullAdder IS
30      -- {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
31      PORT
32      (
33          A : IN STD_LOGIC;
34          B : IN STD_LOGIC;
35          Cin : IN STD_LOGIC;
36          S : OUT STD_LOGIC;
37          Cout : OUT STD_LOGIC
38      );
39      -- {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!
40
41  END fullAdder;
42
43
44  --  Architecture Body
45
46  ARCHITECTURE fullAdder_architecture OF fullAdder IS
47
48      signal sig1, sig2, sig3 : std_logic;
49  BEGIN
50      sig1 <= A XOR B;
51      sig2 <= A AND B;
52      sig3 <= Cin AND sig1;
53      S    <= sig1 XOR Cin;
54      Cout <= sig2 OR sig3;
55  END fullAdder_architecture;
56
```

Figure 1

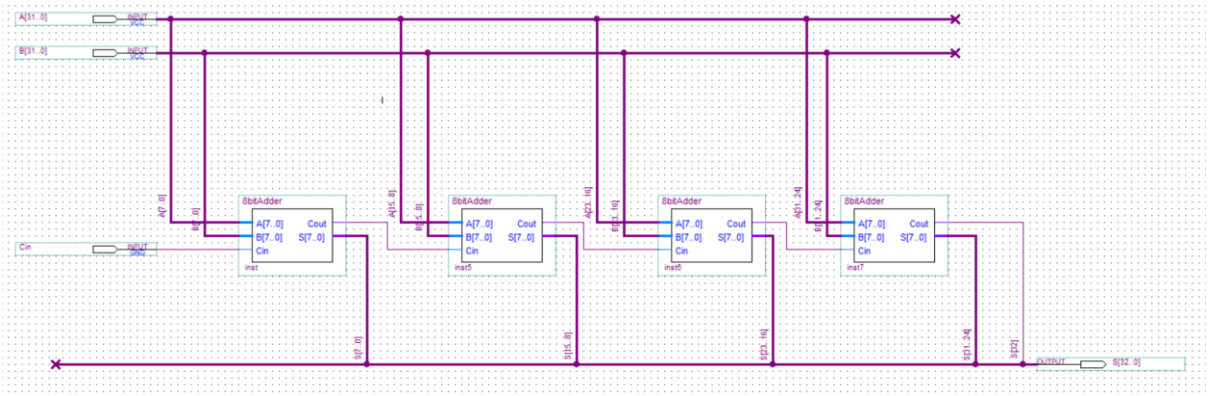

Figure 2

To create the 32-bit ripple adder (figure 2), I first created a full adder in VHDL (figure 1), created a symbol file for it, then created a 2 bit, 4 bit and 8bit adder (pictures omitted due to similarity to 32 bit adder). By creating these smaller adders first, I could keep the schematic diagrams small so that any errors in the diagram can be spotted easily.

Carry Select Adder (CSA)

The carry select adder makes use of the 8-bit ripple adder I had already created, but also required a 1-bit 2 input MUX and an 8-bit 2 input MUX, both of which I created in VHDL (figures 3 and 4)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;


--  Entity Declaration

ENTITY mux IS
    -- {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
    PORT
    (
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        SEL : IN STD_LOGIC;
        S : OUT STD_LOGIC

    );
    -- {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!

END mux;


--  Architecture Body

ARCHITECTURE mux_architecture OF mux IS

    signal sig1, sig2 :std_logic;
BEGIN
    sig1 <= A AND NOT(SEL);
    sig2 <= B AND SEL;
    S   <=  sig1 OR sig2;
END mux_architecture;
```

Figure 3

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;


--  Entity Declaration

ENTITY mux8bit IS
    -- {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
    PORT
    (
        A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        SEL : IN STD_LOGIC;
        S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)

    );
    -- {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!

END mux8bit;


--  Architecture Body

ARCHITECTURE mux8bit_architecture OF mux8bit IS

BEGIN

    S   <= A when (SEL = '0') else B;
END mux8bit_architecture;
```

Figure 4

Once I had created the required Multiplexors, I could create the sub-circuits (figure 5) which were comprised of two 8-bit adders and one of each type of MUX

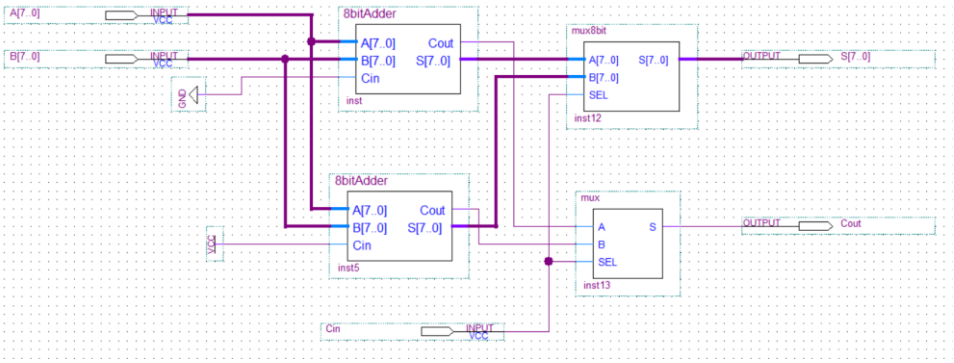

Figure 5

The 32-bit CSA could then be constructed by using one 8-bit ripple adder followed by 3 of these sub-circuits.

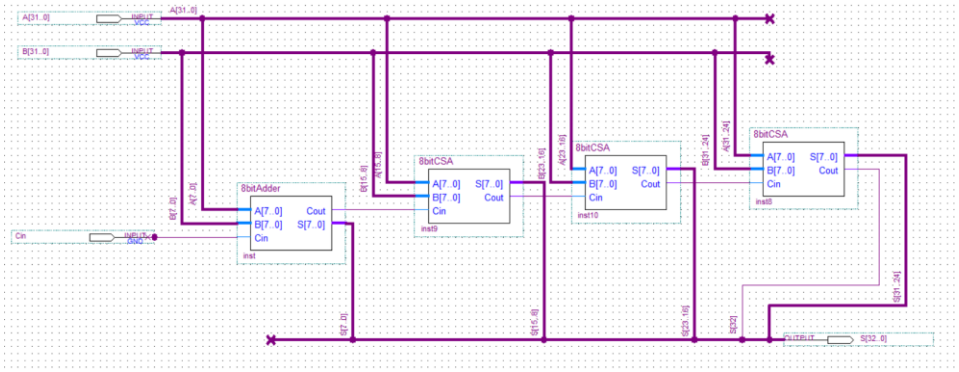

Figure 6

Speed and Operation

For each of the two adders I created a waveform which had the same changes in input values. I could not test the timings for all possible combinations of input values, so I chose the values of all bits set to 1 (all_1), the least significant bit set to 1, or all bits set to 0

Input Values				CSA				Ripple		
A	B	Cin		input change	output change	delay (ns)		input change	output change	delay (ns)
1	1	0		30	41.72	11.72		30	41.464	11.464
1	1	1		60	71.548	11.548		60	69.239	9.239
all_1	1	0		90	126.024	36.024		90	147.158	57.158
all_1	1	1		180	191.548	11.548		150	159.238	9.238
all_1	all_1	0		240	255.103	15.103		210	222.334	12.334
all_1	all_1	1		300	311.548	11.548		270	279.238	9.238
0	0	0		360	376.194	16.194		330	346.414	16.414

Figure 7 (all times are in ns)

Observations to be made from this data:

Highlighted cells in Figure 7 represent output changes that were preceded by glitches before the correct value was reached. Both types of adder experienced glitches at the same input changes. These happened to be changes where many bits had their value changed. These glitches are likely caused by the ripple effect, as the output value will rapidly change values as the carry bit propagates through the adder.

Secondly, the results show that the Ripple adder generally performs faster than the CSA. I had expected the CSA to be faster due to the reduced carry delay. I expect that these unexpected results may be due to the limitations of the software's timing simulator.

Another detail with remark is that the delay when changing many bits in A is much larger than the delay when changing the same number of bits in B.

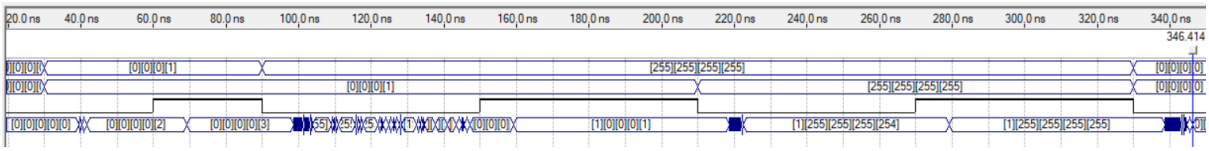

Figure 8: Ripple adder waveform

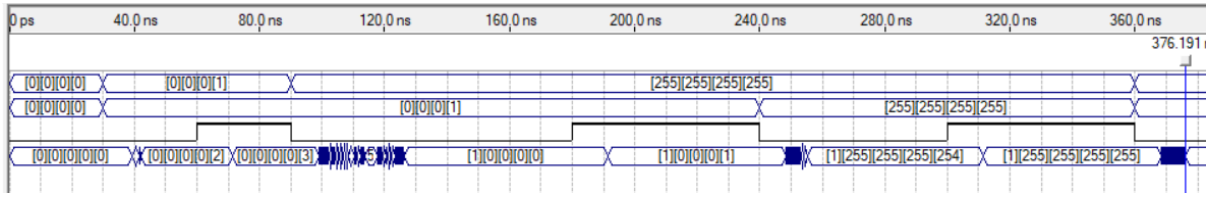

Figure 9: CSA waveform

FPGA implementation

While FPGAs will always perform worse than a specifically designed ASIC, in the case of our circuit it would not be worth investing in an ASIC, as due to the small size of the circuit the delays produced are also very small, so even if an ASIC performed 10 times faster, it would only reduce the delay by no more than 50ns, which is not worth paying the huge development cost for.