Stopwatch on 100 seconds
Pulse and digital techniques project

Thibault Vuillemin

The stopwatch is divided into 4 main elements: the counter, the memory, the displays and the control circuit. Added to this is a clock divider, allowing to switch from one clock period of 0.02µs (50Mhz in the simulation) to 0.1 s. Some of the modules consist only of links between several basic modules (eg: stopwatch, counter 1000, and control pulse).

The modules:

Clock Divider (div.vhd)

The clock divider lets you switch from the system clock of 50MHz to a clock directly usable for the counter, 10Hz.

Display module (display.vhd)

The display module converts an integer (on 4 bits) into a 7-bit vector, each representing a segment of the display.

```
process(E)
begin
      if(E = "0000") then
            H <= "0111111";
      elsif(E = "0001") then
            H <= "0000110";
      [...]
      else
            H <= "1111001";
      end if;
end process;
HEX <= not H;
```

Counter 10 (count10.vhd)

```
process(clk)
begin
	if(clk'event and clk = '1') then
		if(nclear = '0') then
			compt <= 0;
		elsif(EN = '1') then
			if(compt = 9) then
				compt <= 0;
			else
				compt <= compt + 1;
			end if;
		end if;
	end if;
end process;
```

The counter by 10 stores in a signal (count) its value. At every clock cycle, and if the input EN is active (active input high), the signal is incremented. After 9, the counter returns to 0. It has an synchronous input initialization: nclear1.

comparEN.vhd:

When a counter by 10 goes from 9 to 0, this module sends a signal to the input EN of the next counter. This allows to activate (briefly) the counter of "tens" (seconds) when the one of units (tenths of a second) goes from 9 to 0, and likewise with the hundreds with the passage of tens.

Counter 1000 (counter.vhd)

I created the counter module by 1000 from previously created modules. The signals EN1 and EN2 are necessary in order to pass the information of the output of the comparator modules at the input of the counter 10 following.

```
signal EN1, EN2 : std_logic;
begin
cpt <= S;
U0 : div port map(clk_in, clock);
U1 : compt10 port map(S(3 downto 0), nclear, EN, clock);
U2 : comparEN port map(EN, S(3 downto 0), EN1);
U3 : compt10 port map(S(7 downto 4), nclear, EN1, clock);
U4 : comparEN port map(EN1, S(7 downto 4), EN2);
U5 : compt10 port map(S(11 downto 8), nclear, EN2, clock);
end arch;
```

Memory (memory.vhd)

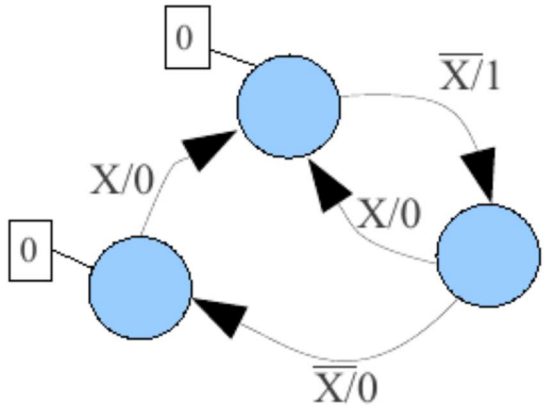
The memory module stores on 12 bits the content sent as input when nload is 0 (synchronous input active low).

Stopwatch (stopwatch.vhd)

The stopwatch only implements and interconnects the different modules explained previously. The memory is connected at the input to the output of the counter and at the output to the different displays. In this way, you can pause the display while continuing to increment the counter.

1Impulse (1impulse.vhd)

This module allows to limit to a single clock cycle a button press. This allows to avoid that the command module starts going too fast. If the user presses the button for too long (more than one clock cycle ie more than 0.1 seconds).
Below is the implementation of the Mealy machine in the form of 2 processes.
The first modifies the outputs and stores them in a next status signal of the machine. The second move to each subsequent state.



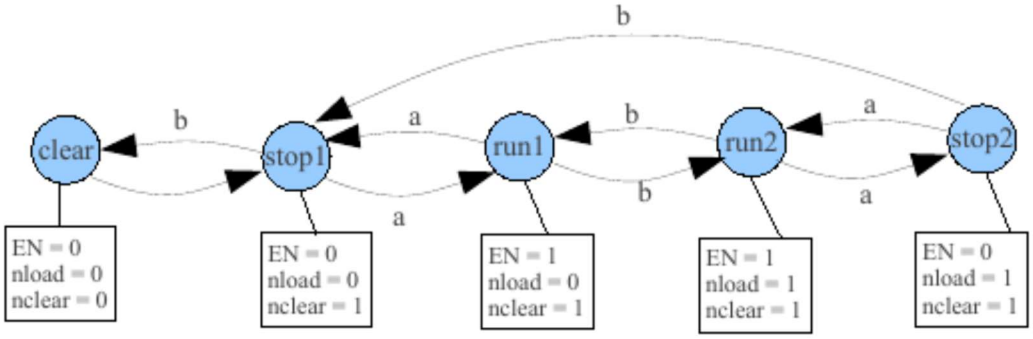

```
entity one_impulse is
	port(
		clk_in : in std_logic;
		E : in std_logic;
		S : out std_logic
	);
end entity;

architecture a of one_impulse is
type state is (X0,X1,X2);
signal current, nexti : state := X0;
begin
process(E, current)
begin
	-- combinatory system :we have to define the output every time
	case current is
		when X0 => if E = '0' then
					S <= '1';
					nexti <= X1;
				else
					S <= '0';
					nexti <= X0;
				end if;
		when X1 =>  S <= '0';   --when we are at state x1, output is 0
				if E = '1' then
					nexti <= X0;
				else
					nexti <= X2;    --if input always active, output is state X2
				end if;
		when X2 => S <= '0';

				if E = '1' then
					nexti <= X0;
```

```
                                        else
                                                nexti <= X2;
                                        end if;
        end case;

end process;
process(clk_in)
begin

        if clk_in'event and clk_in = '1' then
                current <= nexti;
        end if;
end process;

end a;
```

Command circuit: (cmd.vhd)

The control circuit is also defined by a graph, but this time it is a Moore's machine, where the outputs depend on the current state and no longer on the transition. In that case, a 3-process implementation seemed simpler.

- The first state defines the next state depending on the current state and variables.
- The second goes from one state to the next at each clock cycle.
- The 3rd defines the outputs according to the current state (see below)




```
process(current, a, b)
begin
        case current is
                when clear => nexti <= stop1;
                when stop1 => if a = '1' then
                                                        nexti <= run1;
                                                elsif b = '1' then
                                                        nexti <= clear;
                                                else
                                                        nexti <= stop1;
                                                end if;
                when run1 => if a = '1' then
                                                        nexti <= stop1;
                                                elsif b = '1' then
                                                        nexti <= run2;
                                                else
                                                        nexti <= run1;
                                                end if;
                when run2 => if a = '1' then
                                                        nexti <= stop2;
                                                elsif b = '1' then
                                                        nexti <= run1;
                                                else
                                                        nexti <= run2;
                                                end if;
                when stop2 => if a = '1' then
                                                        nexti <= run2;
                                                elsif b = '1' then
                                                        nexti <= stop1;
                                                else
                                                        nexti <= stop2;
                                                end if;

        end case;
end process;
```

CmdbyImpulse.vhd :

This module simply applies the pulse generator previously seen to the two inputs (pb1 and pb2) of the control circuit that it implements.

Top (final_result.vhd):

The global module connects the stopwatch and the pulse control module. A clock divider component had to be added in order to also reduce the clock signal used for the impulses.

```
U0 : stopwatch port map(clk_in, nclear, nload, EN, HEX0, HEX1, HEX2);
U1 : cmdbyimpulse port map(clkDiv, pb1, pb2, nclear, EN, nload);
U3 : div port map(clk_in, clkDiv);
end a;
```

Encountered difficulties and overall conclusion of the project:

A lot of the errors I encountered during the project compilations were related to syntax errors. I tested almost every module with simple LEDs during lab sessions to avoid having too much debugging to do.

Regarding the display, I also forgot that the segments were active in the low state. That's why I added a line HEX <= not H; in the display module outside the process.

When linking the main modules, I was confronted with a problem of synchronization between the pulse control sequencer and the stopwatch. Indeed, I did not think about applying the clock divider to the input of the control circuit, which generated pulses too short to be captured by the stopwatch. So, I created a new instance of the divider in the general program (final_result.vhd). One possible optimization would have been to implement this divider only once and to send the modified clock signal directly to the stopwatch. In the current state, the divider is also present inside counter module per 1000 (counter.vhd)

Even though my stopwatch seems fully functional, it is steel quite frustrating not being able to test the final result on the board. This is manly due to me, not realizing the amount of work that all of the modules required. I also believed I aimed a little bit too high by trying to implement the state machines to control the behavior of the stopwatch. Now that I realize everything that needed to be done, I know for sure I wouldn’t have been able to do it on my own in 4 lessons. I also believe that such project is way easier to realize when working in pairs. Unfortunately, my lab partner had personal issues that forced him going back in his home country.

Overall, I found this project very fulfilling because I got to practice my ability to code in VHDL and I think it will be very useful either for my last year of study in my home university.