Personal Information

Please fill in the following information

First Name:

Last Name:

SCIPER Nr.:

You must put your name and SCIPER number on all the answer sheets where these can be written, please use the fields in the top of every second page.

Instructions

The rules of the exam are:

- You have 180 minutes, from 15:15 – 18:15, to complete as many tasks as you can.
- The exam is OPEN BOOK with access to the lab computers (as during the labs).
- You can use your own computer as well, but it is your responsibility that it works (no debugging from us of any computer issues on private computers during the exam).
- You may use Vivado tools for simulation, syntax checking, and so forth.
- You may use the internet for searching for information, **but you are not allowed under any circumstances to communicate with anyone.**
- **AI systems like chatGPT, google Bard etc. are forbidden during the exam.**

The Moodle contains a ZIP file that you can download once the examination starts. This ZIP file holds templates for VHDL code that you need to complete or edit for some of the tasks.

When handing in, please note:

- For questions where you must submit VHDL code, please upload files prefixed with your SCIPER number, i.e., named `SCIPER_taskX.vhdl` with SCIPER to be replaced with your SCIPER number and `taskX` being the task with its number.
- For other tasks, please use the red rectangular response areas to write your answer. An example of such a box is provided at the bottom of this page. It is recommended to make a sketch on your own paper before drawing in the response area if you are unsure about your answer. You can ask the exam supervisors for more paper if necessary.
- The answer sheets will be collected at the end of the examination.

EXAMPLE RESPONSE AREA

Example response area.

Part 1: Erroneous Designs and VHDL Code

This part of the exam deals with examples of VHDL code written in ways that do not match a specification or where the VHDL code has numerous errors that do not allow synthesis.

Task 1.1: Computational block with unexpected behaviour

Listing 1 shows sequential pseudocode of an algorithm specification that an intern at the company BIGCHIP was asked to implement as a synchronous circuit. For the algorithm specification, the updates shown in Listing 1 have to happen immediately to all the variables. This means that when `A` is incremented the new value of `A` should be used to calculate `B` and the new value of `B` should be used to update `C`. For the implementation of the synchronous circuit, the circuit must produce a new and coherent set of outputs (corresponding to one execution of the algorithm in Listing 1) in each cycle.

```
1  A = A + 1;
2  B = A xor "10101010";
3  C = B - 1;
```

Listing 1: Pseudocode for the hardware implemented in Listing 2 for Task 1.1.

The VHDL code snippet in Listing 2 is an attempt to implement this code in hardware by the intern. Two signals have been added to control the operation of the block in a synchronous manner:

- `OFFxSI`: If `OFFxSI` is HIGH ('1'), the circuit should re-initialize `AxD`, `BxD`, and `CxD` no matter the value of `ONxSI`.
- `ONxSI`: If `ONxSI` is HIGH ('1') and `OFFxSI` LOW ('0') the circuit updates every cycle.

While the hardware synthesis succeeds (i.e., there are no errors hindering synthesis/implementation in Vivado), the implemented design (on the chip) does not meet the specifications above for the operations on `A, B, C` and control signals `OFFxSI` and `ONxSI`. In the following tasks, you will explain why.

```vhdl
entity task1_1 is
  port (
    CLKxCI, RSTxRI : in std_logic;
    ONxSI, OFFxSI  : in std_logic;
    OutxDO : out unsigned(24-1 downto 0)
  );
end task1_1;

architecture rtl of task1_1 is
  signal AxD, BxD, CxD : unsigned(8-1 downto 0);
begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      AxD <= (others => '0');
      BxD <= (others => '0');
      CxD <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      if (OFFxSI = '1') then
        AxD <= (others => '0');
        BxD <= (others => '0');
        CxD <= (others => '0');
      end if;
      if (ONxSI = '1') then
        AxD <= AxD + 1;
        BxD <= AxD xor "10101010";
        CxD <= BxD - 1;
      end if;
    end if;
  end process;
  OutxDO <= AxD * BxD * CxD;
end rtl;
```

Listing 2: Task 1.1 code snippet (`task1_1.vhdl`) implementing Listing 1 pseudocode.

Task 1.1a

When the circuit was tested, it was observed that when `OFFxSI = '1'` the circuit never re-initializes `AxD, BxD, CxD` to all 0 when `ONxSI = '1'`. It is only when `OFFxSI = '1'` and `ONxSI = '0'` that the circuit correctly re-initializes `AxD, BxD, CxD` to all 0. Explain why this happens, i.e., why does changing the value of `OFFxSI` have no effect when `ONxSI = '1'`.

Response area for task 1.1a.

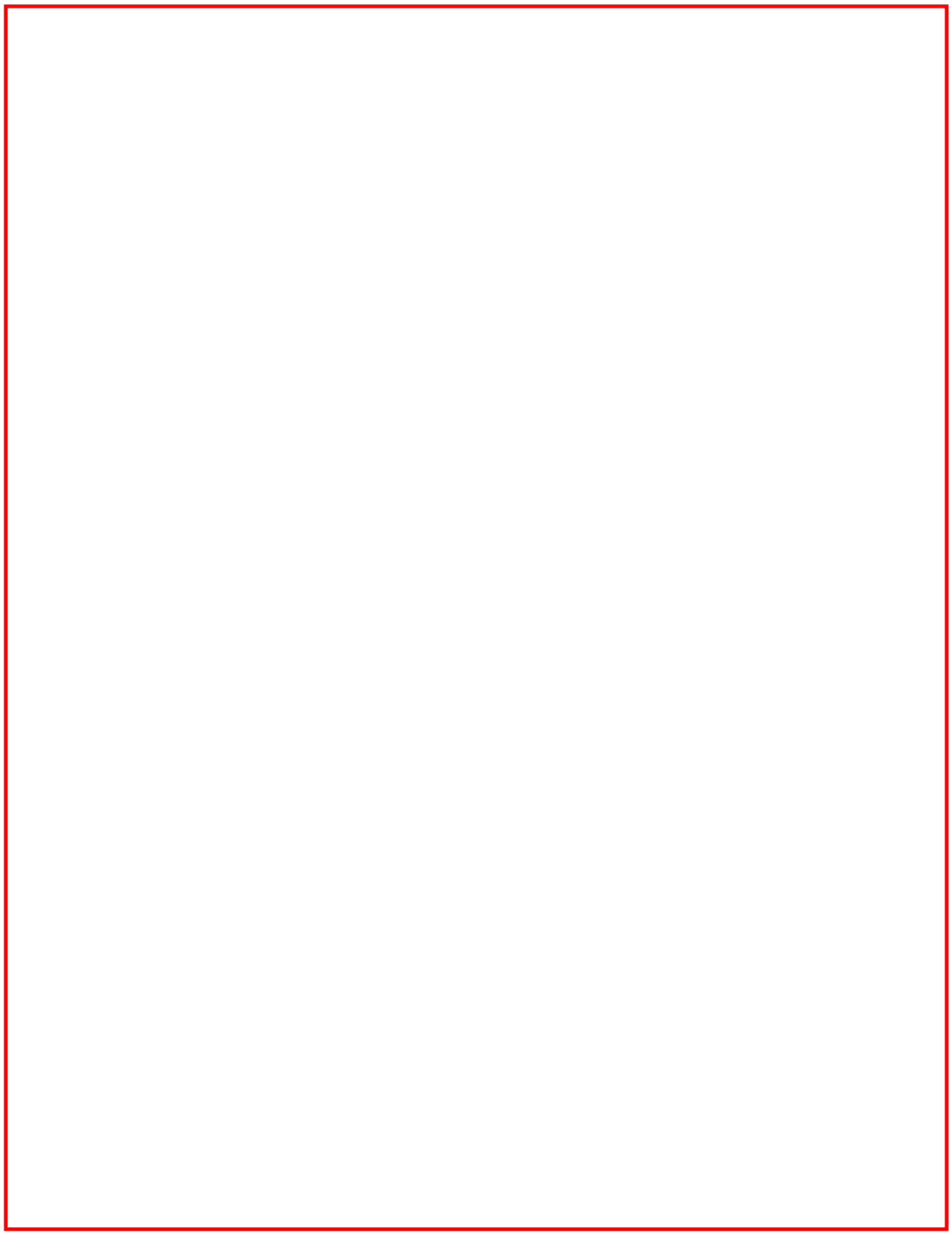
Task 1.1b

When the circuit was tested, it was observed that contrary to the specifications, when `AxD` is incremented, the new value of `AxD` (i.e., `AxD+1`) is not used to calculate `BxD <= AxD xor "10101010"`. Likewise, when calculating `CxD` from `BxD`, the new value of `BxD` is not used. In other words the outputs `AxD`, `BxD`, and `CxD` are inconsistent when compared to the values of the variables in the algorithm specification after one pass of the algorithm. Explain why this happens.

Response area for task 1.1b.

Task 1.1c

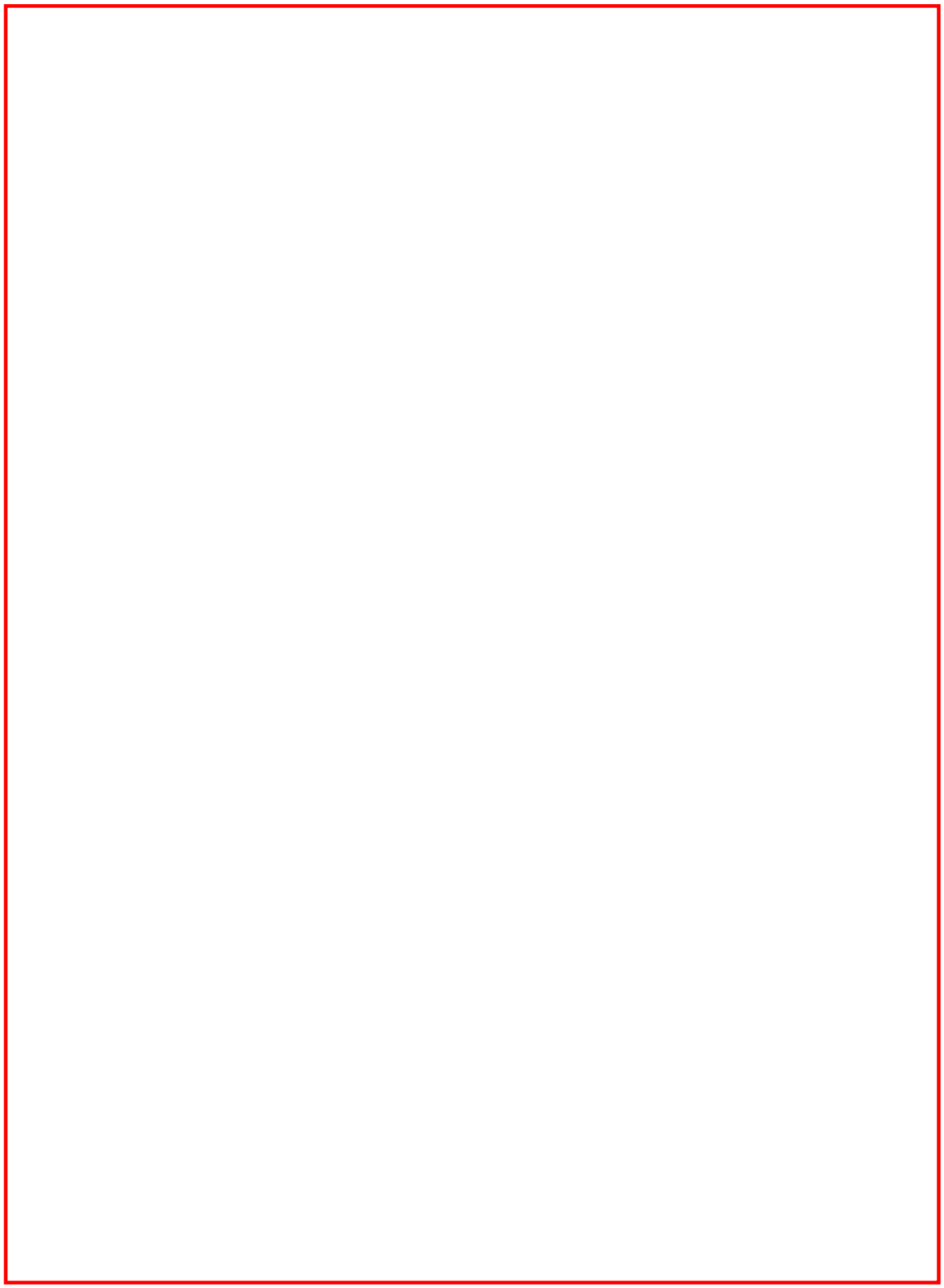
In its current form, i.e., without creating a new process for combinational logic (`process(all)` or writing additional logic outside the clocked process (`process(CLK, RSTxRI)`), how would you fix the code so that the outputs in each cycle are consistent with the values of A, B, and C at the end of each execution of the reference algorithm in Listing 1. Describe whether it is possible to do so and how you would modify the VHDL code inside the clocked process to do this. You can write out the new VHDL code by hand just for that small part (lines 35-37) in the response area below.



Response area for task 1.1c.

Task 1.1d

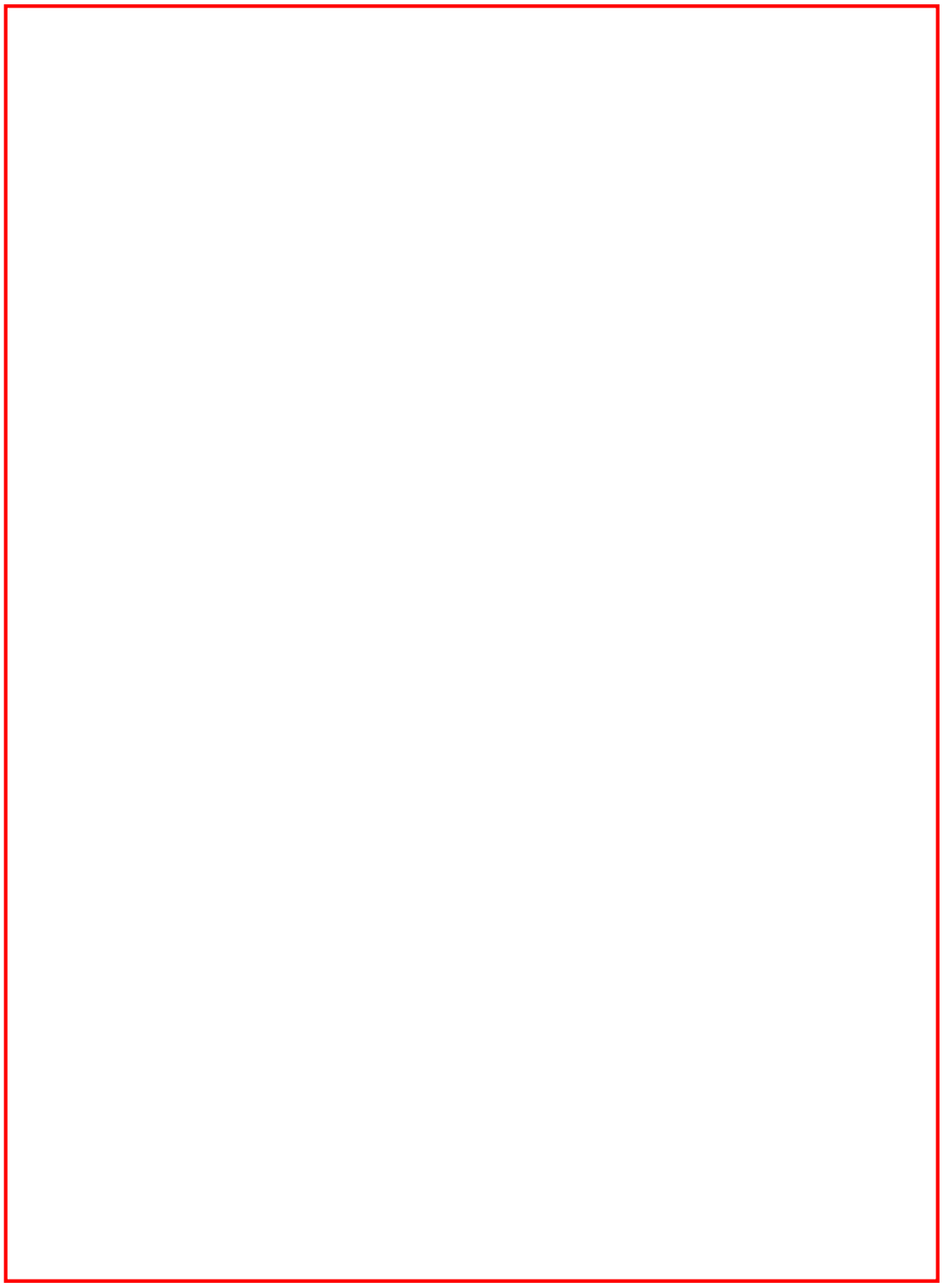
This style of writing code where everything is written inside a single process that describes both sequential and combinational logic is called a *one-segment description*. This is considered to be bad practice! Explain why this style of writing VHDL code is considered bad practice in the response area below. Think back to the exercise in the class on this and your responses to previous sub-tasks in this part of the exam.



Response area for task 1.1d.

Task 1.1e

Now, we consider rewriting the code to use separate processes to separate the sequential and combinational logic. Use the response area below to now write out 2 processes. You need to use one clocked process that updates the registers and one process for the combinational logic. You just have to write the 2 processes and any new signal definitions, you do not need to re-write the entity definition part with the ports or the output. Please use the notation `xDN` and `xDP` for the registers.



Response area for task 1.1e.

Part 2: VHDL to Schematics

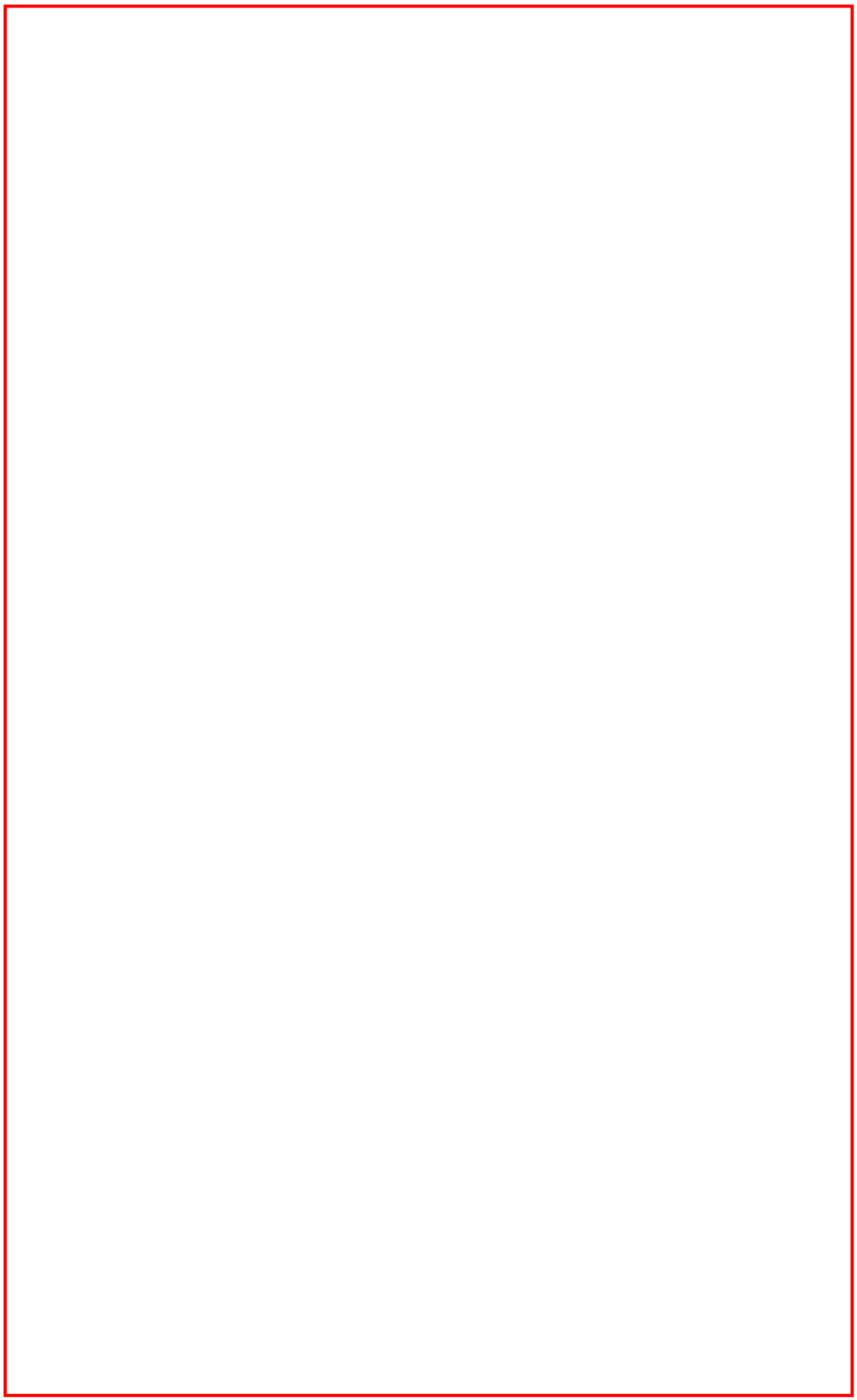
This part of the exam is concerned with translating VHDL code to circuit schematics.

Task 2.1: Dot product computation

The VHDL code snippet in Li sting 3 describes a module that performs dot products. The two input elements `AxDI` and `BxDI` are Multiplied and Accumulated to the partial result (this is called a MAC operation). The partial result is stored in the register `MacxDP` and outputted at `MacxDO`. The MAC operation is performed in each clock cycle only if `ENxSI` is active. The signal `ClearxSI` is used to start a new MAC operation. Your task is to translate this code into a schematic using standard components such as multipliers, adders, multiplexers, and so forth.

```
10 entity task2_1 is
11   port (
12   CLKxCI    : in std_logic;
13   RSTxRI    : in std_logic;
14   ENxSI     : in std_logic;
15   ClearxSI  : in std_logic;
16   AxDI      : in unsigned(16-1 downto 0);
17   BxDI      : in unsigned(16-1 downto 0);
18   MacxDO    : out unsigned(16-1 downto 0)
19   );
20 end task2_1;
21
22 architecture rtl of task2_1 is
23   signal ProdxS                : unsigned(32-1 downto 0);
24   signal SumxS, MacxDN, MacxDP : unsigned(16-1 downto 0);
25 begin
26
27 process (CLKxCI, RSTxRI)
28   begin
29     if (RSTxRI ='1') then
30       MacxDP <= (others => '0');
31     elsif (CLKxCI'event and CLKxCI = '1') then
32       MacxDP <= MacxDN;
33     end if;
34   end process;
35
36   ProdxS <= AxDI * BxDI;
37   SumxS  <= ProdxS(16-1 downto 0) + MACxDP;
38   MacxDN <= ProdxS(16-1 downto 0) when ClearxSI = '1' and ENxSI = '1' else
39             SumxS                 when ENxSI    = '1' else
40             MACxDP;
41
42   MacxDO <= MACxDP;
43 end rtl;
```

Listing 3: VHDL code snippet for Task 2.1 from `task2_1.vhdl`.



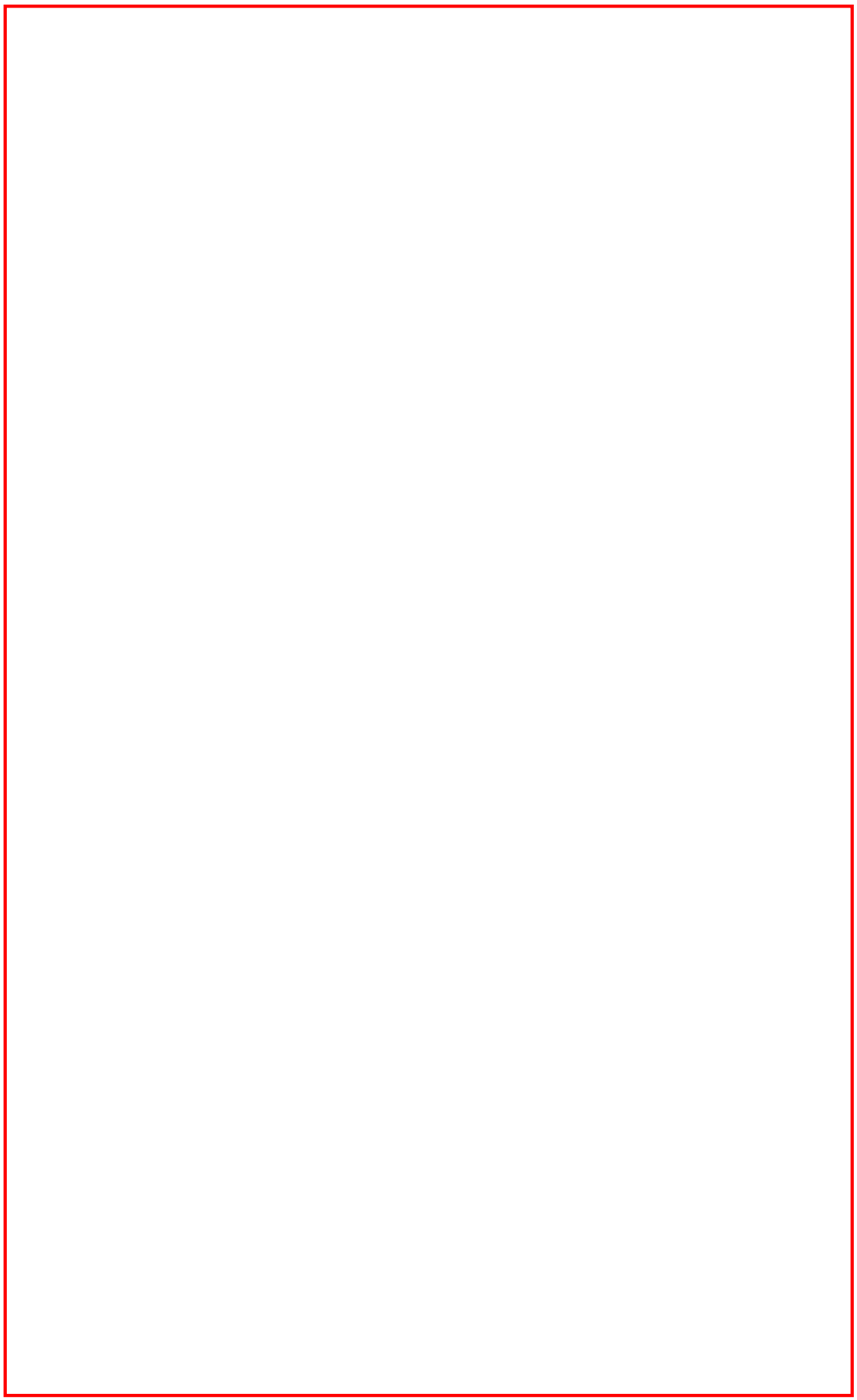
Response area for drawing the schematic of Task 2.1.

Task 2.2: Arithmetic Logic Unit

The VHDL code snippet in Listing 4 describes an ALU performing various operations on an input AxDI. Your task is to translate this code into a schematic using standard components.

```
10 entity task2_2 is
11   port (
12     CLKxCI  : in std_logic;
13     RSTxRI  : in std_logic;
14     SelxSI  : in std_logic_vector(2-1 downto 0);
15     AxDI    : in unsigned(8-1 downto 0);
16     ResxDO  : out unsigned(8-1 downto 0)
17   );
18 end task2_2;
19
20 architecture rtl of task2_2 is
21   signal ResxDN, ResxDP : unsigned(8-1 downto 0);
22 begin
23
24 process (CLKxCI, RSTxRI)
25 begin
26   if (RSTxRI ='1') then
27     ResxDP <= (others => '0');
28   elsif (CLKxCI'event and CLKxCI = '1') then
29     ResxDP <= ResxDN;
30   end if;
31 end process;
32
33 process(all)
34 begin
35   if (SelxSI = "00") then
36     if (AxDI >= 127) then
37       ResxDN <= to_unsigned(1, ResxDN'length);
38     else
39       ResxDN <= (others => '0');
40     end if;
41   elsif (SelxSI = "01") then
42     if (AxDI >= 127) then
43       ResxDN <= AxDI;
44     else
45       ResxDN <= (others => '0');
46     end if;
47   elsif (SelxSI = "10") then
48     ResxDN <= '0' & AxDI(8-1 downto 1);
49   else
50     ResxDN <= AxDI(8-2 downto 0) & '0';
51   end if;
52 end process;
53
54 ResxDO <= ResxDP;
55 end rtl;
```

Listing 4: VHDL code snippet for Task 2.2 from `task2_2.vhdl`.



Response area for drawing the schematic of Task 2.2.

Part 3: Schematics to VHDL

This part of the exam is concerned with converting schematics to VHDL code. **The schematics do not have any errors in them.**

Task 3.1: Implementation of Mathematical Function

In this task, you will explain what the circuit in Figure 1 is doing, translate the schematic of the circuit into VHDL code, and write a testbench.

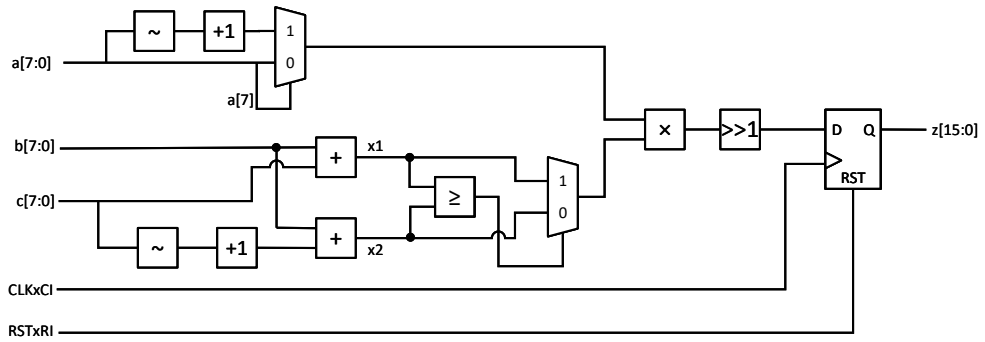


Figure 1: Schematic for Task 3.1.

Task 3.1a: Circuit Analysis

Explain what the circuit in Figure 1 is doing, i.e., what is the purpose of this circuit. As this circuit implements a mathematical function $z(a, b, c)$, you should write out this function and relate the different parts of the function to the parts of the circuits like the multiplexers and comparators. Note that the block with $\sim$ simply inverts all the bits of the input. Inverting the bits and adding 1 performs the 2's complement on that number and converts a signed number from positive to negative or negative to positive, i.e., -1 would be transformed to 1 and vice versa. The comparator outputs a `'1'` if `x1 >= x2` and otherwise a `'0'`.

Response area for explaining the functionality of the circuit in Task 3.1a.

Task 3.1b: VHDL Implementation

Translate the schematic into VHDL code using the template file `task3_1b.vhdl` and upload your response to Moodle. Please name your file `SCIPER_task3_1b.vhdl` and replace `SCIPER` with your SCIPER number.

The ports `a`, `b`, and `c` are all 8-bit signed numbers in 2's complement and `z` is 16-bit signed. The bit-widths of the internal signals are up to you, but you should try to avoid overflows. You can check for overflows in Task 3.1c by comparing the output of your testbench using the mathematical function derived in Task 3.1a.

Task 3.1c: VHDL Simulation

Write a testbench for the component you implemented in Task 3.1b using `task3_1c_tb.vhdl` and upload your response to Moodle. Please name your file `SCIPER_task3_1c_tb.vhdl` and replace `SCIPER` with your SCIPER number.

You testbench should consider the following 5 input combinations:

1. `a=3, b=2, and c=4`
2. `a=30, b=-25, and c=-10`
3. `a=-128, b=-32, and c=-14`
4. `a=-128, b=-128, and c=-127`
5. `a=127, b=90, and c=-128`

For each of these input combinations, you should calculate the expected value based on your equation and compare to what you get in simulation. The last 3 input patterns can help you identify if there are any overflows in your design. Remember to reset your circuit first and also generate a clock. Leave 1 clock cycle between each of the above steps to observe the behavior of the circuit. It is enough to just observe the behavior of the circuit, there is no need to put assertions on the circuit output for verifying its correctness.

In the response area below, please indicate your output result from the VHDL code for each of the 5 different input combinations. If the output from your VHDL code does not match your mathematical function, please explain why you think this is the case and what you may have to change in your code. If the result matches what you expect, it means that you have correctly implemented the function in your code.

1. z=
2. z=
3. z=
4. z=
5. z=

Response area for Task 3.1c.

Part 4: Circuit Design and Analysis

This part of the exam is concerned with circuit design and analysis from a high-level description.

Task 4.1: FIR Filter Implementation

In this task, we consider an FIR filter used in a wireless system. An FIR filter can be described by the equation

$$y[n] = \sum_{k=0}^{K} x[n-k]h[k]\,,$$

where K is the filter order, $x[n-k]$ is the input in cycle $n-k$ (so $x[n-k]$ represents delayed versions of x), $h[k]$ is the k-th filter weight, and $y[n]$ is the output in cycle n. Input data is received in every cycle and the filter just runs continuously.

Task 4.1a

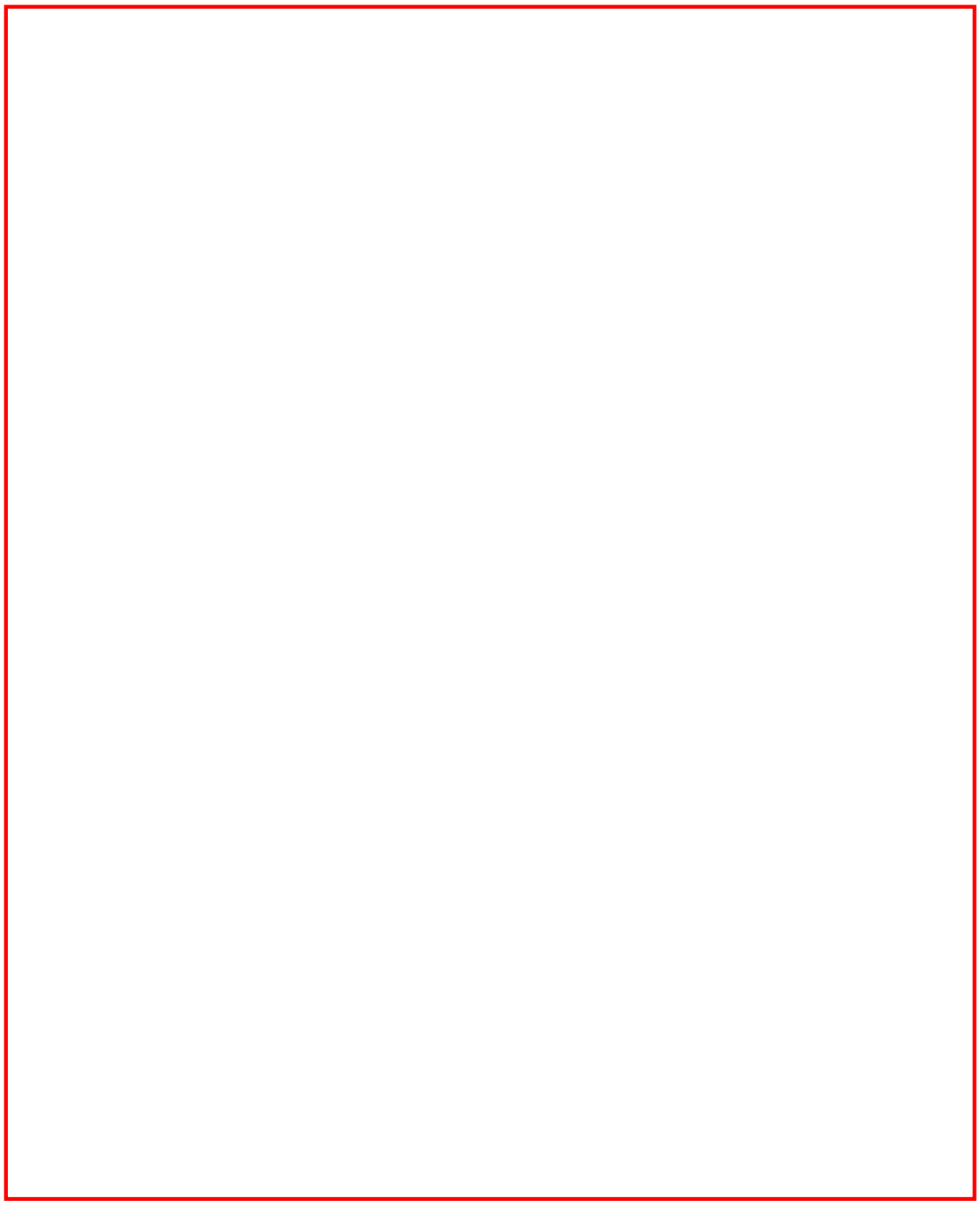
For $K = 2$, draw the isomorphic hardware architecture of the FIR filter using multipliers, adders, and registers. Signals `x[n]`, delayed versions `x[n-1]` and `x[n-2]`, filter weights `h[0]`, `h[1]`, `h[2]`, and the output `y[n]` must all be explicitly labelled in the circuit. Assuming that the input x is 8-bits and all the filter weights are 8-bits, please indicate after each multiplication and addition what the required bit-width would have to be.

You can simplify your drawing by using a black vertical bar for the registers (▮). You can decide whether to add registers in the beginning for x and at the end for y, but registers are necessary to get the delayed input signal x. It can help to write out the equation for different n.

Response area for task 4.1a.

Task 4.1b

Now, we consider a change in the system. The clock speed is $40\,\mathrm{MHz}$, but the rate at which valid data is delivered at is only $20\,\mathrm{MHz}$. Hence, only every 2nd data input is valid, yet, the circuit is clocked in every cycle and can not be disabled. As an example, in cycle 0 the circuit receives $x[0]$, in cycle 1 it receives an arbitrary unrelated (invalid) input, and in cycle 2 it receives $x[1]$ and so forth. The invalid outputs can be any value as they are ignored by subsequent circuit. Given that we do not have valid data in every cycle, write out a new correctly pipelined version of your FIR filter that takes into account that there is only one valid data item in every second cycle to correctly do the filter convolution. This is a task in pipelining, there is no control logic to add on top of the circuit you have previously defined.



Response area for task 4.1b.

Part 5: Finite-State Machines (FSMs)

In this part, you will design a finite-state machine (FSM). Just as a reminder on state notation, we show the general notation for a state in a Moore FSM in Figure 2. Just like in the exercises, you can write out what your default Moore output values are and only if the Moore output is different from that do you need to write out the Moore outputs explicitly in each state. That is, if you write that `Out1xSO` by default is `"10"`, then it can be left out of the node in Figure 2.

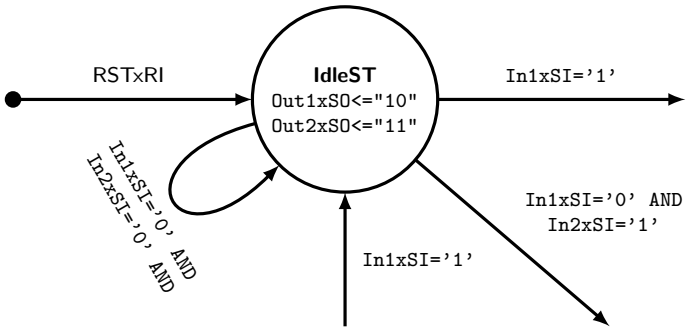


Figure 2: State notation with state name in bold. Moore outputs are indicated inside the node of a state. The initial state after reset needs is indicated. A state needs an arrow to indicate when we remain in the state along with the conditions for remaining in the current state. In this example, the transition `In1xSI='1'` implies that no matter the value of `In2xSI`, if `In1xSI='1'`, then we transition and it is not necessary to write the transition condition as `(In1xSI='1' AND In2xSI='0') OR (In1xSI='1' AND In2xSI='1')` which is much longer.

Task 5.1: Turnstile Entry Controller

Consider a turnstile entry controller into a subway station: a rotating gate with a coin receiver. There are two inputs to the system and two outputs:

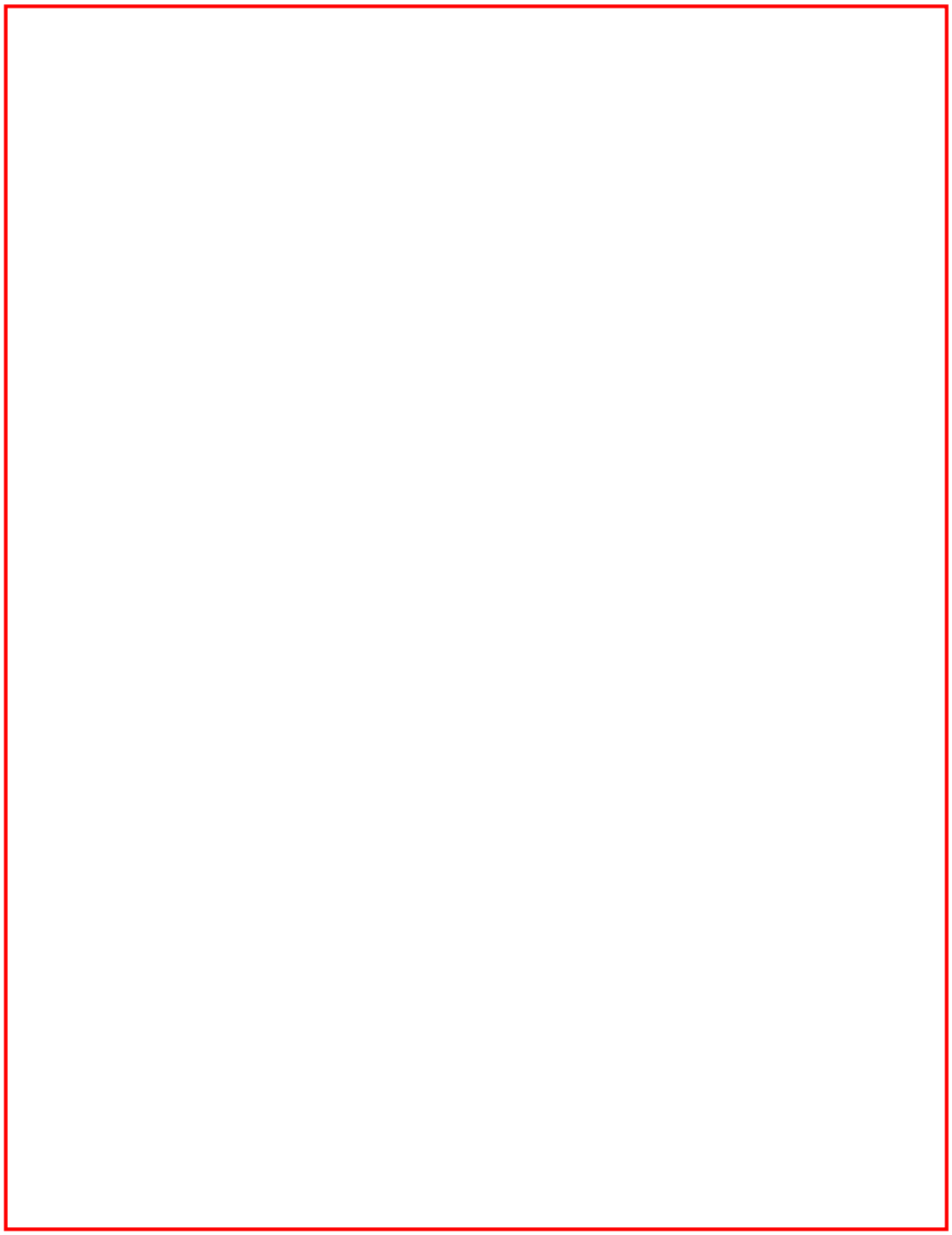
1. `CoinxSI`: Indicates a coin has been inserted
2. `PushxSI`: Indicates the arm is being pushed
3. `GatexSO`: Is `'1'` for open or `'0'` for closed in the *unlocked* and *locked* states, respectively.
4. `EjectxSO`: Is `'1'` for ejecting an inserted coin when the gate is already open, otherwise `'0'`.

The gate is initially *locked* and will not rotate or allow a passenger to pass through, no matter how many times the gate is pushed. When a coin is inserted, i.e., `CoinxSI` goes HIGH (`'1'`), the system changes the state to *unlocked*. In the *unlocked* state, any further coin insertion has no effect and the coin is simply ejected. The turnstile controller remains in the *open* state if no push is sensed for N seconds and then changes back to state *locked* and clears the counter. If the turnstile controller detects a push while in the *open* state, it also goes back to the *locked* state even if the counter has not reached N seconds and it clears the counter.

You can assume that there is an external counter outside of the FSM which the FSM can clear with a signal `CNTClearxSO`, enable with `CNTEnablexSO`, and the counter value is available as input `CNTxDI`.

Task 5.1a

Design a **Moore FSM** to control the turnstile entry based on the provided description. Use the drawing area below for writing out your state diagram. Remember to indicate the output values and the values required for state transitions. You may indicate in your response the default outputs of your FSM and then only when the output is different from that do you have to write it inside the node of a state. Please give your states and additional signals meaningful names. You can implement your FSM with any number of states and any number of additional signals beyond the described input and output signals if this is necessary.



Response area for drawing the Moore FSM for the turnstile entry controller in Task 5.1a.

Part 6: Architectural Transformations and Timing Analysis

This part is concerned with architectural transformations, timing analysis, and pipelining.

Task 6.1: Algorithm to Architecture

We consider a datapath shown in Figure 3. The area and delay specifications of the operations (and the registers) are provided in Table 1. Assume that the inputs are stable at $t_\Delta = 0$. Remember that the time-per-data-item is defined in both clock cycles and time (nanoseconds). The time-per-data-item in clock cycles is the number of computation cycles between releasing two subsequent data items, that is, producing two results.

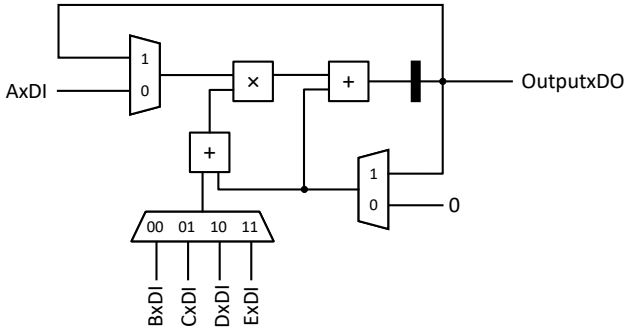


Figure 3: Diagram of Task 6.1.

Table 1: Area and timing for various components for Task 6.1.

Component	Area	t_{pd} [ns]	t_{cd} [ns]	t_{su} [ns]	t_{ho} [ns]
2-1 MUX	2	0.1	0.1	–	–
4-1 MUX	5	0.15	0.15	–	–
Mult.	12	1.5	0.5	–	–
Add/Sub	4	0.6	0.3	–	–
Register	1	0.25	0.15	0.3	0.1

Consider the schematic of the iteratively decomposed datapath in Figure 3. This circuit processes 5 numbers over multiple cycles. We distinguish the following four path groups with the corresponding minimum and maximum delays:

- Input-to-output delay: t_{io}^{min} and t_{io}^{max}.
- Input-to-register delay: t_{is}^{min} and t_{is}^{max}.
- Register-to-output delay: t_{so}^{min} and t_{so}^{max}.
- Register-to-register delay: t_{ss}^{min} and t_{ss}^{max}.

The circuit iteration over 5 cycles is as follows (this illustrates how the muxes are controlled):

1. `OutputxDN <= AxDI * BxDI`
2. `OutputxDN <= OutputxDP * (OutputxDP + CxDI) + OutputxDP`
3. `OutputxDN <= OutputxDP * (OutputxDP + DxDI) + OutputxDP`
4. `OutputxDN <= OutputxDP * (OutputxDP + ExDI) + OutputxDP`
5. `OutputxDN <= AxDI * BxDI`

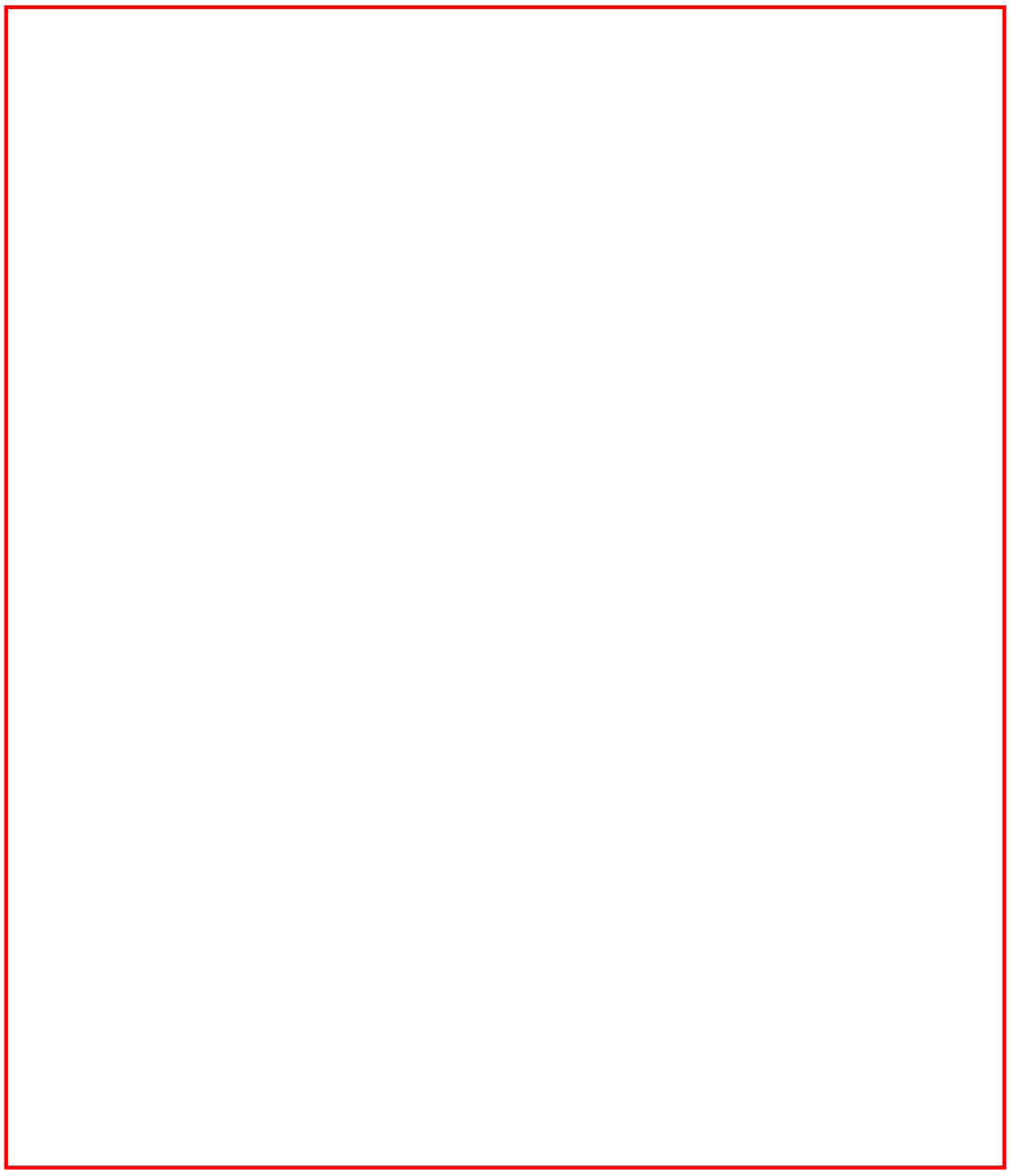
Task 6.1a: Iteratively Decomposed Circuit Analysis

1. Which of the four path groups is not meaningful (has no path) in Figure 3? For the meaningful path groups, please estimate the min and max of these, i.e., t_{xx}^{min} and t_{xx}^{max}.
2. What is the area and minimum clock period? For this and the remaining questions, you can assume that the inputs are stable at $t_{\Delta} = 0$.
3. What is the time-per-data-item in clock cycles and nanoseconds for the circuit?

Response area for steps 1-3 in Task 6.1a.

Task 6.1b: Isomorphic Circuit Analysis

1. Draw the isomorphic architecture of a circuit that generates the same result in 1 cycle. The isomorphic circuit should have registers with the provided timing parameters at the inputs and outputs.
2. What is the minimum clock period for the isomorphic architecture?
3. Draw the pipelined isomorphic architecture so that each pipeline stage matches an iteration of the iterative circuit. The pipelined isomorphic circuit should have registers with the provided timing parameters at the inputs and outputs.
4. What is the minimum clock period and area for the pipelined circuit?
5. What is the time-per-data-item in clock cycles and nanoseconds for the pipelined circuit?

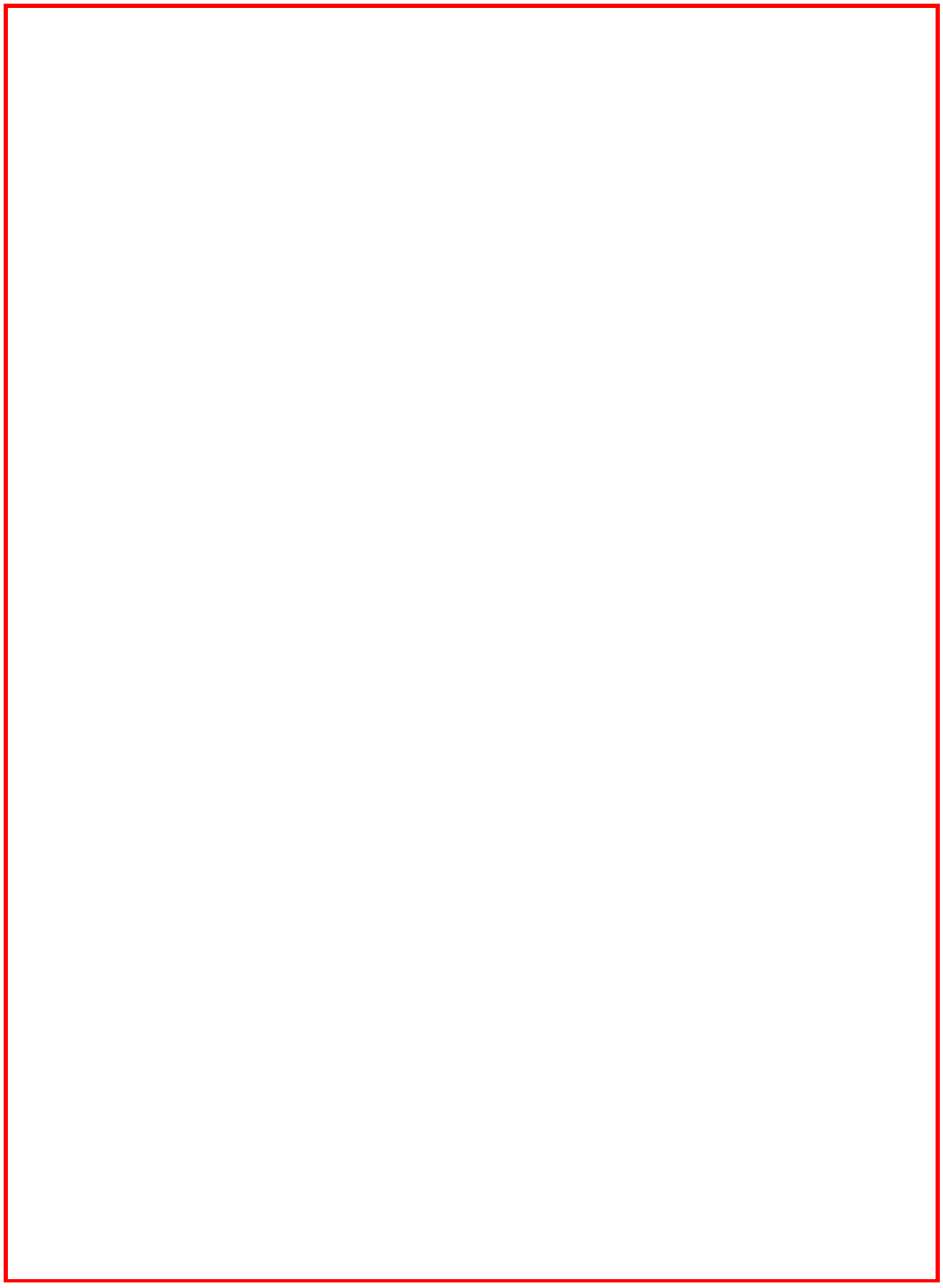


Response area for step 1-2 for the **isomorphic architecture** in Task 6.1b.

Response area for steps 3-5 for the **pipelined isomorphic architecture** in Task 6.1b.

Task 6.1c: Architecture Comparison

1. Compute and compare the efficiency (in terms of area-delay product) of the iterative, isomorphic, and pipelined circuits. What do you observe? Which is the most efficient implementation in terms of its AT-product (provide the AT-products)?
2. In what situations would you opt for the circuit with the worse AT-product (despite its AT-product disadvantage) and why?



Response area for steps 1-2 for Task 6.1c on the AT-products.