Digital Systems
Final examination. June 13, 2022

Time limit: 120 minutes.

Explain **what you want to do and how** you want to do it before doing anything else.

1 From VHDL code to a state diagram (20 %)

a) Draw the block diagram of a typical synchronous (clock-driven) FSM that would be synthesized from a VHDL code made of one synchronous process and two combinational process).

b) Draw the graph (or state diagram) of a synchronous (clock-driven) FSM described by

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

entity fsm is
  port( clk, s_coin, flag_60, flag_600    : in std_logic;
        s_temp : in std_logic_vector(6 downto 0);
        s_fan, s_heater, s_drum, s_right : out std_logic);
end;

architecture arch of fsm is
  type multiplier_state is (I, RH, LH, R, L);
  signal current_state : multiplier_state := I;
  signal next_state : multiplier_state;

begin
  state_reg : process(clk) is
  begin
    if rising_edge(clk) then current_state <= next_state;
    end if;
  end process;

  next_state_logic : process(current_state, flag_60,
                             flag_600, s_coin, s_temp) is
  begin
    next_state <= current_state;
    case current_state is
      when I =>
        if s_coin='1' then next_state <= RH;
        end if;
      when RH =>
        if flag_600='1' then next_state <= I;
        elsif flag_60='1' and unsigned(s_temp)<=80 then
          next_state <= LH;
        elsif flag_60='1' and unsigned(s_temp)>80 then
          next_state <= L;
        elsif flag_60='0' and unsigned(s_temp)>80 then
          next_state <= R;
        end if;
      when LH =>
```

```
        if flag_600='1' then next_state <= I;
        elsif flag_60='1' and unsigned(s_temp)<=80 then
          next_state <= RH;
        elsif flag_60='1' and unsigned(s_temp)>80 then
          next_state <= R;
        elsif flag_60='0' and unsigned(s_temp)>80 then
          next_state <= L;
        end if;
      when R =>
        if flag_600='1' then next_state <= I;
        elsif flag_60='1' and unsigned(s_temp)>=70 then
          next_state <= L;
        elsif flag_60='1' and unsigned(s_temp)<70 then
          next_state <= LH;
        elsif flag_60='0' and unsigned(s_temp)<70 then
          next_state <= RH;
        end if;
      when L =>
        if flag_600='1' then next_state <= I;
        elsif flag_60='1' and unsigned(s_temp)>=70 then
          next_state <= R;
        elsif flag_60='1' and unsigned(s_temp)<70 then
          next_state <= RH;
        elsif flag_60='0' and unsigned(s_temp)<70 then
          next_state <= LH;
        end if;
      when others => null;
    end case;
  end process;

  output_logic : process (current_state) is
  begin
    s_fan <= '1'; s_heater <= '1'; s_drum <= '1'; s_right <= '1';
    case current_state is
      when I => s_fan <= '0'; s_heater <= '0';
                s_drum <= '0'; s_right <= '0';
      when RH => null;
      when LH => s_right <= '0';
      when R => s_heater <= '0';
      when L => s_heater <= '0'; s_right <= '0';
      when others => null;
    end case;
  end process;

end;
```

2 An edge detector of asynchronous signals (10 %)

Design a **falling** edge detector for asynchronous signals.

a) First draw the block diagram.

b) Next describe it in `VHDL`.

3 Unregistering a signal (10 %)

The following **VHDL** code shows two registered signals: `a` is of type `unsigned(2 downto 0)` and `flag` is of type `std_logic`.

```
process(clk) is
begin
  if rising_edge(clk) then
    flag<='0';
    if a=5 then a<=(others=>'0');
    else a<=a+1;
    end if;
    if a=4 then flag<='1'; end if;
  end if;
end process;
```

a) How many 1-bit flip-flops are used to synthesize this code?

b) Reduce this number unregistering some signals without changing the functionality of the circuit. Describe in `VHDL` the new circuit.

c) How many 1-bit flip-flops are used in this new synthesis?

4 Designing a time counter (20 %)

We want to describe in `VHDL` a counter of seconds (up to 59) and minutes (up to 59) following the structure of the next block diagram, where the signals $flag\_s$ and $flag\_m$ are an enable signal to the next block. The frequency of the signal clk is 10 Hz.

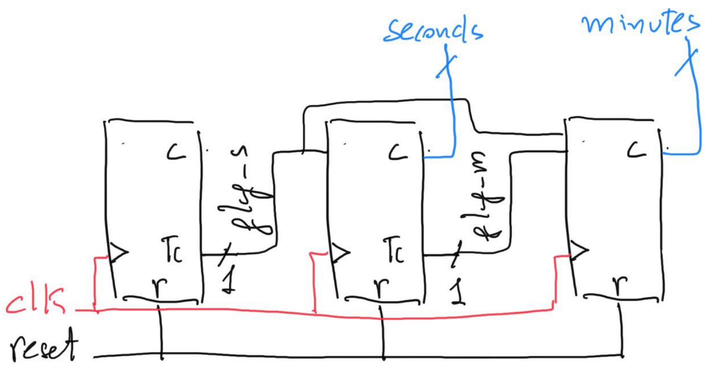

Figure 1: Block diagram of the time counter.

a) First, identify the inputs and outputs (and its size) of the whole `entity` named `counter`. These inputs and outputs are of type `std_logic` or `std_logic_vector`.

b) Next, describe the whole `entity` using one `process` for each one of the three blocks.

5 Designing a new AVR instruction (15 %)

Consider a modified `Mini AVR` that only has the following set of instructions.

- `LDI Rd,K`; Opcode: `1110 KKKK dddd KKKK`
- `MOV Rd,Rr`; Opcode: `0010 11rd dddd rrrr`
- `IN Rd,A`; Opcode: `1011 0AAd dddd AAAA`
- `OUT A,Rr`; Opcode: `1011 1AAr rrrr AAAA`
- `LD Rd, X`; Opcode: `1001 000d dddd 1100`
- `ST X, Rr`; Opcode: `1001 001r rrrr 1100`

We want to add a new instruction that stores an 8 bit constant (a literal) to a an output with the following syntax:

- `OUTI A,K`

a) Propose an `Opcode` compatible with the existing set of instructions. Point out the range of each of the operands.

b) Considering the block diagram of *Figure* 2, highlight the `signals` and `blocks` that will be used by this instruction and add, if necessary, new `signals` and `blocks`.

c) Which blocks of this diagram will be modified to consider the new instructions? Briefly explain these modifications.

6 Understanding assembler code (15 %)

The `Mini AVR` architecture of *Figure* 2 has in its ROM the following code:

```
0.  NOP
1.  LDI r17,0x0F
2.  IN r16,1
3.  EOR r16,r17
4.  NOP
5.  BRNE -4
6.  RJMP -1
```

a) Draw a waveform with the first 8 `clocks` after a general `reset` [1], considering that the **asynchronous input** `port_A1` [2], changes from 0x00 to 0x0F immediately after the first rising `clock`. Draw the value of:

- `program counter`: `pr_pc`
- `port_A1` asynchronous input
- `r16` register
- `Z` flag of the status register

[1] This general `reset` is not shown in *Figure* 2.
[2] See *Figure* 2.

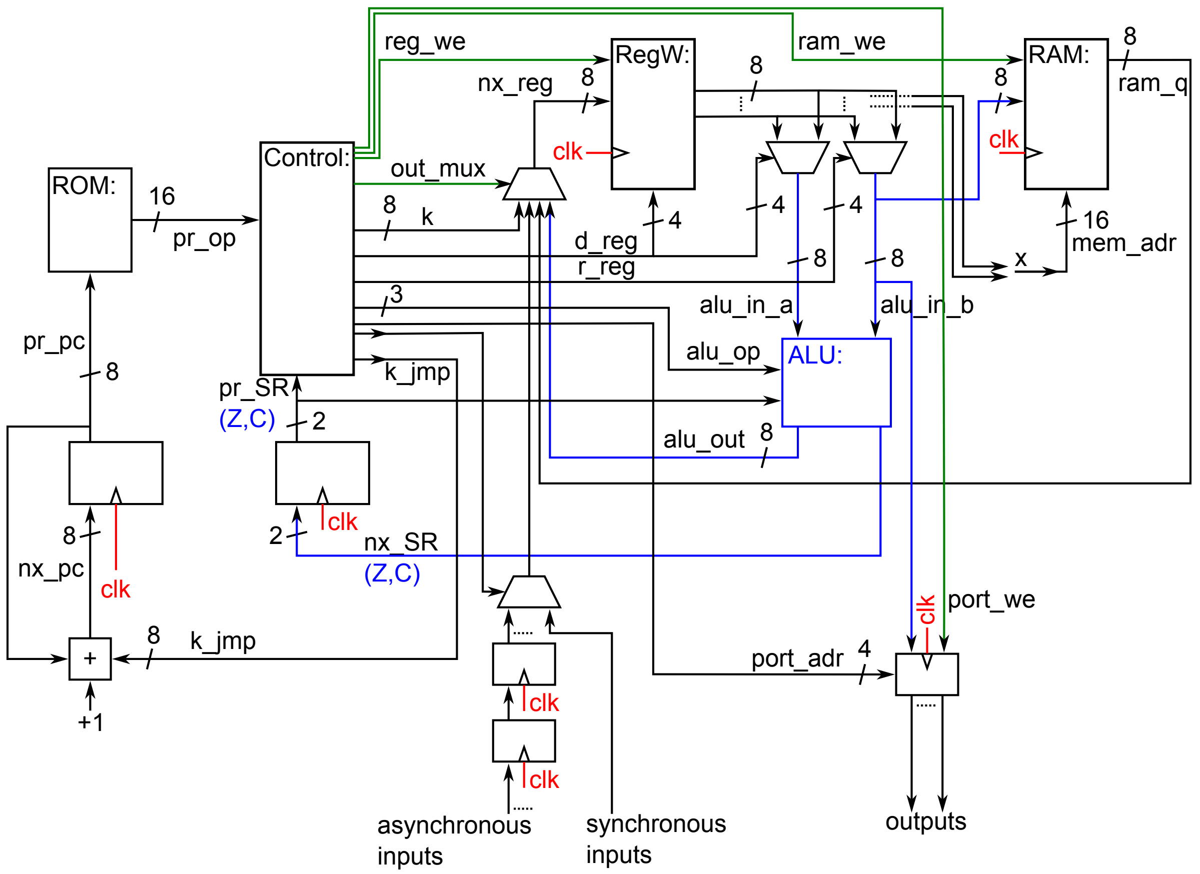

Figure 2: Mini AVR block diagram.

7 From assembler to opcode (5 %)

a) Knowing the syntax and opcode of the following instructions,

- `LDI Rd,K`; Opcode: `1110 KKKK dddd KKKK`
- `OUT A,Rr`; Opcode: `1011 1AAr rrrr AAAA`

and considering the next assembler instructions

```
0. LDI r30,0xAA
1. OUT 8,r30
```

indicate the bits in the first two positions of the `Mini AVR` program ROM.

8 Storing a literal to RAM (5 %)

a) Write the assembler instructions needed to store the value 55 in the position 512 of the `Mini AVR` data RAM.