Digital Systems VLSI

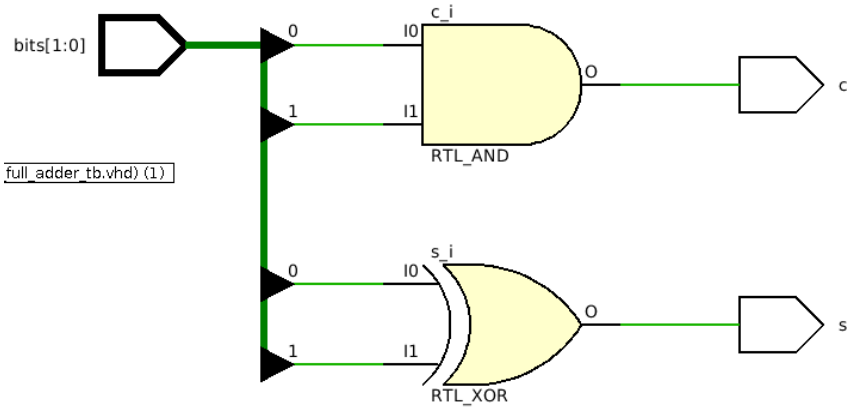
Nanos Georgios

Topic 1)

The following is the Dataflow description of the Half Adder (HA):

```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is port(
  bits: in std_logic_vector(1 downto 0);
  s: out std_logic;
  c: out std_logic
);
end half_adder;

architecture dataflow of half_adder is
begin
  s <= bits(0) xor bits(1);
  c <= bits(0) and bits(1);
end dataflow;
```

The corresponding structural diagram (RTL schematic) is the following:

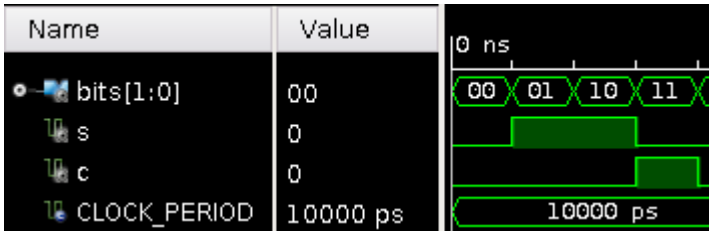


The following is the testbench file for the implementation of the simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity half_adder_tb is
end half_adder_tb;

architecture bench of half_adder_tb is

  component half_adder is port (
    bits: in std_logic_vector(1 downto 0);
    s: out std_logic;
    c: out std_logic
  );
  end component;

  signal bits: std_logic_vector(1 downto 0) := "00";
  signal s: std_logic;
  signal c: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: half_adder port map (
    bits  => bits,
    s  => s,
```

```
    c  => c
  );

  stimulus: process
  begin
    for i in 0 to 3 loop
      bits <= std_logic_vector(to_unsigned(i, 2));
      wait for CLOCK_PERIOD;
    end loop;
    bits <= (others => '0');
    wait;
  end process;

end bench;
```

The following are snapshots of the simulation:



Finally, the critical path, with a total delay of 5,579ns, is estimated to be the path from the bits given as input (bits (0), bits (1)) to the exit of the XOR gate (s) . This is justified by the fact that an XOR gate, which is a two-level gate, is used to calculate the sum s, while a simple AND gate is used to calculate the retained output (c).

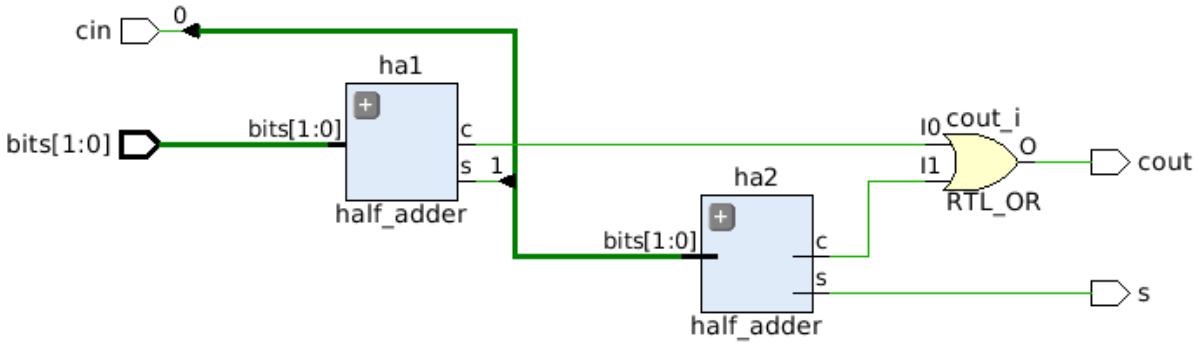
Topic 2)

The following is the Structural description of the Full Adder (FA):

```
library ieee;
use ieee.std_logic_1164.all;

entity full_adder is port(
```

```
  bits: in std_logic_vector(1 downto 0);
  cin: in std_logic;
  s: out std_logic;
  cout: out std_logic
);
end full_adder;

architecture structural of full_adder is

  component half_adder is port(
    bits: in std_logic_vector(1 downto 0);
    s: out std_logic;
    c: out std_logic
  );
  end component;

  signal ha1_s: std_logic;
  signal ha1_c: std_logic;
  signal ha2_bits: std_logic_vector(1 downto 0);
  signal ha2_c: std_logic;

begin

  ha1: half_adder port map(
    bits => bits,
    s => ha1_s,
    c => ha1_c
  );

  ha2_bits <= ha1_s&cin;

  ha2: half_adder port map(
    bits => ha2_bits,
    s => s,
    c => ha2_c
  );

  cout <= ha1_c or ha2_c;

end structural;
```

The corresponding structural diagram (RTL schematic) is the following:



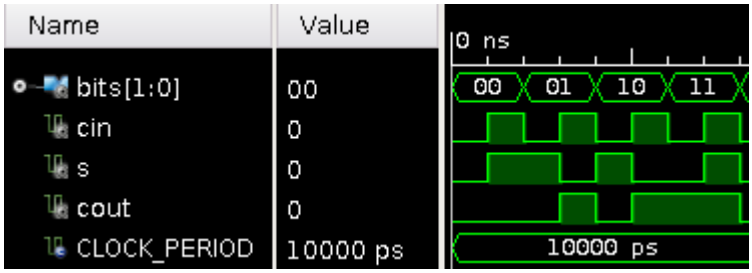
We see that the full adder uses two half-adders, to one are added the two input bits and to the other the sum resulting from the first with the input holder.

The following is the testbench file for the implementation of the simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_tb is
end full_adder_tb;

architecture bench of full_adder_tb is

  component full_adder is port (
     bits: in std_logic_vector(1 downto 0);
     cin: in std_logic;
     s: out std_logic;
     cout: out std_logic
  );
  end component;

  signal bits: std_logic_vector(1 downto 0) := "00";
  signal cin: std_logic := '0';
  signal s: std_logic;
```

```
  signal cout: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: full_adder port map (
    bits  => bits,
    cin => cin,
    s  => s,
    cout  => cout
  );

  stimulus: process
  begin
    for i in 0 to 3 loop
      bits <= std_logic_vector(to_unsigned(i, 2));
      cin <= '0';
      wait for CLOCK_PERIOD;
      cin <= '1';
      wait for CLOCK_PERIOD;
    end loop;
    bits <= (others => '0');
    cin <= '0';
    wait;
  end process;

end bench;
```

The following is the testbench file for the implementation of the simulation:



The critical path, finally, as in the semiconductor of the previous query, is 5,579ns (input bits to output s), although in the RTL schematic this path consists of two XOR gates, as opposed to one XOR

of the semiconductor. The same delay is due to the structures of FPGA (LUT), because, although at the level of logical gates we have paths of different lengths, at the level of FPGA the paths are practically equal.

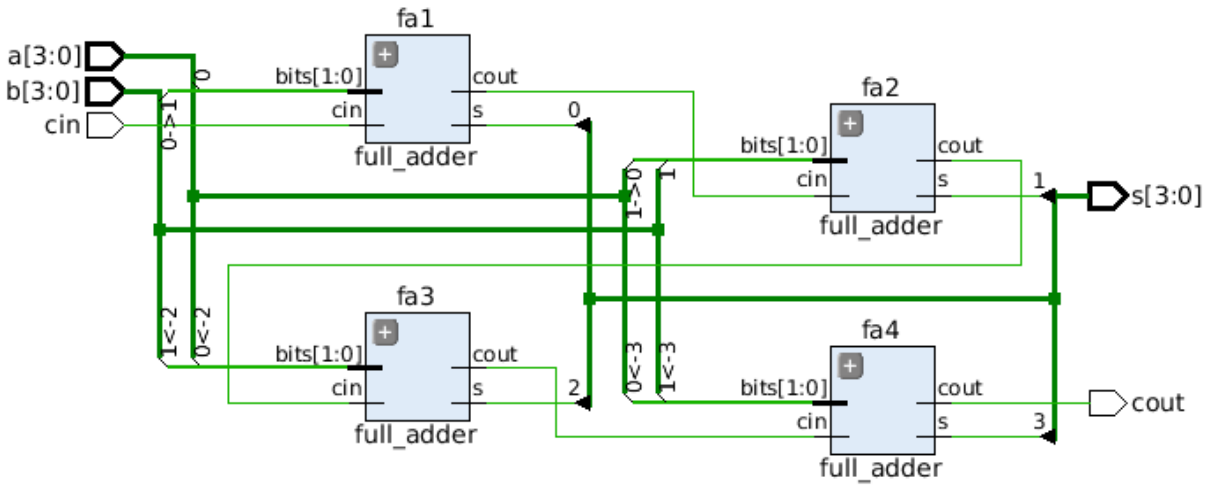
Topic 3)

The following is the Structural description of the 4-bit Parallel Adder (4-bit PA):

```
library ieee;
use ieee.std_logic_1164.all;

entity parallel_adder is port(
  cin: in std_logic;
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
  s: out std_logic_vector(3 downto 0);
  cout: out std_logic
);
end parallel_adder;

architecture structural of parallel_adder is

  component full_adder is port(
    bits: in std_logic_vector(1 downto 0);
    cin: in std_logic;
    s: out std_logic;
    cout: out std_logic
  );
  end component;

  signal fa1_s: std_logic;
  signal fa1_cout: std_logic;
  signal fa2_s: std_logic;
  signal fa2_cout: std_logic;
  signal fa3_s: std_logic;
  signal fa3_cout: std_logic;
  signal fa4_s: std_logic;

begin
```

```
  fa1: full_adder port map(
     cin => cin,
     bits(0) => a(0),
     bits(1) => b(0),
     s => fa1_s,
     cout => fa1_cout
  );

  fa2: full_adder port map(
     cin => fa1_cout,
     bits(0) => a(1),
     bits(1) => b(1),
     s => fa2_s,
     cout => fa2_cout
  );

  fa3: full_adder port map(
     cin => fa2_cout,
     bits(0) => a(2),
     bits(1) => b(2),
     s => fa3_s,
     cout => fa3_cout
  );

  fa4: full_adder port map(
     cin => fa3_cout,
     bits(0) => a(3),
     bits(1) => b(3),
     s => fa4_s,
     cout => cout
  );

  s <= fa4_s&fa3_s&fa2_s&fa1_s;

end structural;
```

The corresponding structural diagram (RTL schematic) is the following:



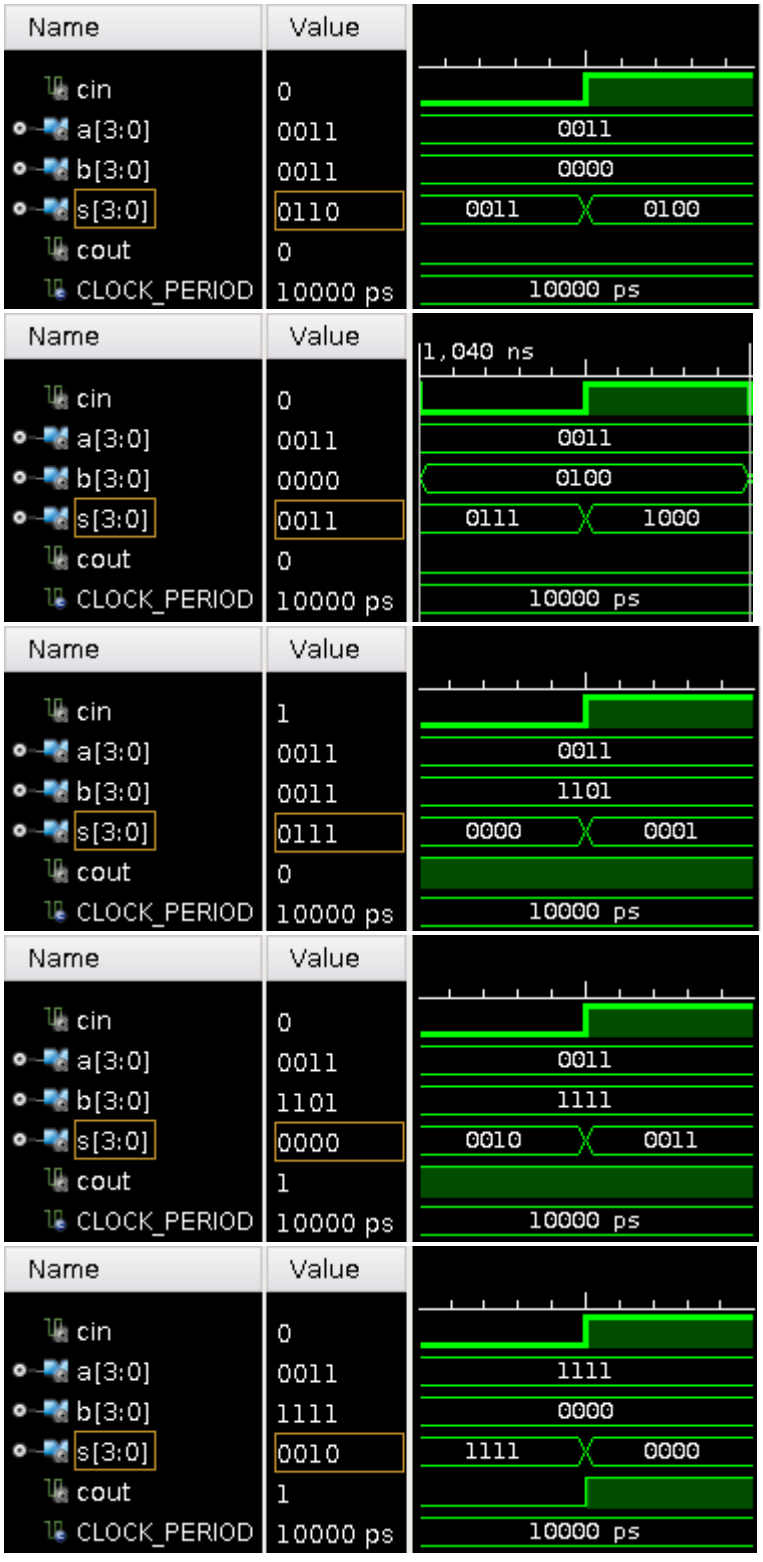
The 4-bit parallel adder consists of 4 complete adders, one for each bit of the add-ons.

The following is the testbench file for the implementation of the simulation:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity parallel_adder_tb is
end parallel_adder_tb;

architecture bench of parallel_adder_tb is

  component parallel_adder is port (
     cin: in std_logic;
     a: in std_logic_vector(3 downto 0);
     b: in std_logic_vector(3 downto 0);
     s: out std_logic_vector(3 downto 0);
     cout: out std_logic
  );
  end component;

  signal cin: std_logic := '0';
  signal a: std_logic_vector(3 downto 0) := "0000";
  signal b: std_logic_vector(3 downto 0) := "0000";
```

```
  signal s: std_logic_vector(3 downto 0);
  signal cout: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: parallel_adder port map (
     cin => cin,
     a => a,
     b => b,
     s  => s,
     cout  => cout
  );

  stimulus: process
  begin
     for i in 0 to 15 loop
        a <= std_logic_vector(to_unsigned(i, 4));
        for j in 0 to 15 loop
           b <= std_logic_vector(to_unsigned(j, 4));
           cin <= '0';
           wait for CLOCK_PERIOD;
           cin <= '1';
           wait for CLOCK_PERIOD;
        end loop;
     end loop;
     a <= (others => '0');
     b <= (others => '0');
     cin <= '0';
     wait;
  end process;

end bench;
```

Here are some snapshots of the simulation:



In the critical path, as expected, we find the path from the input (a / b) to the last output bit (s [3])

(6.920ns). Since the calculation of s [3] by the last adder requires that the outputs of the balances be calculated, the fact that it is the critical path makes the most sense.

Topic 4)

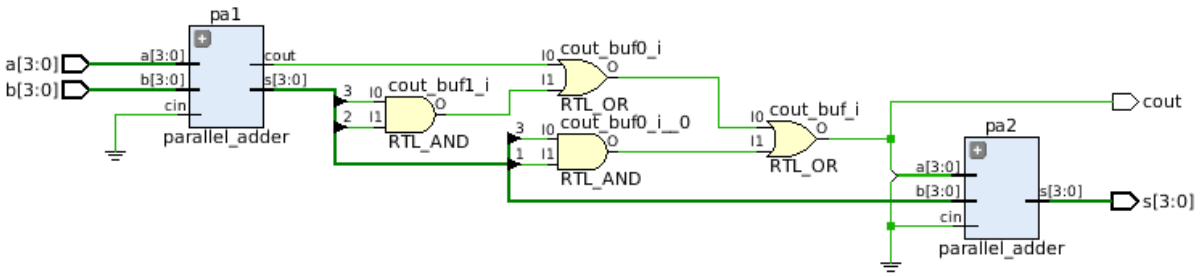
To implement the BCD full adder (with input holder) we used a BCD semi-adder structure (without input holder). The full BCD adder uses 2 BCD semi-adders, the first adds the 2 adders and the second adds the sum resulting from the first with the input holder. That is, the design was done according to the simple adders of the previous questions.

The following is the Structural description of the BCD Half Adder (BCD HA):

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_half_adder is port(
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
  s: out std_logic_vector(3 downto 0);
  cout: out std_logic
);
end bcd_half_adder;

architecture structural of bcd_half_adder is

  component parallel_adder is port(
    cin: in std_logic;
    a: in std_logic_vector(3 downto 0);
    b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    cout: out std_logic
  );
  end component;

  signal pa1_s: std_logic_vector(3 downto 0);
  signal pa1_cout: std_logic;
  signal pa2_a: std_logic_vector(3 downto 0);
  signal cout_buf: std_logic;

begin

  pa1: parallel_adder port map(
    cin => '0',
```

```
    a => a,
    b => b,
    s => pa1_s,
    cout => pa1_cout
  );

  cout_buf <= pa1_cout or (pa1_s(3) and pa1_s(2)) or (pa1_s(3) and pa1_s(1));
  pa2_a <= '0'&cout_buf&cout_buf&'0';

  pa2: parallel_adder port map(
    cin => '0',
    a => pa2_a,
    b => pa1_s,
    s => s
  );

  cout <= cout_buf;

end structural;
```

The corresponding structural diagram (RTL schematic) is the following:



The BCD half-adder uses 2 parallel adders to add 0110 in the event that the sum resulting from the first overflows.

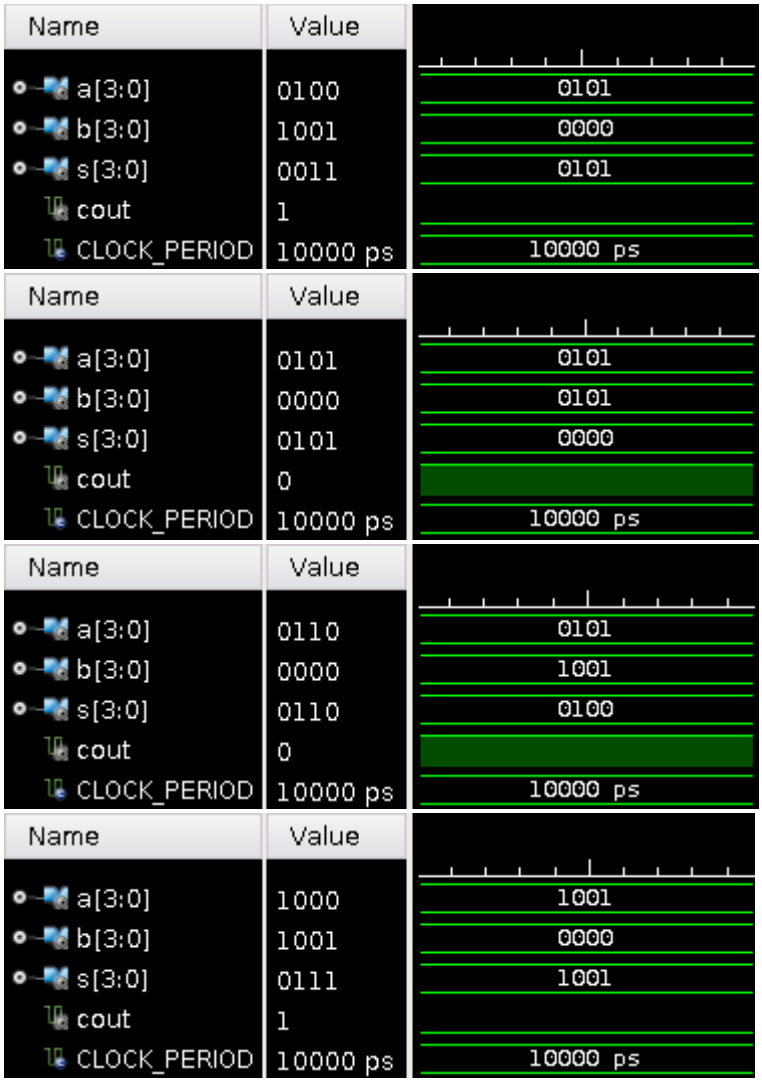
The following is the testbench file for the implementation of the simulation:

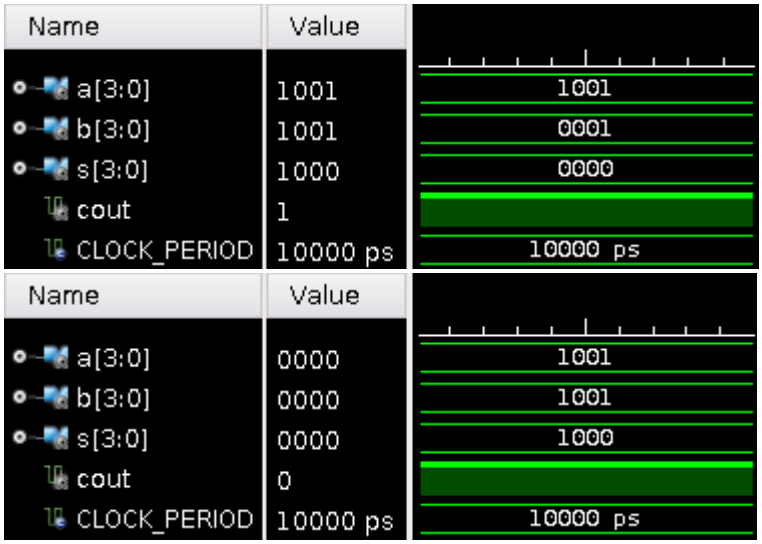
```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity bcd_half_adder_tb is
end bcd_half_adder_tb;

architecture bench of bcd_half_adder_tb is

  component bcd_half_adder is port (
    a: in std_logic_vector(3 downto 0);
    b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    cout: out std_logic
  );
  end component;

  signal a: std_logic_vector(3 downto 0) := "0000";
  signal b: std_logic_vector(3 downto 0) := "0000";
  signal s: std_logic_vector(3 downto 0);
  signal cout: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: bcd_half_adder port map (
    a => a,
    b => b,
    s  => s,
    cout  => cout
  );

  stimulus: process
  begin
    for i in 0 to 9 loop
      a <= std_logic_vector(to_unsigned(i, 4));
      for j in 0 to 9 loop
        b <= std_logic_vector(to_unsigned(j, 4));
        wait for CLOCK_PERIOD;
      end loop;
    end loop;
```

```
    a <= (others => '0');
    b <= (others => '0');
    wait;
  end process;

end bench;
```

Here are some snapshots of the simulation:



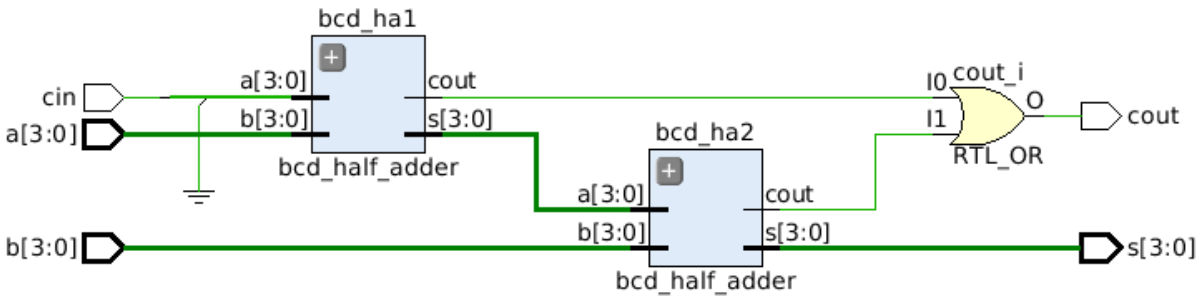


The following is the Structural description of the BCD Full Adder (BCD FA):

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_full_adder is port(
  cin: in std_logic;
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
  s: out std_logic_vector(3 downto 0);
  cout: out std_logic
);
end bcd_full_adder;

architecture structural of bcd_full_adder is

  component bcd_half_adder is port(
    a: in std_logic_vector(3 downto 0);
    b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    cout: out std_logic
  );
  end component;

  signal bcd_ha1_a: std_logic_vector(3 downto 0);
  signal bcd_ha1_s: std_logic_vector(3 downto 0);
```

```
  signal bcd_ha1_cout: std_logic;
  signal bcd_ha2_cout: std_logic;

begin

  bcd_ha1_a <= "000"&cin;
  bcd_ha1: bcd_half_adder port map(
     a => bcd_ha1_a,
     b => a,
     s => bcd_ha1_s,
     cout => bcd_ha1_cout
  );

  bcd_ha2: bcd_half_adder port map(
     a => bcd_ha1_s,
     b => b,
     s => s,
     cout => bcd_ha2_cout
  );

  cout <= bcd_ha1_cout or bcd_ha2_cout;

end structural;
```

The corresponding structural diagram (RTL schematic) is the following:



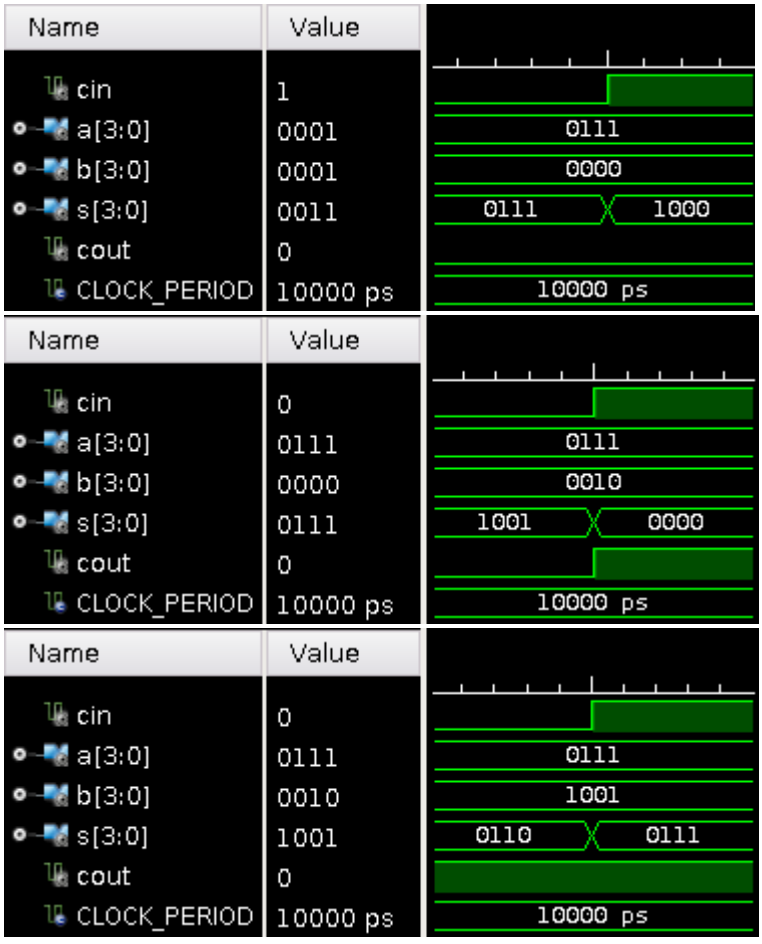
The complete BCD adder uses 2 BCD semi-adders.

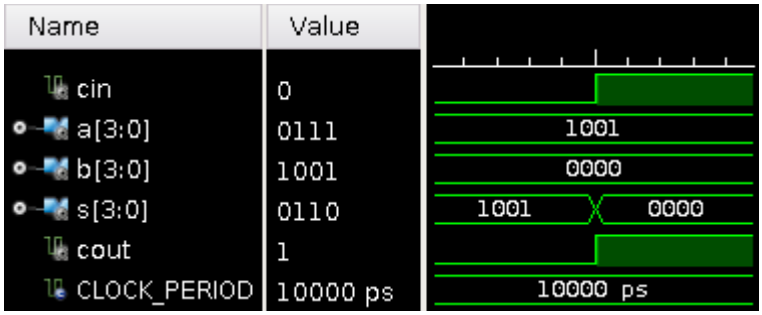
The following is the testbench file for the implementation of the simulation:

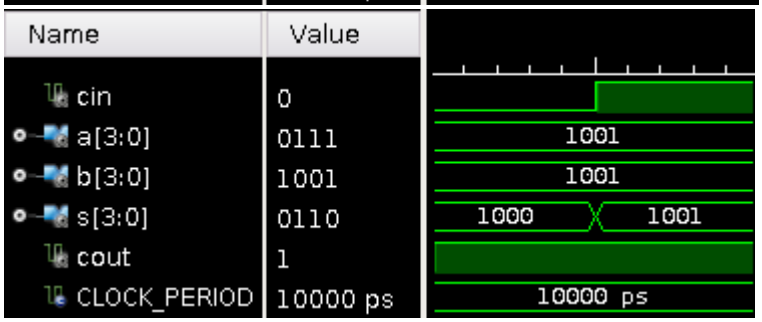
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bcd_full_adder_tb is
end bcd_full_adder_tb;

architecture bench of bcd_full_adder_tb is

  component bcd_full_adder is port (
    cin: in std_logic;
    a: in std_logic_vector(3 downto 0);
    b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    cout: out std_logic
  );
  end component;

  signal cin: std_logic := '0';
  signal a: std_logic_vector(3 downto 0) := "0000";
  signal b: std_logic_vector(3 downto 0) := "0000";
  signal s: std_logic_vector(3 downto 0);
  signal cout: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: bcd_full_adder port map (
    cin => cin,
    a => a,
    b => b,
    s  => s,
    cout  => cout
  );

  stimulus: process
  begin
    for i in 0 to 9 loop
      a <= std_logic_vector(to_unsigned(i, 4));
```

```
        for j in 0 to 9 loop
            b <= std_logic_vector(to_unsigned(j, 4));
            cin <= '0';
            wait for CLOCK_PERIOD;
            cin <= '1';
            wait for CLOCK_PERIOD;
        end loop;
    end loop;
    cin <= '0';
    a <= (others => '0');
    b <= (others => '0');
    wait;
  end process;

end bench;
```

Here are some snapshots of the simulation:







The critical path in BCD_FullAdder is the path from the 4-bit input to the cout output (7.186ns). Depending on the simple full adder, we expect a similar delay in the couts and s [3] (the output that is most delayed), only that the BCD adder, unlike the simple one, is a little longer delayed by the inmate. This is due to the complex circuits that are "hidden" inside each of its building units.

Topic 5)

The 4-digit BCD adder consists of 4 BCD adders of the previous query and is, in practice, analogous to the 4-digit adder of the third query. The inputs of the circuit, A and B, are 15-bit vectors, but for the implementation of the subject, each four-bit is a decimal digit, so, in essence, A and B are 4-digit numbers.

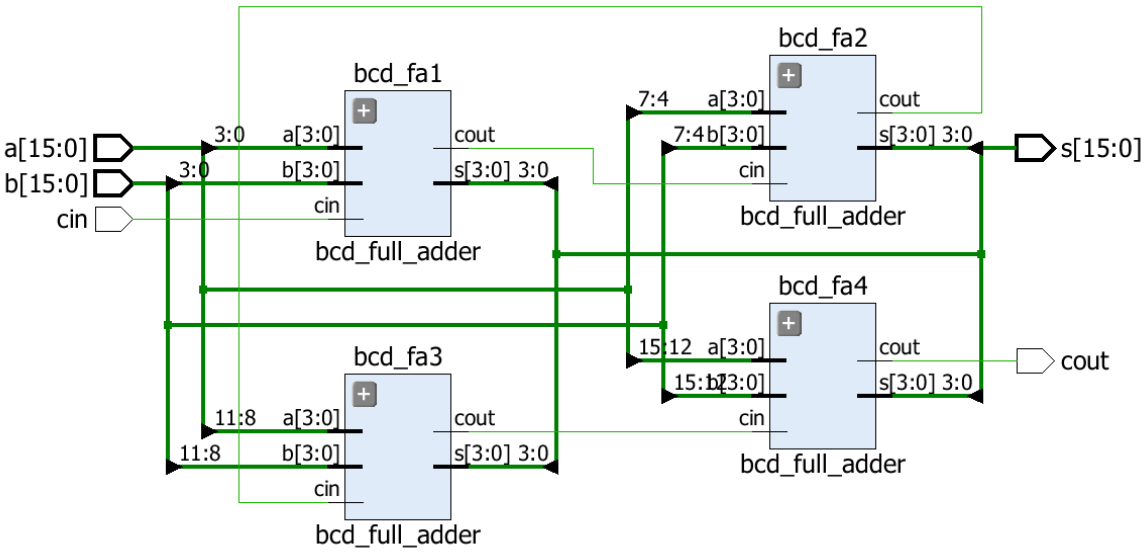
The following is the code of structural architecture:

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_parallel_adder is port(
   cin: in std_logic;
   a: in std_logic_vector(15 downto 0);
   b: in std_logic_vector(15 downto 0);
   s: out std_logic_vector(15 downto 0);
   cout: out std_logic
```

```
);
end bcd_parallel_adder;

architecture structural of bcd_parallel_adder is

  component bcd_full_adder is port(
     cin: in std_logic;
     a: in std_logic_vector(3 downto 0);
     b: in std_logic_vector(3 downto 0);
     s: out std_logic_vector(3 downto 0);
     cout: out std_logic
  );
  end component;

  signal bcd_fa1_s: std_logic_vector(3 downto 0);
  signal bcd_fa1_cout: std_logic;
  signal bcd_fa2_s: std_logic_vector(3 downto 0);
  signal bcd_fa2_cout: std_logic;
  signal bcd_fa3_s: std_logic_vector(3 downto 0);
  signal bcd_fa3_cout: std_logic;
  signal bcd_fa4_s: std_logic_vector(3 downto 0);

begin

  bcd_fa1: bcd_full_adder port map(
     cin => cin,
     a => a(3)&a(2)&a(1)&a(0),
     b => b(3)&b(2)&b(1)&b(0),
     s => bcd_fa1_s,
     cout => bcd_fa1_cout
  );

  bcd_fa2: bcd_full_adder port map(
     cin => bcd_fa1_cout,
     a => a(7)&a(6)&a(5)&a(4),
     b => b(7)&b(6)&b(5)&b(4),
     s => bcd_fa2_s,
     cout => bcd_fa2_cout
  );
```

```
  bcd_fa3: bcd_full_adder port map(
     cin => bcd_fa2_cout,
     a => a(11)&a(10)&a(9)&a(8),
     b => b(11)&b(10)&b(9)&b(8),
     s => bcd_fa3_s,
     cout => bcd_fa3_cout
  );

  bcd_fa4: bcd_full_adder port map(
     cin => bcd_fa3_cout,
     a => a(15)&a(14)&a(13)&a(12),
     b => b(15)&b(14)&b(13)&b(12),
     s => bcd_fa4_s,
     cout => cout
  );

  s <= bcd_fa4_s&bcd_fa3_s&bcd_fa2_s&bcd_fa1_s;
end structural;
```

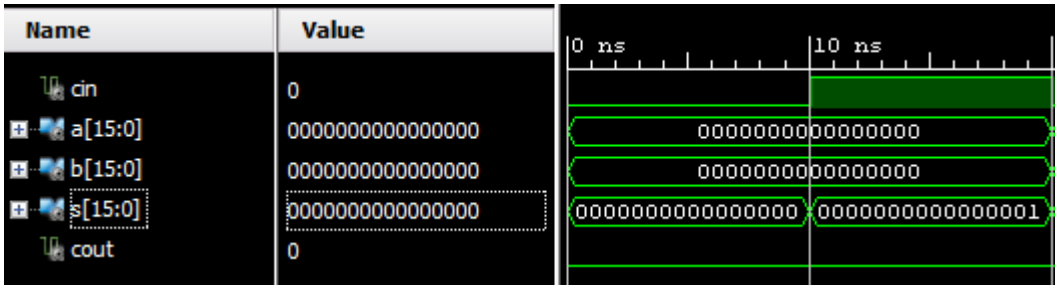
And then the schematic RTL:

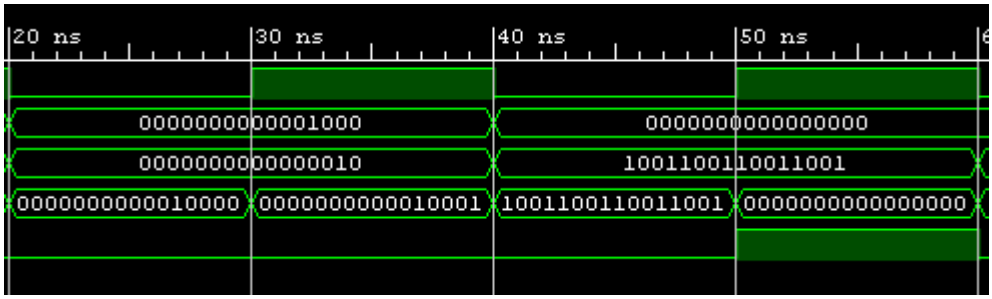


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bcd_parallel_adder_tb is
end bcd_parallel_adder_tb;

architecture bench of bcd_parallel_adder_tb is

  component bcd_parallel_adder is port (
    cin: in std_logic;
    a: in std_logic_vector(15 downto 0);
    b: in std_logic_vector(15 downto 0);
    s: out std_logic_vector(15 downto 0);
    cout: out std_logic
  );
  end component;

  signal cin: std_logic := '0';
  signal a: std_logic_vector(15 downto 0) := (others => '0');
  signal b: std_logic_vector(15 downto 0) := (others => '0');
  signal s: std_logic_vector(15 downto 0);
  signal cout: std_logic;

  constant CLOCK_PERIOD: time := 10 ns;

begin

  test: bcd_parallel_adder port map (
    cin => cin,
    a => a,
    b => b,
    s  => s,
    cout  => cout
  );

  stimulus: process
  begin
    a <= "0000"&"0000"&"0000"&"0000";
    b <= "0000"&"0000"&"0000"&"0000";
```

```
    cin <= '0';
    wait for CLOCK_PERIOD;
    cin <= '1';
    wait for CLOCK_PERIOD;

    a <= "0000"&"0000"&"0000"&"1000";
    b <= "0000"&"0000"&"0000"&"0010";
    cin <= '0';
    wait for CLOCK_PERIOD;
    cin <= '1';
    wait for CLOCK_PERIOD;

    a <= "0000"&"0000"&"0000"&"0000";
    b <= "1001"&"1001"&"1001"&"1001";
    cin <= '0';
    wait for CLOCK_PERIOD;
    cin <= '1';
    wait for CLOCK_PERIOD;

    a <= (others => '0');
    b <= (others => '0');
    cin <= '0';
    wait;
  end process;

end bench;
```

The following are snapshots from the relevant testbench:





Regarding the critical path, obviously the path with the longest delay is from the first digit of the inputs (eg A [3-0]) to the last of the output (s [15-12]) (15.615ns).