Projecto de Sistemas Digitais
Digital Systems Design
MEEC, 2023/24, 2nd Exam

Table 1: Timing characteristics and resource utilization of logic-arithmetic components				
	Area	t_P	t_{SETUP}	t_{HOLD}
16-bit Register	10	2 ns	1 ns	1 ns
BRAM	100	2 ns	1 ns	1 ns
16-bit Adder / Subtractor	16	5 ns		
16-bit Multiplier (simple)	120	10 ns		
MUX 2:1 (16-bit data words)	16	3 ns		
MUX 4:1 (16-bit data words)	32	4 ns		
MUX 8:1 (16-bit data words)	64	5 ns		
16-bit logic operators	16	3 ns		

Note: all answers must be properly justified, without which they will not be classified.

1. **[4 val]** Consider the following sequence of operations:

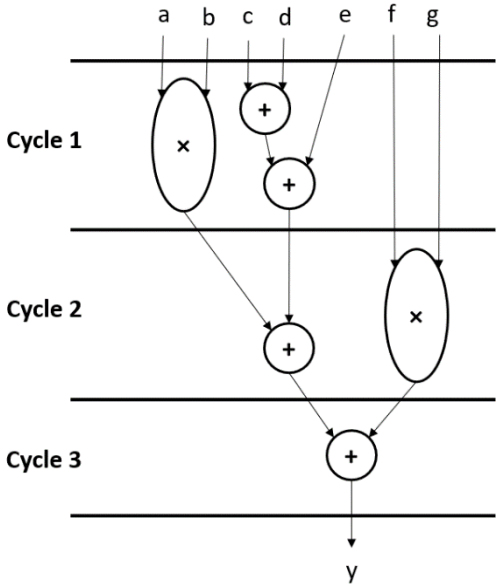
$$x_1 = a_1 \times b_1 + a_2 \times b_2 ;$$
$$x_2 = a_3 \times b_3 + a_4 \times b_4 ;$$
$$s_1 = b_5 + b_6 ;$$
$$x_3 = a_5 \times s_1 + a_6 \times s_1 ;$$
$$s_2 = x_2 + x_3 ;$$
$$y = s_2 + x_1 ;$$

Sketch the data flow graph (DFG) that corresponds **directly** to the sequence.
In the DFG, keep the sequence of operations exactly as indicated.

Define an ordered priority list using the critical path as metric.

Consider that the inputs are stored in independent registers and that the arithmetic operators available are **2 multiplier (simple)** and **2 adders**.
Consider that **all inputs, operators, buses and datapath registers are 16-bit wide** with the timing/area characteristics shown in table 1.
Obtain a list scheduling using the priority list defined and considering a **target clock period of 18 ns**.

2. Consider the schedule defined in the figure, at the right.



The circuit datapath is to be implemented with the minimum number of adders and the minimum number of multipliers required to implement this schedule.

All input values are 16-bit signed integers, stored in independent **non-shareable** registers.

All operators, buses and datapath registers are 16-bit wide, with the timing/area characteristics shown in table 1. All circuit registers must be positive edge-triggered, fully synchronous and be synchronized with a unique global clock.

a. **[4 val]** Design the circuit datapath to achieve the best **performance** (given the constraints indicated).

Indicate the number of registers and multiplexers required.
Draw the block diagram of your datapath.

Estimate the area occupied, the minimum clock period, and the latency for your design.
Justify all the bindings and sharing of resources, and explain how the performance was maximized.

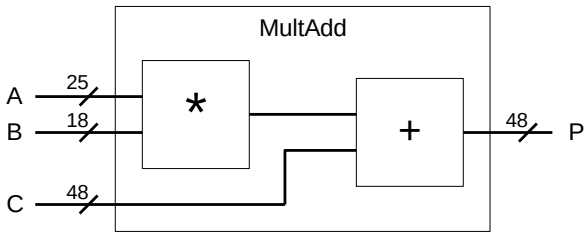
b. **[1 val]** Identify all the control signals required for the datapath designed in (a). Indicate explicitly the values of each control signal for each clock cycle.

3. Consider the MultAdd component that implements the arithmetic operation

$$P = C + A \times B$$

using a dedicated DSP48E1 component in an Artix-7 FPGA.

The operands are signed integers represented in 2's complement with the width-sizes indicated in the figure.



a) **[2 val]** Complete the VHDL specification (below) to completely specify the architecture of a combinatorial circuit to perform the calculation (X × Y) + Z with real operands represented in fixed-point, using the MultAdd component (don't care about minor syntax issues).

The operands are represented in the following fixed-point formats: X: Q16.4 , Y: Q8.8 and Z: Q20.8 .
The circuit has three outputs: the **rexact** output represents the exact result of the calculation without loss of precision or overflow problems, *the* **rtrunc** output represents the result truncated to the Q16.4 format, and the **rround** output represents the result rounded (to the nearest) to the Q16.4 representation.

```
entity exame2 is
  port ( x : in  std_logic_vector (19 downto 0);
         y : in  std_logic_vector (15 downto 0);
         z : in  std_logic_vector (27 downto 0);

         rexact : out  std_logic_vector (  ???  );

         rtrunc : out  std_logic_vector (19 downto 0);
         rround : out  std_logic_vector (19 downto 0));
end exame2;

architecture ex2 of exame2 is
  component MultAdd
    port ( a : in std_logic_vector(24 downto 0);
           b : in std_logic_vector(17 downto 0);
           c : in std_logic_vector(47 downto 0);
           p : out std_logic_vector(47 downto 0));
  end component;

  signal i1 : std_logic_vector(  ???  );
  signal i2 : std_logic_vector(  ???  );
  signal i3 : std_logic_vector(  ???  );
  signal o1 : std_logic_vector(  ???  );

begin

  MA1: MultAdd port map(a => i1, b => i2, c => i3, p => o1);

  ???




end ex2;
```

b) **[1 val]** What are the hexadecimal values output in `rexact` and `rround`, when `X <= X"00048", Y <= X"FE00", Z <= X"0000A80`?

4. Consider that you want to design a circuit, in a Xilinx Artix-7 device, to calculate the mean value of a set of 512 elements $\boldsymbol{a_i}$, as fast as possible:

$$m = \frac{1}{512} \sum_{i=0}^{511} a_i$$

The $\boldsymbol{a_i}$ elements are 18-bit integers stored in one RAMB36E1 block and the mean value $\boldsymbol{m}$ is a real number represented in fixed-point format to be stored in one independent register.

All circuit registers must be *positive edge-triggered* and be synchronized with a unique global clock.

a) **[3 val]** Select the BRAM organization that you consider most convenient to maximize the performance of the circuit.

   Estimate the best throughput achievable with the memory organization specified.
   Indicate how to organize the arithmetic operations to achieve this throughput.
   Sketch the block diagram of the arithmetic unit.

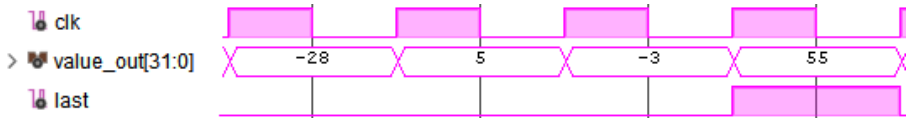
   Estimate the clock period to be used, using the timing characteristics shown in table 1 (consider the timing characteristics of the most approximately sized operator).
   Also estimate the total execution time of calculation.

b) **[1 val]** Define the dimensions of all operands, buses, and registers in the arithmetic unit to avoid the existence of overflows and to avoid any loss of precision in the computations.
   Indicate the fixed-point representation format of the output mean value $\boldsymbol{m}$.

5. Consider a circuit that downsamples an input array with 2N int<32> elements to produce an output stream with N int<32> elements, one output element per clock cycle, such that each output element is the average of a pair of consecutive array elements. The output stream includes a ***last*** signal that indicates the end of the stream.

   The figure below shows the output stream for an 8-element array example (N = 4, *array_size* = 8).

Array Position	0	1	2	3	4	5	6	7
Array Elements	-60	4	10	0	66	-72	75	35



   The circuit consists of one address generator component, one True Dual Port 1K×32-bit BRAM that stores the array and the logic to produce the output stream, as shown in Figure 4 (in the following page).

a) **[2 val]** Complete the VHDL specification (below) of the architecture of the *address_generator* component (don't care about minor syntax issues).
   The *address_generator* generates the address for the 2 read ports of the BRAM and produces an *end_cnt* signal that indicates that the addresses correspond to the last accesses to the array.
   After generating the last addresses, the counter resets itself.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity address_generator is
  port ( clk, rst, enable: in std_logic;
         array_size: in std_logic_vector(10 downto 0);   -- size of the array
         addr1, addr2 : out std_logic_vector(9 downto 0); -- memory addresses
         end_cnt: out std_logic  -- end of count
       );
end address_generator;

architecture Behavioral of address_generator is
  signal address_counter : unsigned( ??? ); -- define range
  -- declare all other internal signals
  ???
begin

  -- counter
  process(clk)
  begin
    if(rising_edge(clk)) then

        ???

   end if;
 end process;

  -- define internal logic and connections
  ???

  -- define output logic and connections
  ???

end Behavioral;
```

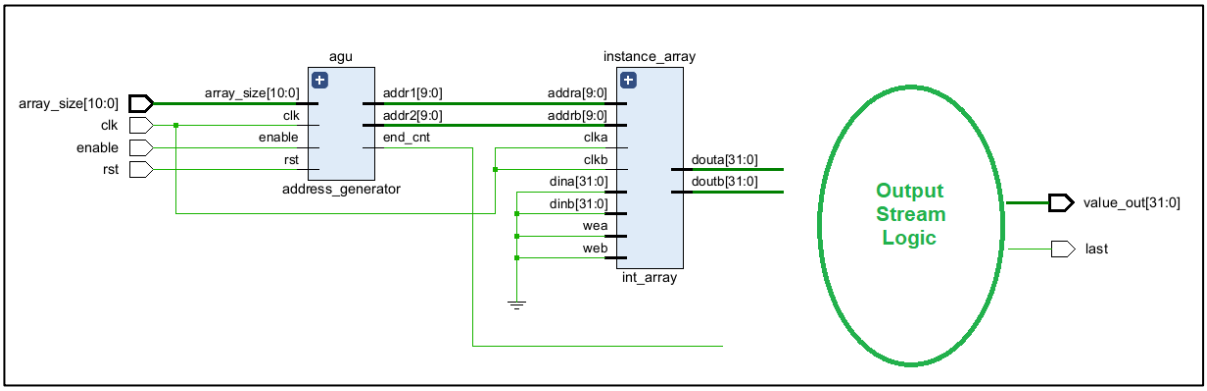


Figure 4

b) **[2 val]** Complete the VHDL specification (below) of the architecture of the *circuit* component (don't care about minor syntax issues) to include the logic and connections to produce the output stream *value_out* and *last* signals.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity circuit is
  port ( clk, rst, enable: in std_logic;
         value_out: out std_logic_vector(31 downto 0);
         array_size: in std_logic_vector(10 downto 0);
         last: out std_logic   );
end circuit;

architecture Behavioral of circuit is
  component int_array
    port ( clka, clkb : in std_logic;
           wea, web : in std_logic__vector(0 downto 0);
           addra, addrb : in std_logic_vector(9 downto 0);
           dina, dinb : in std_logic _vector(31 downto 0);
           douta, doutb : out std_logic_vector(31 downto 0);  );
   end component;
   component address_generator is
     port ( clk, rst, enable: in std_logic;
            array_size: in std_logic_vector(10 downto 0);
            addr1, addr2 : out std_logic_vector(9 downto 0);
            end_cnt: out std_logic  );
   end component;

   signal addr1, addr2 : std_logic_vector(9 downto 0);
   signal sum : signed(32 downto 0);
   signal end_cnt : std_logic;

  -- declare all other internal signals
  ???

begin
 instance_array : int_array port map (
   clka => clk, wea => "0", addra => addr1, dina=>(others=>'0'), douta => val1,
   clkb => clk, web => "0", addrb => addr2, dinb=>(others=>'0'), doutb => val2 );

 agu: address_generator port map (
   clk => clk, rst => rst, enable => enable,
   addr1 => addr1, addr2 => addr2,
   array_size => array_size, end_cnt => end_cnt);

  -- define internal logic and connections
  ???

  -- define output logic and connections
  ???

end Behavioral;
```