DigitalSystems: exam

R. Pacalet

2023-06-26

You can use any document but communicating devices are strictly forbidden. Please number the different pages of your paper and indicate on each page your first and last names. You can write your answers in French or in English, as you wish. Precede your answers with the question's number. If some information or hypotheses are missing to answer a question, add them. If you consider a question as absurd and thus decide to not answer, explain why. If you do not have time to answer a question but know how to, briefly explain your ideas. Note: copying verbatim the slides of the lectures or any other provided material is not considered as a valid answer. Advice: quickly go through the document and answer the easy parts first.

The 3 questions are worth 2 points each. The first problem is worth 6 points The second problem is worth 8 points

1 Questions

1.1 A VHDL synchronous process

The VHDL process in the listing below is intended to be:

- synthesizable,
- synchronous on the rising edge of **`clk`**,
- with a synchronous, active low, reset **`rstn`**.

1. Try to imagine the intended behavior and briefly explain it.
2. What do you think of this code?
3. Are there errors or inefficiencies? If yes explain for each of them why it is an error or an inefficiency and what undesirable effect it has.
4. Finally, write down a new VHDL code with all the errors and inefficiencies fixed and with the behavior you imagined.

```
process(clk, di, dsi, rstn)
begin
  do <= '0';
  if rstn /= '0' and falling_edge(clk) then
    if dsi = '1' then
      do <= di;
    else
      do <= do;
    end if;
  end if;
end process;
```

1.2 Unresolved and resolved types

In the DHT11 project we used a VHDL resolved type to model the data line between the controller and the sensor.

1. What is a resolved type and why is it different from a non-resolved type?
2. Why did we need this for the DHT11 project?

1.3 VHDL processes

We want to design a synthesizable VHDL model of a circuit which all outputs are outputs of registers and all registers are synchronized by the same edge of the same clock.

1. What is the minimum number of VHDL processes we need?
2. Why?

2 Problem: re-synchronization

To design our DHT11 controller we had to handle a re-synchronization issue: as the data line coming from the sensor was not synchronous with our clock, it could change any time, including during the critical time windows around the rising edges of our clock. So, using it without special care could have led to metastability situations.

1. (**2 points**) Metastability

- Explain what metastability is and why it is undesirable in digital hardware designs.
- Explain how we solved the metastability issue in our DHT11 controller.
- Was this solution a 100% guarantee that the DHT11 system will always work properly?

Suppose you are in charge of designing a digital hardware component `foo` that is synchronized on the rising edges of one single clock `clk` with a synchronous active low reset `rstn`. One of the inputs of the component is a 2 bits `data` bus from which we compute a 1-bit output `do` with a `some_processing` function provided by package `foo_pkg`. The listing below shows your starting point, that has already been designed by John before he had to stop working on this project.

As for the DHT11 controller, the `data` input of `foo` is not synchronized with respect to `clk`: it can change any time, including during the critical time windows around the rising edges of `clk`. But different from the DHT11 controller, `data` is multi-bit.

If we were using the same solution as for the DHT11 controller we could end up with invalid values of `data` injected in our design because we could sample a mixture of old and new values. Example: suppose `data` transitions from `01` to `10` during the critical time windows around a rising edge of `clk`. We could sample the new value of the left bit (`1`) and the old value of the right bit (`1`), ending with `11` wich is none of the old or new real `data` values. And of course, this could have severe consequences...

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

use work.foo_pkg.all;

entity foo is
  port(
    clk:  in  std_ulogic;
    rstn: in  std_ulogic;
    data: in  std_ulogic_vector(1 downto 0);
    do:   out std_ulogic
  );
end entity foo;

architecture rtl of foo is
  signal internal_data: std_ulogic_vector(1 downto 0);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if rstn = '0' then
        internal_data <= (others => '0');
      else
        internal_data <= data;
      end if;
    end if;
  end process;

  do <= some_processing(internal_data);
end architecture rtl;
```

2. (**2 points**) Imagine and explain a solution to re-synchronize the `data` input of `foo` before using it in computations. If it helps you can invent a custom communication protocol of your own between the source of `data` and `foo`; to do so you can add more inputs or outputs to our component, but you cannot add inputs that would be synchronized with respect to `clk` (if we could do so, we would simply decide that `data` is synchronous).

3. (**2 points**) Modify John's VHDL code to implement your proposal.

3 Problem: design of a median filter

Median filters are used in image processing as a cleaning tool to remove defects and noise from pictures. It is very simple in principle but its implementation in a dedicated piece of hardware poses some interesting challenges.

The algorithm is very simple: every pixel of the picture to be filtered is replaced by the median value of the neighboring pixels. The picture is thus transformed by the median filter into another picture that has exactly the same size. For every pixel P of the input picture we first create a list of the 9 (3×3) pixels surrounding P[1]. The 9 pixels are then sorted. The median value is the value located in the middle of the sorted list. The pixel P in the filtered picture takes this median value. In our example the pictures are gray scale pictures, 8 bits per pixel. The pixel values are between 0 (black) and 255 (white).

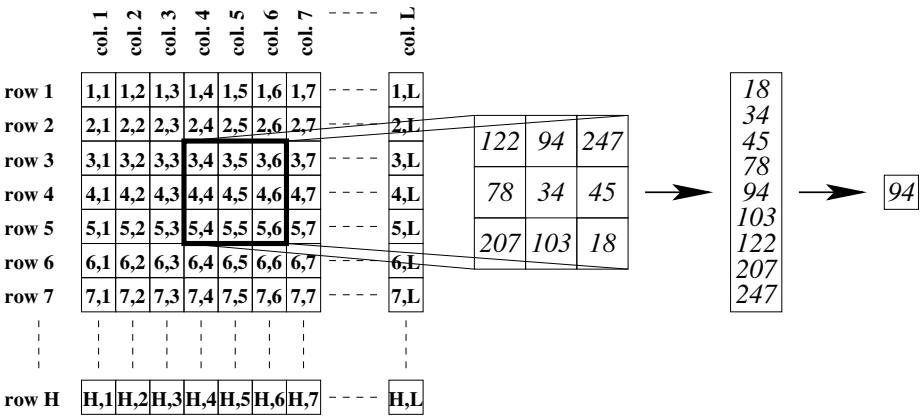

Figure 1: The principle of the median filter

Consider the situation depicted in Figure 1. Let $[X, Y]$ be the pixel located at row X, column Y of a picture. The value of pixel $[4, 5]$ in the filtered picture is computed from the 3×3 neighborhood of the pixel $[4, 5]$ of the source picture, that is, the list of pixels $[3, 4], [3, 5], [3, 6], [4, 4], [4, 5], [4, 6], [5, 4], [5, 5], [5, 6]$. The list of the pixel values in our example is $122, 94, 247, 78, 34, 45, 207, 103, 18$. The sorted list is $18, 34, 45, 78, 94, 103, 122, 207, 247$. The median value is 94, so the value of the pixel in the filtered picture is 94. Figure 2 shows the effect of a 3×3 median filter on a picture of Humphrey Bogart.

The goal of this problem is to design a digital hardware machine extracting the median value of a list of 9 pixels. This machine could then be used as a co-processor for a general purpose CPU to speed up a software implementation. It could also be integrated in a larger design that would implement the filter completely in hardware.

[1]Pixels on the edges of the picture have less than 9 neighbors. In most cases the missing pixels are simply copies of the edge pixels or set to a fixed color (e.g., black or white).

Original Noisy Filtered

Figure 2: Example of effect of a 3×3 median filter

One of the sub-components of our design is `CE`, a simple **C**ompare and **E**xchange unit. Its interface is specified in Figure 3 and Table 1.

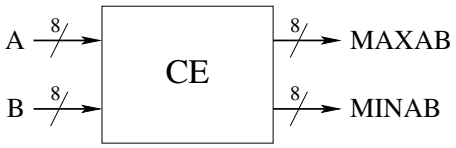

Figure 3: Interface specification of `CE`

Table 1: Interface specification of `CE`

Name	Size	Direction	Description
`A`	8	Input	Pixel value
`B`	8	Input	Pixel value
`MAXAB`	8	Output	The maximum of the two pixel values
`MINAB`	8	Output	The minimum of the two pixel values

`CE` is purely combinatorial. It compares the two input pixels and puts the largest of its two inputs on the `MAXAB` output and the other on its `MINAB` output.

1. (**2 points**) Design `CE` in plain synthesizable VHDL (entity and architecture).

To extract the median value of 9 pixels we will now design a **MED**ian sorter unit, `MED`. It is sequential (synchronous) and it instantiates `CE` once and only once. Its interface is specified in Figure 4 and Table 2. The communications between `MED` and the enclosing system are summarized on Figure 5.

Figure 4: Interface specification of `MED`

Table 2: Interface specification of `MED`

Name	Size	Direction	Description
`DI`	8	Input	**D**ata **I**nput bus
`DSI`	1	Input	**D**ata **S**trobe **I**n; when high indicates that `DI` carries a valid input pixel
`BYP`	1	Input	Control input that changes the behavior of `MED` (see below)
`CLK`	1	Input	**CL**oc**K**; `MED` is synchronous on the rising edge of `CLK`
`DO`	8	Output	**D**ata **O**utput bus; when the processing is done this output bus holds the resulting median value

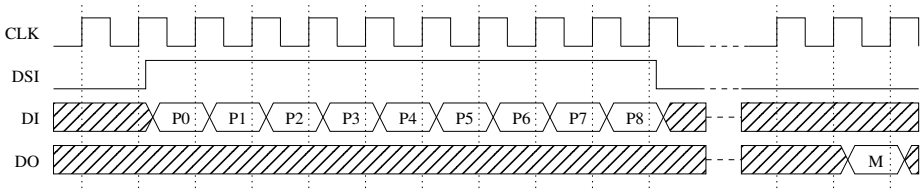

Figure 5: Waveforms of the `MED` input/output communications

The system inputs 9 pixels (`P0`, `P1`, ..., `P8`), one per clock period. During the 9 periods `DSI` is active. There is no interruption during this input phase: the 9 periods are consecutive. After the last pixel is input `MED` starts extracting the median value and, when done, puts it on the `DO` bus. This computation takes several clock periods. After the processing is done and the result is output a new set of 9 pixels may be presented. We assume in the following that the surrounding system always uses this simple protocol. It never tries to input a new set before the previous computation is done. The architecture of `MED` is depicted on Figure 6. `MUX8` objects are 2 to 1 multiplexers of 8 bits words. `R0`, `R1`, ..., `R8` are 8 bits registers.

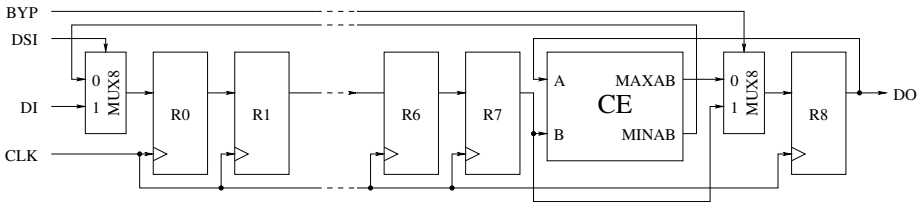

Figure 6: Internal architecture of `MED`

2. (**4 points**) As you did for `CE`, design `MED` in plain synthesizable VHDL (entity and architecture).

3. (**1 point**) Study the internal architecture of `MED` and try to imagine how the (missing) control part can extract the median value by driving the `DI`, `DSI` and `BYP` inputs. Describe the sequence of operations, from the input of the 9 pixels to the output of the result.

4. (**1 point**) Assuming `MED` runs at a 500 MHz clock frequency, what is the maximum throughput (in pixels per second)? Would it be sufficient to filter in real time a video stream with 25 pictures per second, 720×576 pixels each?