Digital Systems VLSI

Nanos Georgios

The purpose of this laboratory exercise was to familiarize us with the piping technique (Pipeline), through the implementation of modern computer circuits, whose different subsystems can process different subset of data in parallel. More specifically, we implemented:

- *Question 1:* a *Full Adder (FA)*
- Question 2: a 4-bit Propagation Aggregator using the Pipeline technique
- Question 3: a 4 bit Systolic Propagation Multiplier using Modern Complete Additions

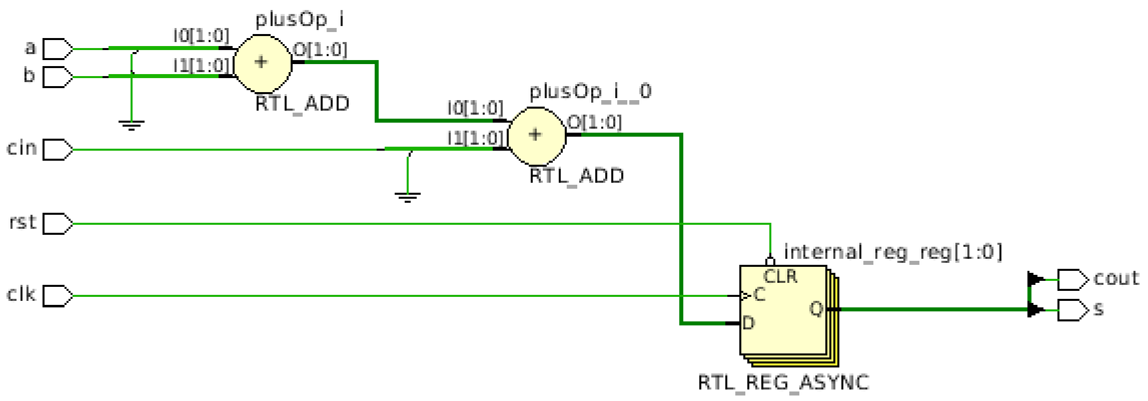
Topic 1)

Behavioral Description of the Modern Complete Addition

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


entity full_adder is port(
  a: in std_logic;
  b: in std_logic;
  cin: in std_logic;
  s: out std_logic := '0';
  cout: out std_logic := '0';
  rst : in std_logic;
  clk : in std_logic
);
end full_adder;
```

```vhdl
architecture behavioural of full_adder is

signal internal_reg : std_logic_vector(1 downto 0) := (others => '0');
begin

  -- Simple full adder that produces output on clock edge --
  process(clk, rst)
  begin
     if rst = '0' then
        internal_reg <= "00";
     elsif rising_edge(clk) then
        internal_reg <= ('0' & a) + ('0' & b) + ('0' & cin);
     end if;
  end process;
  s <= internal_reg(0);
  cout <= internal_reg(1);
end behavioural;
```

RTL schematic

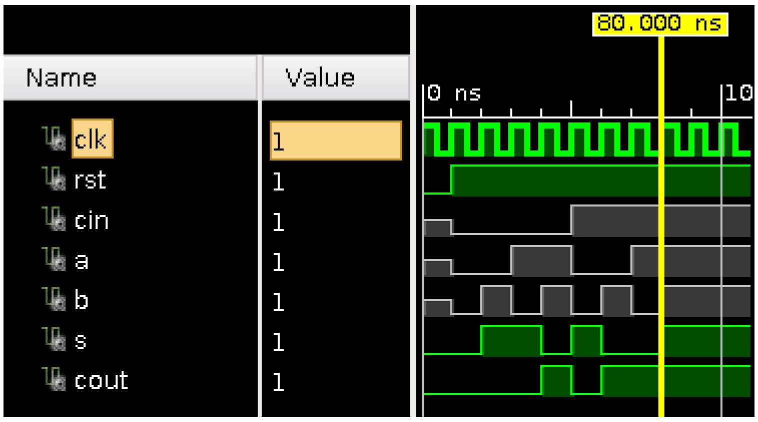


Testbench for checking the correct operation of the circuit

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

entity fulladder_tb is
```

```
end entity;

architecture bench of fulladder_tb is

 component full_adder is
   port (
       a: in std_logic;
       b: in std_logic;
       cin: in std_logic;
       s: out std_logic;
       rst : in std_logic;
       cout: out std_logic;
       clk: in std_logic
   );
 end component;

 signal a: std_logic;
 signal b: std_logic;
 signal cin: std_logic;
 signal rst : std_logic;
 signal s: std_logic;
 signal cout: std_logic;
 signal clk: std_logic;

 constant CLOCK_PERIOD : time := 10 ns;

begin
  mapping : full_adder
   port map (
     a => a,
     b => b,
     cin => cin,
     s => s,
     rst => rst,
     cout => cout,
     clk => clk
   );

 stimulus : process
 begin
 -- check disabled --
 rst <= '0';
a <= '-';
b <= '-';
cin <= '-';
wait for CLOCK_PERIOD;
 rst <= '1';
 -- test every possible value --
   for i in std_logic range '0' to '1' loop
      cin <= i;
      for j in std_logic range '0' to '1' loop
          a <= j;
          for k in std_logic range '0' to '1' loop
              b <= k;
              wait for CLOCK_PERIOD;
          end loop;
      end loop;
  end loop;
  wait;
end process;
 generate_clock : process
 begin
   clk <= '1';
   wait for CLOCK_PERIOD/2;
   clk <= '0';
   wait for CLOCK_PERIOD/2;
 end process;
end architecture;
```

Snapshots of the Simulation

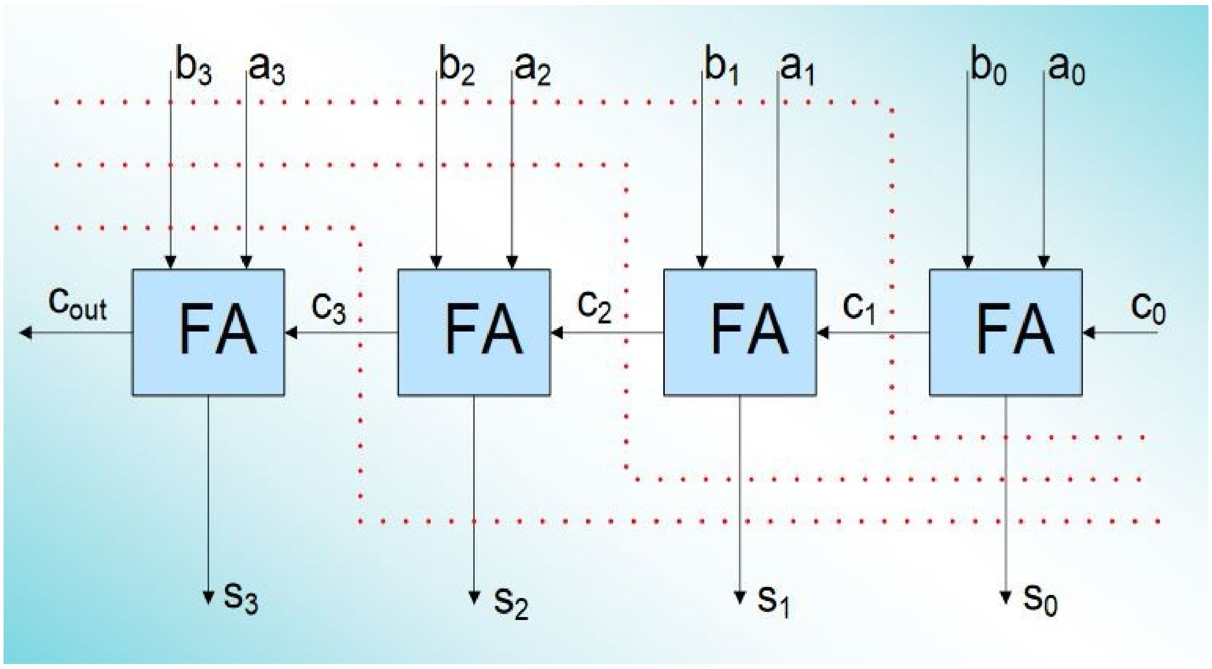


Critical Path

In the modern FA there are two paths, one for each exit. Any of these can be considered critical, as the final results are given after a specific flip flop. The critical path time lag is therefore 4,112 n.

Topic 2)

The structure that was implemented is the following, as given in the theory:



where the red, dotted lines are delays. Because the FAs are modern, the operation is partial and not instantaneous, as it would be if it were asynchronous, combinatorial logic. To guarantee the correct operation of the system, delays are added to the inputs and outputs of the adders, depending on their location. Thus, the last adder, which takes its inmate in the 3rd and gives a result in the 4th cycle, needs 3 entry delays and no output, the penultimate one that takes the inmate in the 2nd and gives an effect in the 3rd cycle needs 2 inputs and an exit coke. Finally, the respective result is given after 3 cycles (so at the beginning of the 4th cycle).

Structural Description of the Modern 4-bit Prisoner Propagation Addition

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

entity pipeline_4bit_adder is
 port(
      a : in  std_logic_vector(3 downto 0);
      b : in  std_logic_vector(3 downto 0);
      cin : in  std_logic;
      clk : in std_logic;
      rst : in std_logic;
      s : out std_logic_vector(3 downto 0);
      cout : out std_logic
     );
end pipeline_4bit_adder;

architecture structural of pipeline_4bit_adder is

 component full_adder is
   port(
       a: in std_logic;
       b: in std_logic;
       cin: in std_logic;
       s: out std_logic := '0';
       rst : in std_logic;
       cout: out std_logic := '0';
       clk : in std_logic
       );
 end component;
 component d_flipflop is
     port(
       d : in  std_logic;
       q : out  std_logic;
       clk : in std_logic;
       rst : in std_logic
      );
 end component;
 component delay_2 is
   port(
     d : in  std_logic;
     q : out  std_logic;
     clk : in std_logic;
     rst : in std_logic
     );
 end component;

 component delay_3 is
   port(
     d : in  std_logic;
     q : out  std_logic;
     clk : in std_logic;
     rst : in std_logic
     );
 end component;

 signal sum_reg : std_logic_vector(3 downto 0);
 signal carry_vec : std_logic_vector(4 downto 0);
 signal fa_s : std_logic_vector(3 downto 0);
 signal fa_a : std_logic_vector(3 downto 0);
 signal fa_b : std_logic_vector(3 downto 0);

begin
 fa_a(0) <= a(0);
 fa_b(0) <= b(0);
 carry_vec(0) <= cin;
 generate_adders: for i in 0 to 3 generate
 fa: full_adder
   port map (
     a => fa_a(i),
     b => fa_b(i),
     cin => carry_vec(i),
     cout => carry_vec(i+1),
     clk => clk,
     rst => rst,
     s => fa_s(i)
```

```
  );
end generate;

delay_s0_3 : delay_3
  port map (
    d => fa_s(0),
    q => sum_reg(0),
    clk => clk,
    rst => rst
  );

delay_s1_2 : delay_2
  port map (
    d => fa_s(1),
    q => sum_reg(1),
    clk => clk,
    rst => rst
  );

delay_s2_1 : d_flipflop
  port map (
    d => fa_s(2),
    q => sum_reg(2),
    clk => clk,
    rst => rst
  );

delay_a1_1 : d_flipflop
  port map (
    d => a(1),
    q => fa_a(1),
    clk => clk,
    rst => rst
  );

delay_a2_2 : delay_2
  port map (
    d => a(2),
    q => fa_a(2),
    clk => clk,
    rst => rst
  );

 delay_a3_3 : delay_3
  port map (
    d => a(3),
    q => fa_a(3),
    clk => clk,
    rst => rst
  );

 delay_b1_1 : d_flipflop
  port map (
    d => b(1),
    q => fa_b(1),
    clk => clk,
    rst => rst
  );

 delay_b2_2 : delay_2
  port map (
    d => b(2),
    q => fa_b(2),
    clk => clk,
    rst => rst
  );

  delay_b3_3 : delay_3
  port map (
    d => b(3),
    q => fa_b(3),
    clk => clk,
    rst => rst
  );

 sum_reg(3) <= fa_s(3);
 s <= sum_reg;
 cout <= carry_vec(4);

end architecture ; -- arch
```

The following auxiliary structures were used for the above description:

- D Flip Flop (1 cycle delay)

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

entity d_flipflop is
 port(
     d : in  std_logic;
     q : out  std_logic := '0';
     clk : in std_logic;
     rst : in std_logic
   );
end entity;

architecture behavioural of d_flipflop is
begin
   process(clk, rst)
   begin
       if rst = '0' then
           q <= '0';
       elsif clk' event and clk = '1' then
           q <= d;
       end if;
   end process;
end behavioural;
```

- 2 cycle delay (2 d flip flop combination)

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

entity delay_2 is
   port(
       d : in  std_logic;
       q : out  std_logic;
```

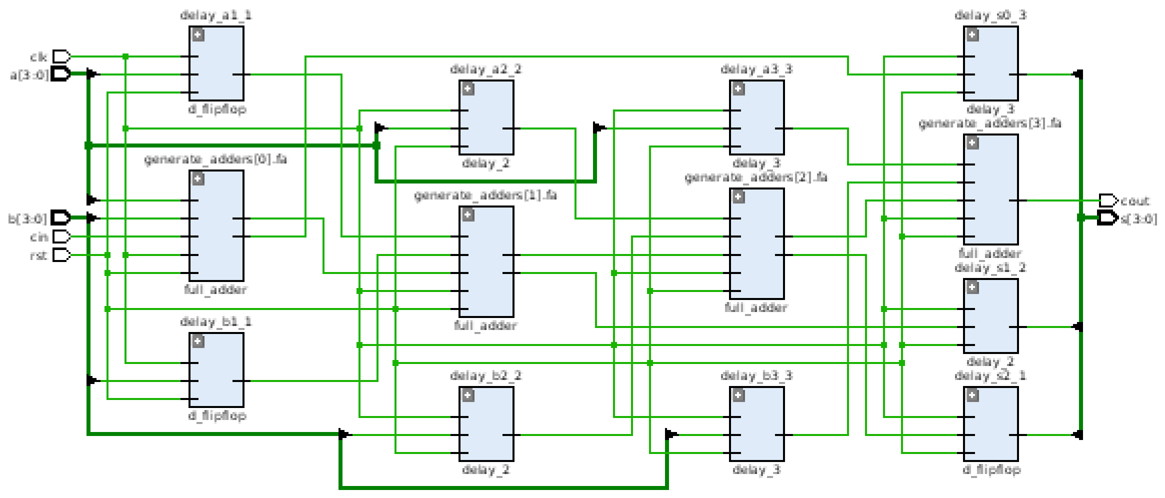
```
      clk : in std_logic;
      rst : in std_logic
     );
end delay_2;

architecture structural of delay_2 is

component d_flipflop is
  port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
  );
end component;

signal buffer_bit : std_logic;

 begin
  delay1 : d_flipflop
  port map (
      d => d,
      q => buffer_bit,
      clk => clk,
      rst => rst
  );

  delay2 : d_flipflop
  port map (
      d => buffer_bit,
      q => q,
      clk => clk,
      rst => rst
  );

end architecture;
```

- 3 cycle delay (combination of a d flip flop and a 2 cycle delay)

```
library ieee;
```

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

entity delay_3 is
  port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
  );
end delay_3;

architecture structural of delay_3 is

  component d_flipflop is
      port(
        d : in  std_logic;
        q : out  std_logic;
        clk : in std_logic;
        rst : in std_logic
      );
  end component;

  component delay_2 is
      port(
        d : in  std_logic;
        q : out  std_logic;
        clk : in std_logic;
        rst : in std_logic
      );
  end component;

signal buffer_bit : std_logic;

 begin
  delay2 : delay_2
  port map (
      d => d,
      q => buffer_bit,
      clk => clk,
      rst => rst
  );

  delay1 : d_flipflop
  port map (
      d => buffer_bit,
      q => q,
      clk => clk,
      rst => rst
  );

end architecture;
```

RTL schematic

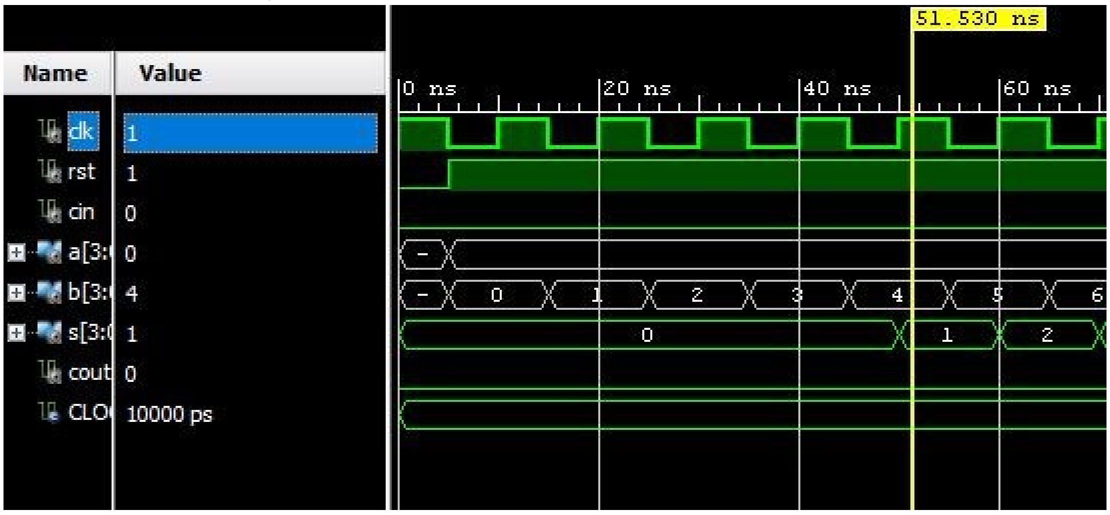


Testbench for checking the correct operation of the circuit

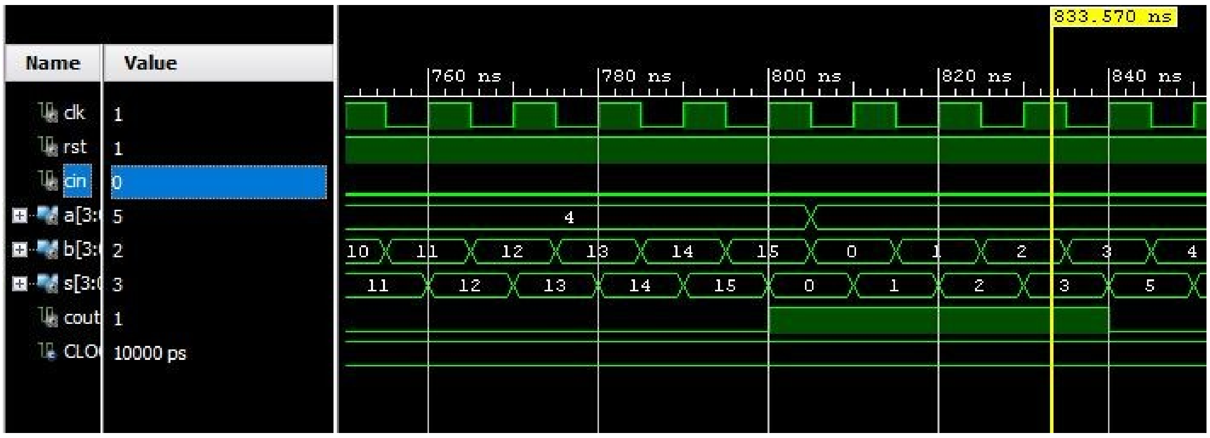
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pipeline_4bit_adder_tb is
end entity;

architecture bench of pipeline_4bit_adder_tb is

 component pipeline_4bit_adder is
   port (
       a : in  std_logic_vector(3 downto 0);
       b : in  std_logic_vector(3 downto 0);
       cin : in  std_logic;
       clk : in std_logic;
       rst : in std_logic;
       s : out std_logic_vector(3 downto 0);
       cout : out std_logic
   );
 end component;

 component d_flipflop is
   port(
        d : in  std_logic;
        q : out  std_logic;
        clk : in std_logic;
        rst : in std_logic
   );
 end component;


 signal a: std_logic_vector(3 downto 0) := (others => '0');
 signal b: std_logic_vector(3 downto 0) := (others => '0');
 signal cin: std_logic;
 signal rst : std_logic;
 signal s: std_logic_vector(3 downto 0);
signal cout: std_logic;
signal clk: std_logic;

constant CLOCK_PERIOD : time := 10 ns;


begin

 mapping : pipeline_4bit_adder
  port map (
    a => a,
    b => b,
    cin => cin,
    s => s,
    rst => rst,
    cout => cout,
    clk => clk
  );

stimulus : process
begin
 rst <= '0';
a <= "----";
b <= "----";
cin <= '0';
wait for CLOCK_PERIOD/2;
 rst <= '1';

for i in std_logic range '0' to '1' loop
  cin <= i;
  for j in 0 to 15 loop
    a <= std_logic_vector(to_unsigned(j, 4));
    for k in 0 to 15 loop
      b <= std_logic_vector(to_unsigned(k, 4));
      wait for CLOCK_PERIOD;
    end loop;
  end loop;
end loop;
cin <= '0';
a <= "0000";
```

```
b <= "0000";
wait;
end process;
 generate_clock : process
 begin
   clk <= '1';
   wait for CLOCK_PERIOD/2;
   clk <= '0';
   wait for CLOCK_PERIOD/2;
 end process;
end architecture;
```

Snapshots of the Simulation This shows the delay previously described, which is caused by delays. Excluding the first cycle used to highlight the rst function, the result of the first operation (a = 0000, b = 0000) comes out in the 4th cycle, while the result of the second operation (a = 0000, b = 0001) comes out in 5th cycle, as noted.



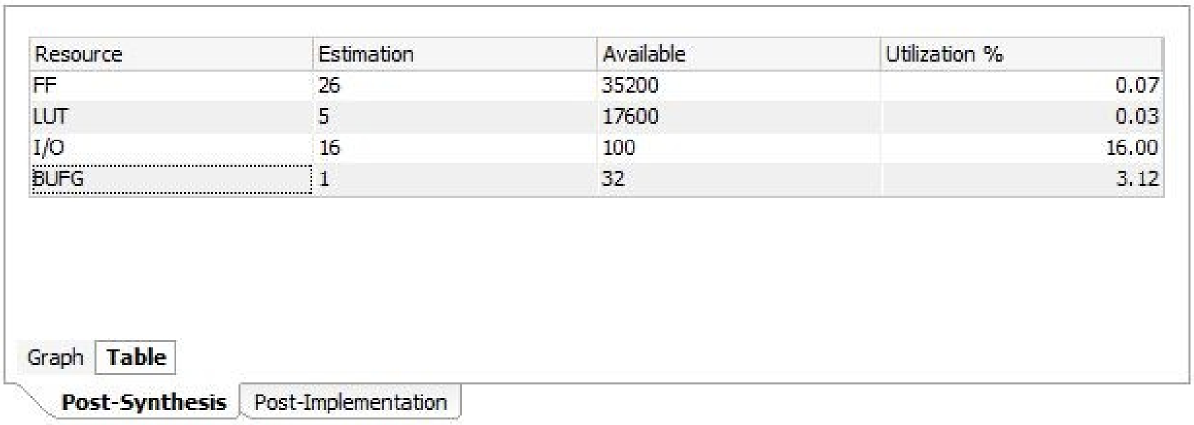
Below it appears that in each clock cycle, after the initial delay, we have the correct output for the input we had 4 cycles before. Thus, in this case, the result is 19, since, before 4 cycles the input was a = 4, b = 15.



Therefore, the response delay of the circuit to an input is equal to: Tlatency = 3 · Tcycle
Critical Path / Resources

In this implementation there are 4 equivalent critical paths, which correspond to the outputs of the system, ie they start from the inputs of the 4th adder and the inputs of the last delays of the first 3 adders and end at the outputs, cout and s [0], s [1], s [2], s [3]. It seems, thus, that the result of the operation comes out at the same time and in a time equal to 4,112 ns, just like for 1 FA. Resources after synthesizing a parallel modern adder:

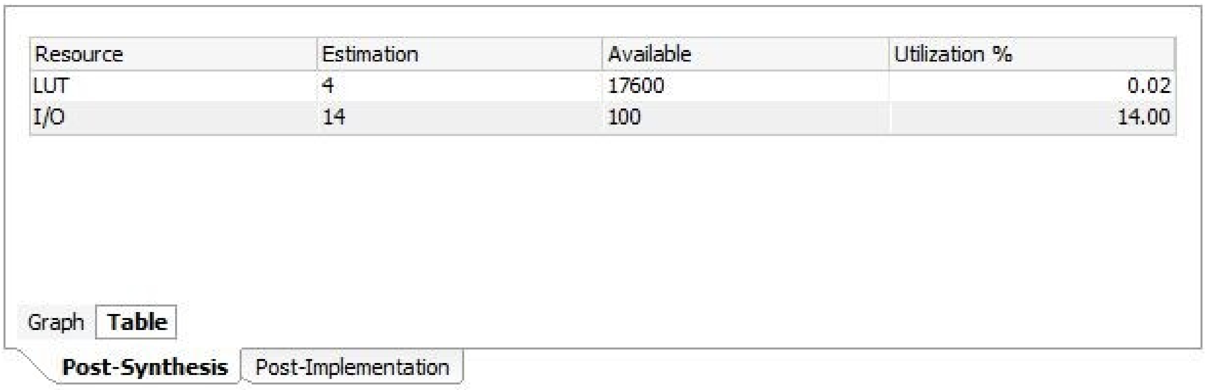
In relation to the parallel adder of laboratory exercise 2, we have an improvement (excluding the initial delay), since in that case the critical path had a time of 6,920ns. This, of course, is to the detriment of the resources, since in the case of the modern adder the above units were used to achieve the delays, which translate into flip flops, which were missing from the implementation of the asynchronous adder, as shown below.
Resources after synthesizing a parallel modern adder:



Resource	Estimation	Available	Utilization %
FF	26	35200	0.07
LUT	5	17600	0.03
I/O	16	100	16.00
BUFG	1	32	3.12

Graph
Table
Post-Synthesis
Post-Implementation

Resources after synthesizing a parallel combiner:



Resource	Estimation	Available	Utilization %
LUT	4	17600	0.02
I/O	14	100	14.00

Graph
Table
Post-Synthesis
Post-Implementation

Topic 3)
Structural Description of the 4-bit Systolic Prisoner Propagation Multiplier

```
library ieee;
```

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

entity pipeline_4bit_mult is
  port(
     a : in  std_logic_vector(3 downto 0);
     b : in  std_logic_vector(3 downto 0);
     clk : in std_logic;
     rst : in std_logic;
     p : out std_logic_vector(7 downto 0)
  );
end pipeline_4bit_mult;

architecture structural of pipeline_4bit_mult is

  component full_adder_star is
     port(
        a_in : in  std_logic;
         a_out: out std_logic;
         b_in : in  std_logic;
         b_out : out std_logic;
         sin : in std_logic;
         cin : in  std_logic;
         clk : in std_logic;
         rst : in std_logic;
         sout : out std_logic;
         cout : out std_logic
     );
  end component;

  component d_flipflop is
     port(
        d : in  std_logic;
        q : out  std_logic;
        clk : in std_logic;
        rst : in std_logic
     );
  end component;
```

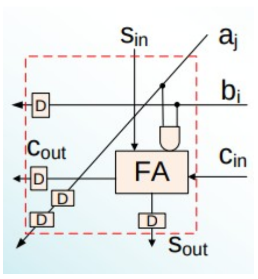
```vhdl
  component delay_3 is
    port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
    );
  end component;

  component delay_2 is
    port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
    );
  end component;

-- a input signal transmit vectors --
type a_array is array(4 downto 0) of std_logic_vector(4 downto 0);
signal a_signals : a_array := (others => (others => 'X'));

-- b input signal transmit vectors --
type b_array is array(4 downto 0) of std_logic_vector(4 downto 0);
signal b_signals : b_array := (others => (others => 'X'));

-- carries of each level --
type cout_array is array(3 downto 0) of std_logic_vector(4 downto 0);
signal cout_signals : cout_array := (others => (others => '0'));

-- output of each level (cout & s) --
-- last level is shorter as the last to values are put directly as output p --
type s_array is array(4 downto 0) of std_logic_vector(4 downto 0);
signal s_signals : s_array := (others => (others => '0'));



signal buffered_output1 : std_logic;
signal buffered_output2_vec : std_logic_vector(1 downto 0);
signal buffered_output3_vec : std_logic_vector(2 downto 0);
signal b2_delayed_2 : std_logic;
signal b3_delayed_3 : std_logic;


begin

  a_signals(0)(0) <= a(0);
  b_signals(0)(0) <= b(0);

  g_loop : for i in 0 to 3 generate
     generate_fa : for j in 0 to 3 generate
     fa : full_adder_star
        port map (
           a_in => a_signals(i)(j),
           a_out => a_signals(i+1)(j),
           b_in => b_signals(i)(j),
           b_out => b_signals(i)(j + 1),
           sin => s_signals(i)(j+1),
           sout => s_signals(i+1)(j),
           cin => cout_signals(i)(j),
           cout => cout_signals(i)(j+1),
           clk => clk,
           rst => rst
        );
     end generate;
  end generate g_loop;

  generate_delay : for i in 0 to 2 generate
     delay1 : d_flipflop
        port map (
           d => cout_signals(i)(4),
           q => s_signals(i+1)(4),
           clk => clk,
           rst => rst
        );

  end generate;

  delay_p0_1 : delay_2
  port map (
    d => s_signals(1)(0),
    q => buffered_output1,
    clk => clk,
    rst => rst
);

delay_p0_2 : delay_2
port map (
    d => buffered_output1,
    q => buffered_output2_vec(0),
    clk => clk,
    rst => rst
);

delay_p1_2 : delay_2
port map (
    d => s_signals(2)(0),
    q => buffered_output2_vec(1),
    clk => clk,
    rst => rst
);

delay_p0_3 : delay_2
port map (
    d => buffered_output2_vec(0),
    q => buffered_output3_vec(0),
    clk => clk,
    rst => rst
);

delay_p1_3 : delay_2
port map (
    d => buffered_output2_vec(1),
    q => buffered_output3_vec(1),
    clk => clk,
    rst => rst
);

delay_p2_3 : delay_2
port map (
    d => s_signals(3)(0),
    q => buffered_output3_vec(2),
    clk => clk,
    rst => rst
);

delay_p0_4 : delay_3
port map (
    d => buffered_output3_vec(0),
    q => p(0),
    clk => clk,
    rst => rst
);

delay_p1_4 : delay_3
port map (
    d => buffered_output3_vec(1),
    q => p(1),
    clk => clk,
    rst => rst
);

delay_p2_4 : delay_3
port map (
    d => buffered_output3_vec(2),
    q => p(2),
    clk => clk,
    rst => rst
);

delay_p3_4 : delay_3
port map (
    d => s_signals(4)(0),
    q => p(3),
    clk => clk,
    rst => rst
);

delay_p4_4 : delay_2
port map (
    d => s_signals(4)(1),
    q => p(4),
    clk => clk,
    rst => rst
);


delay_p5_4 : d_flipflop
port map (
    d => s_signals(4)(2),
    q => p(5),
    clk => clk,
    rst => rst
);

delay_b1_2 : delay_2
port map (
    d => b(1),
    q => b_signals(1)(0),
    clk => clk,
    rst => rst
);

delay_b2_2 : delay_2
port map (
    d => b(2),
    q => b2_delayed_2,
    clk => clk,
    rst => rst
);

delay_b2_4 : delay_2
port map(
    d => b2_delayed_2,
    q => b_signals(2)(0),
    clk => clk,
    rst => rst
);

delay_b3_3 : delay_3
```

```
port map (
    d => b(3),
    q => b3_delayed_3,
    clk => clk,
    rst => rst
);

delay_b3_6 : delay_3
port map(
    d => b3_delayed_3,
    q => b_signals(3)(0),
    clk => clk,
    rst => rst
);

delay_a1_1 : d_flipflop
port map(
    d => a(1),
    q => a_signals(0)(1),
    clk => clk,
    rst => rst
);


delay_a2_2 : delay_2
port map(
    d => a(2),
    q => a_signals(0)(2),
    clk => clk,
    rst => rst
);


delay_a3_3 : delay_3
port map(
    d => a(3),
    q => a_signals(0)(3),
    clk => clk,
    rst => rst
);
```

```
  p(6) <= s_signals(4)(3);
  P(7) <= cout_signals(3)(4);

end architecture ;
```

The following auxiliary structures were used for the above description:

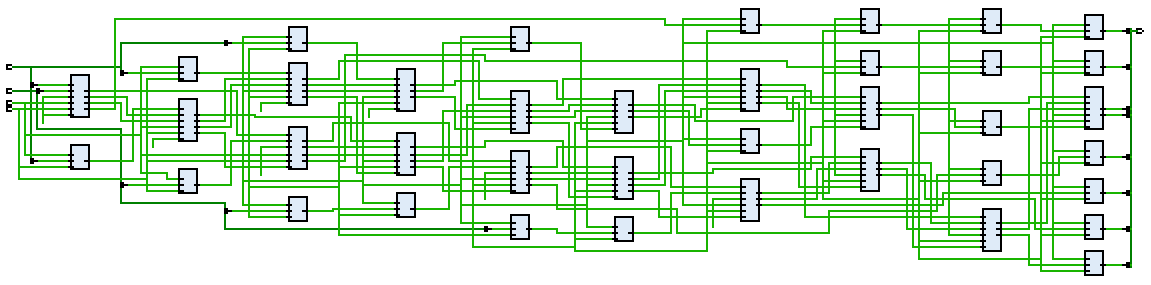
- ***D Flip Flop***
- delay of 2 cycles
- delay of 3 cycles
- *the adder of the following figure, to add the 2 bit product (aj, bi) with one bit of the sum of the previous multiplication step (sin)*



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

entity full_adder_star is
 port(
     a_in : in  std_logic;
     a_out: out std_logic;
     b_in : in  std_logic;
     b_out : out std_logic;
     sin : in std_logic;
     cin : in  std_logic;
     clk : in std_logic;
     rst : in std_logic;
     sout : out std_logic;
     cout : out std_logic
 );
end full_adder_star;
```

```
architecture structural of full_adder_star is

 component full_adder is
   port(
       a: in std_logic;
       b: in std_logic;
       cin: in std_logic;
       s: out std_logic := '0';
       rst : in std_logic;
       cout: out std_logic := '0';
       clk : in std_logic
   );
 end component;
 component d_flipflop is
   port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
   );
 end component;

 component delay_2 is
   port(
      d : in  std_logic;
      q : out  std_logic;
      clk : in std_logic;
      rst : in std_logic
   );
 end component;


signal input : std_logic;
 begin
  input <= a_in and b_in;
  adder : full_adder
   port map (
     a => sin,
     b => input,
     cin => cin,
    cout => cout,
    rst => rst,
    clk => clk,
    s => sout
  );

  b_delay : d_flipflop
  port map (
    d => b_in,
    q => b_out,
    rst => rst,
    clk => clk
  );



  a_delay : delay_2
  port map (
    d => a_in,
    q => a_out,
    rst => rst,
    clk => clk
  );

end architecture;
```

RTL schematic

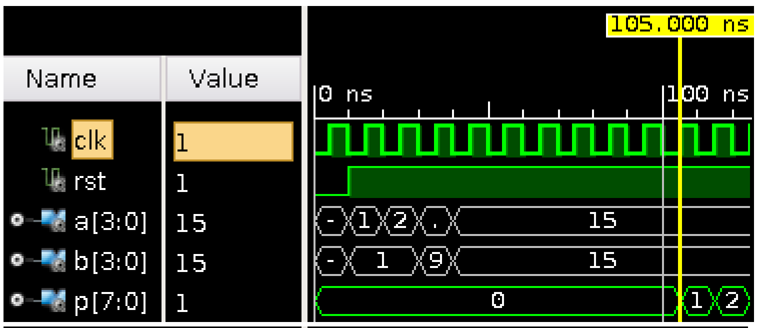


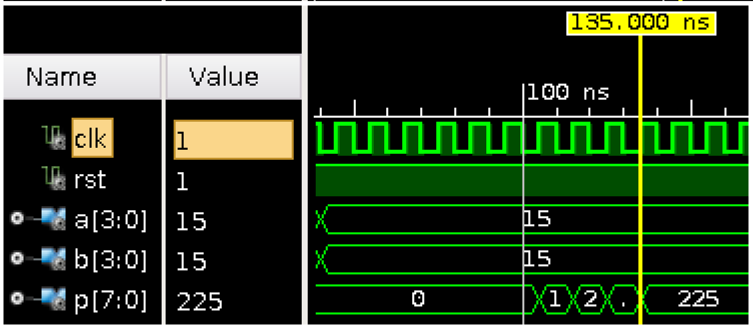
Testbench for checking the correct operation of the circuit

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

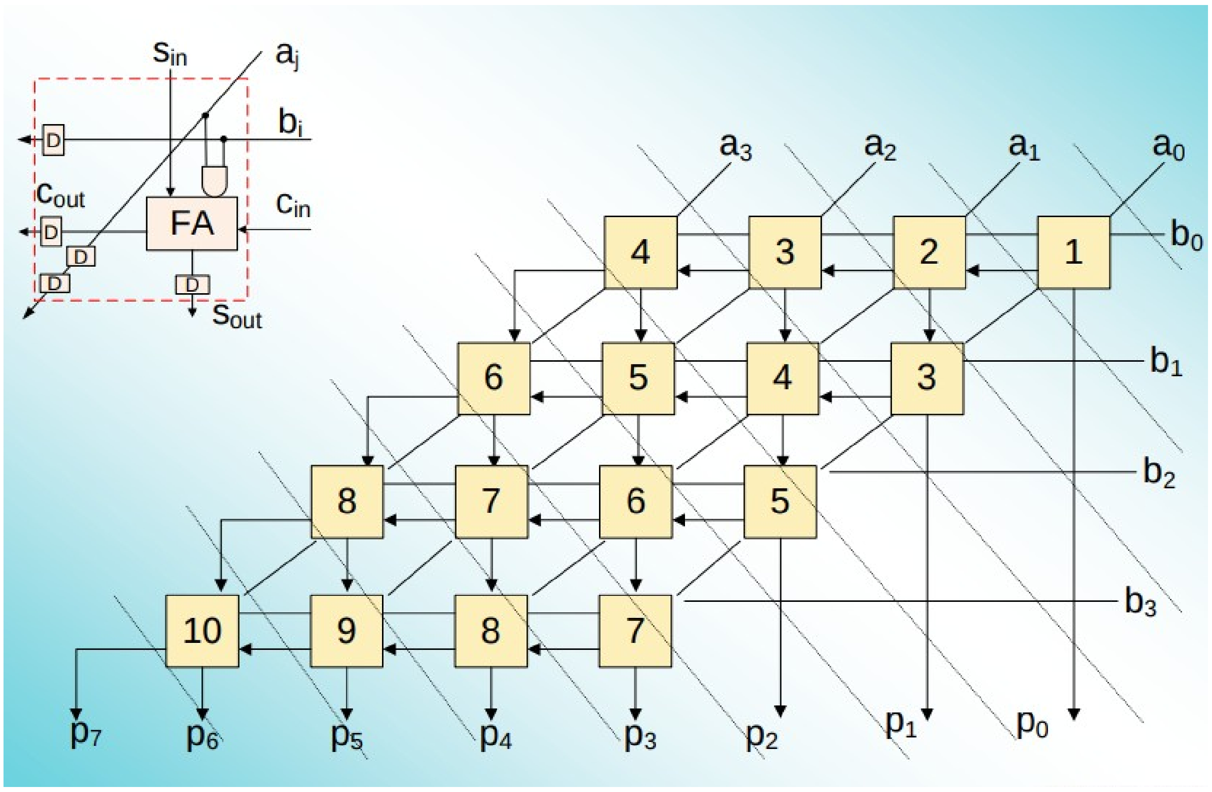
```vhdl
use ieee.numeric_std.all;

entity pipeline_4bit_mult_tb is
end entity;

architecture bench of pipeline_4bit_mult_tb is

 component pipeline_4bit_mult is
   port (
     a : in  std_logic_vector(3 downto 0);
     b : in  std_logic_vector(3 downto 0);
     clk : in std_logic;
     rst : in std_logic;
     p : out std_logic_vector(7 downto 0)
   );
 end component;

 component full_adder_star is
   port(
     a_in : in  std_logic;
     a_out: out std_logic;
     b_in : in  std_logic;
     b_out : out std_logic;
     sin : in std_logic;
     cin : in  std_logic;
     clk : in std_logic;
     rst : in std_logic;
     sout : out std_logic;
     cout : out std_logic
   );
 end component;

 component d_flipflop is
   port(
     d : in  std_logic;
     q : out  std_logic;
     clk : in std_logic;
     rst : in std_logic
   );
 end component;
 component delay_3 is
   port(
     d : in  std_logic;
     q : out  std_logic;
     clk : in std_logic;
     rst : in std_logic
   );
 end component;

 component delay_2 is
   port(
     d : in  std_logic;
     q : out  std_logic;
     clk : in std_logic;
     rst : in std_logic
   );
 end component;

 signal a: std_logic_vector(3 downto 0) := (others => '0');
 signal b: std_logic_vector(3 downto 0) := (others => '0');
 signal rst : std_logic;
 signal p: std_logic_vector(7 downto 0);
 signal clk: std_logic;

 constant CLOCK_PERIOD : time := 10 ns;

begin

   mapping : pipeline_4bit_mult
   port map (
     a => a,
     b => b,
     p => p,
     rst => rst,
     clk => clk
   );

 stimulus : process
 begin
 rst <= '0';
 a <= "----";
 b <= "----";
 wait for CLOCK_PERIOD;
 rst <= '1';

--    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));
--        end loop;
--    end loop;

   a <= "0001";
   b <= "0001";
   wait for CLOCK_PERIOD;

   a <= "0010";
   b <= "0001";
   wait for CLOCK_PERIOD;
   a <= "1010";
   b <= "1001";
   wait for CLOCK_PERIOD;

   a <= "1111";
   b <= "1111";
   wait for CLOCK_PERIOD;

   wait;
 end process;
   generate_clock : process
   begin
     clk <= '0';
     wait for CLOCK_PERIOD/2;
     clk <= '1';
     wait for CLOCK_PERIOD/2;
   end process;
end architecture;
```

Simulation snapshots





We observe that the output of the circuit corresponding to a pair of 4 bit inputs (a, b) is generated after 9 cycles (at the beginning of the 10th cycle) of the clock counting from the moment that the pair of these inputs is detected at a positive edge of watch. This is due to the way in which parallel computations are applied to the circuit. Also, after the initial delay, we get a result after each cycle, a logical consequence of using a pipeline. The circuit implemented for this query is shown in the following figure:



The response delay of the circuit, then, to an input is equal to: Tlatency = 9 · Tcycle

Critical Path

In this implementation we have 8 equivalent critical paths, as well as our outputs, since we get the results at the same time. Paths have, for exits p [0] to p [5], starting from the previous delay, while paths for exits p [6] and p [7] start at the entrance of the last FA *. The time, as before, is 4,112 ns, since the critical paths are essentially the same (they cross either a delay or an FA).

In conclusion, in a pipeline, although we have a large time latency from the change of inputs to the calculation of the new result (in relation to the simple computational circuits), we have a small critical path delay, which is based only on the characteristics of the smaller structures. units (here of the FA).