EDC: exam

Renaud PACALET

28 June 2010

You can use any document you need. On each page of your paper please indicate your name. Put your answers in any order but indicate first the corresponding question number. Example:

```
Question 2.1: I think this is a very good question and...
```

Advice: read every word very carefully and answer first the easy parts. You can write your answers in English or in French, as you like. Do not forget to attach your labs paperwork if you did not send it to me already or send them to me today, sharp deadline.

The five questions of the first part are worth 2 points each. If you consider a question as ambiguous explain why, then make the assumptions needed to solve the ambiguities and answer the modified question. If you consider a question as absurd explain why. If you find an error in a question explain why you think it is a error, propose a correction if it is reasonable and answer the modified question.

The second part is a small problem about multiplication-accumulation (MAC). MACs are used in digital signal processing, for instance when computing a dot product of two vectors or when applying a filter to a sequence of samples. It is very simple in principle but its implementation in a dedicated piece of hardware poses some interesting challenges. This part is worth 10 points.

1 Questions

1.1. We consider $ABREG$, a small digital circuit, represented on figure 1. A and B are primary inputs. RA and RB are primary outputs. The two registers are regular D flip-flops. They sample their inputs on the rising edge of clock CLK. Express in CTL a temporal-logic property stating that the environment is not allowed to change both A and B in a single clock period.

1.2. We consider INC, a small digital circuit, represented on figure 2. A is a 16 bits primary input vector. SEL is a single bit primary input. DO is a 16 bits primary output vector. The circuit contains a 16 bits multiplexer, a 16 bits register (that samples its input on the rising edge of clock CLK) and a 16 bits incrementer (a simple combinatorial element that adds 1 to its 16 bits input and outputs the 16 least significant bits of the result). What is the minimum number of processes required to model this circuit in synthesizable VHDL? Explain your answer.

1.3. Same question as above about $INCR$, the circuit represented on figure 3.

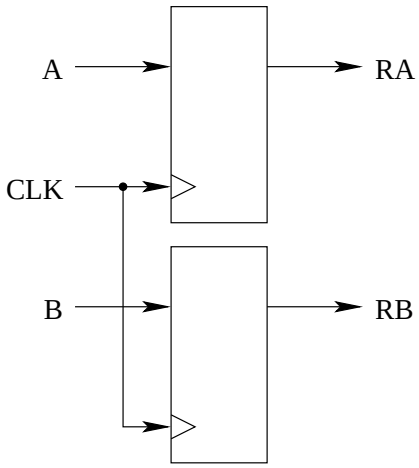

Figure 1: The $ABREG$ digital circuit

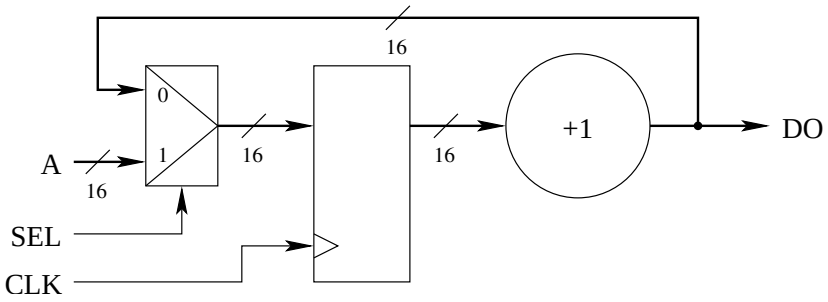

Figure 2: The INC digital circuit

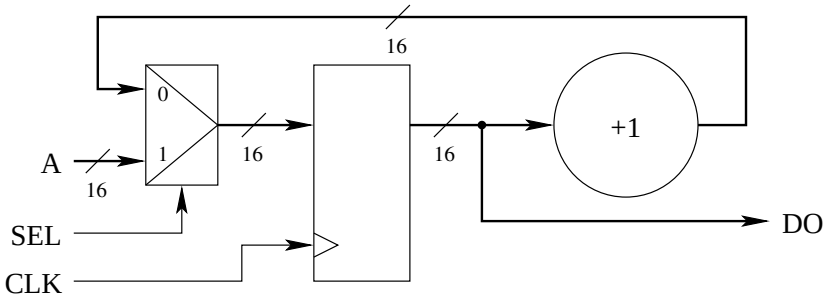

Figure 3: The $INCR$ digital circuit

1.4. In your opinion, what will happen when synthesizing a design that contains the following VHDL process?

```
process(a, b)
begin
  if sel = '0' then
    d <= a;
  elsif sel = '1' then
    d <= b;
  end if;
end process;
```

1.5. In case a software implementation of a given algorithm is not fast enough for a given application field, what are the different possibilities you know that could be tried to solve the issue?

2 Design of a multiplier-accumulator

We want to use the NiosII CPU to perform some digital signal processing. The data we manipulate the most frequently are vectors of complex numbers. Each component of a vector is represented on 32 bits, the 16 leftmost bits representing the real part and the 16 rightmost bits representing the imaginary part. The real and imaginary parts are signed integers in the range $[-32767...32767]$, represented in two's complement. Reminder: the two's complement representation of an n bits integer is defined by equation (1) in which B is the integer value and b_{n-1} (b_0) is the leftmost (rightmost) bit of its two's complement representation .

$$B = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{i=n-2} b_i \times 2^i \tag{1}$$

Unfortunately the NiosII CPU is not very efficient at computing on vectors of complex numbers and the performance of a pure software implementation would not be sufficient for our needs. We thus decide to equip the NiosII CPU with a dedicated co-processor named CPXMAC. The interface complies with the co-processor interface of a NiosII CPU and is represented in figure 4 and table 1. The VHDL code for the entity of CPXMAC is given below.

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

entity cpxmac is
  generic(nwidth: natural);
  port(clk, clk_en, reset, start: in std_ulogic;
       dataa, datab: in signed(31 downto 0);
       n: in std_ulogic_vector(nwidth - 1 downto 0);
       done: out std_ulogic;
       result: out signed(31 downto 0));
end entity cpxmac;
```

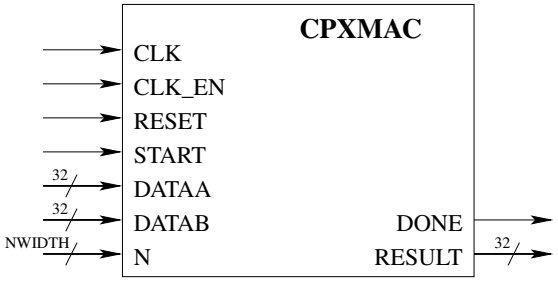

Figure 4: Interface specification of the CPXMAC co-processor

Name	Size	Direction	Description
CLK	1	Input	Clock. CPXMAC is synchronous on the rising edge of CLK
CLK_EN	1	Input	Clock enable. When low, all internal registers are frozen and hold their current value
RESET	1	Input	Synchronous, active high reset. When high on a rising edge of CLK for which CLK_EN is also high, force the content of all internal registers to a predefined value
START	1	Input	Start computation. When high on a rising edge of CLK for which CLK_EN is also high and RESET is low, launches the command specified by N
DATAA	32	Input	Data input. A complex number represented as specified above
DATAB	32	Input	Data input. A complex number represented as specified above
N	NWIDTH	Input	Command selector. The command executed depends on the value of N when the START is given
DONE	1	Output	End of computation. When high on a rising edge of CLK for which CLK_EN is also high and RESET is low, indicates that the previous command is over and its result available on RESULT
RESULT	32	Output	Data output. The result of the last command

Table 1: Interface specification of the CPXMAC module

CPXMAC will allow the CPU to compute the product of two complex numbers in one cycle only. It will also allow to accumulate the consecutive results in an internal register.

CPXMAC contains a combinatorial complex multiplier (CPXMUL) which entity is given below.

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

entity cpxmul is
  port(dataa, datab: in signed(31 downto 0);
       result: out signed(xx - 1 downto 0));
end entity cpxmul;
```

CPXMUL multiplies its two inputs and puts the result on its XX bits $result$ output. The inputs representation is the one already presented. The result is represented in a similar way: the leftmost half is the real part and the rightmost half is the imaginary part. Reminder: the product of two complex numbers is defined by equation (2) in which j represents the square root of -1.

$$(a + j \times b) \times (c + j \times d) = (a \times c - b \times d) + j \times (a \times d + b \times c) \tag{2}$$

TODO (1 point): considering the range of the values at the input side, how many bits are needed to represent the result (that is, what is the value of XX)?

TODO (2 points): Design, in plain synthesizable VHDL the architecture of CPXMUL.

CPXMAC contains an accumulator, that is, a register which size is sufficient to store the dot product of two complex vectors. The largest vectors we consider are 4096 components vectors. Reminder: the dot product of two vectors is defined by equation (3) in which $\vec{x}$ and $\vec{y}$ are two n components vectors ($1 \leq n \leq 4096$) and x_i, for any $0 \leq i < n$, is the i^{th} component of vector $\vec{x}$.

$$\vec{x} \odot \vec{y} = \sum_{i=0}^{i=n-1} x_i \times y_i \tag{3}$$

TODO (1 point): what is the required number of bits for the accumulator?

We want CPXMAC to offer several functions:

- MUL: single product between two complex numbers. The result is stored in the accumulator. The 32 rightmost bits of the imaginary part of the result is put on the $result$ output when this command is executed. The command takes only one cycle from the point of view of the CPU, that is, the $result$ output is combinationally dependent on the $dataa$ and $datab$ inputs. The $done$ signal is raised in the same clock period as $start$.
- MAC: multiplication-accumulation. The product of the two $dataa$ and $datab$ inputs is computed and added to the current content of the accumulator. The result is stored in the accumulator. Reminder: the sum of two complex numbers is defined by equation (4) in which j represents the square root of -1. The 32 rightmost bits of the imaginary part of the result is put on the $result$ output when

this command is executed. The command takes only one cycle from the point of view of the CPU, that is, the $result$ output is combinationally dependent on the $dataa$, $datab$ and the current content of the accumulator. The $done$ signal is raised in the same clock period as $start$.

- INIT: initialization of the accumulator. When executed, this command puts zero in the accumulator. $dataa$ and $datab$ are ignored. A zero value is put on $result$. The command takes only one cycle from the point of view of the CPU, that is, the $done$ signal is raised in the same clock period as $start$.

- RDI0...RDIx: read out the imaginary part of the accumulator, 32 bits at a time. Depending on the width of the accumulator there will be more or less such commands. Important note: in case a RDI operation returns less than 32 useful bits, they are right aligned in the 32 bits result and the other bits are filled with zeros if the read value is positive or null, else with ones (sign extension). RDI0 returns the 32 rightmost bits of the imaginary part of the accumulator, RDI1 returns the 32 following bits of the imaginary part, etc. RDIx returns the leftmost bits of the imaginary part, sign-extended to 32 bits. The command takes only one cycle from the point of view of the CPU, that is, the $done$ signal is raised in the same clock period as $start$.

- RDR0...RDRx: same as RDI0...RDIx but for the real part of the accumulator.

$$(a + j \times b) + (c + j \times d) = (a + c) + j \times (b + d) \tag{4}$$

TODO (2 points): Complete the specifications, decide how many different RDI and RDR commands are needed, decide how many bits are needed for N (that is, what is the value of $NWIDTH$), decide how to encode the different commands on the N input. Finally, draw a block diagram of the architecture of CPXMAC. Clearly define your notations.

TODO (3 points): Design, in plain synthesizable VHDL the architecture of CPXMAC.

TODO (1 points): Bonus question: We consider the dot product between two 4096 components vectors. The operands and the final result are stored in the external memory and we assume that each 32 bits read or write operation in this memory take exactly one cycle. On a NiosII CPU estimate how many cycles it would take to compute a dot product between two 4096 components vectors with and without the co-processor.