DSD EXERCISE

SEQUENTIAL DESIGNS IN VHDL (THE WATCH EXERCISE)

AARHUS UNIVERSITY
SCHOOL OF ENGINEERNG
PHM, CEF

JUNE 2018

Document History

2015-10-26: PHM, Initial version.
2017-10-18: CEF, Fixed exercise numbering.
2018-02-23: CEF, Converted to LaTeX.
2018-06-07: CEF, Fixed figure placements.
2019-03-15: CEF, Added Peer Feedback to J2 exe.
2019-04-12: CEF, Updated Peer Feedback text.
2019-11-12: CEF, Removed Peer Feedback text.
2020-03-13: CEF, Re-added Peer Feedback text.

Goals

In this exercise we'll finally get to play with clock controlled designs, also known as sequential designs. Most FPGA designs use a clock-controlled approach. This is required for many applications and it gives us means to control and manage the internal delays. Long chains of combinatorial logic can result in very long and unpredictable delays. By using clocks and flip-flops, we can break these chains into smaller sections and thereby reduce these problems to create safer and better designs. The goals for this exercise are:

- To get basic experience with using clocks and clocked processes.
- Learn to create counters.
- Get experience with designs combining sequential and combinatorial logic.

In these exercises you will get to create your first digital alarm clock written in VHDL!!! You may even add a buzzer! We'll first start out with a simple counter to count seconds, minutes and hours though.

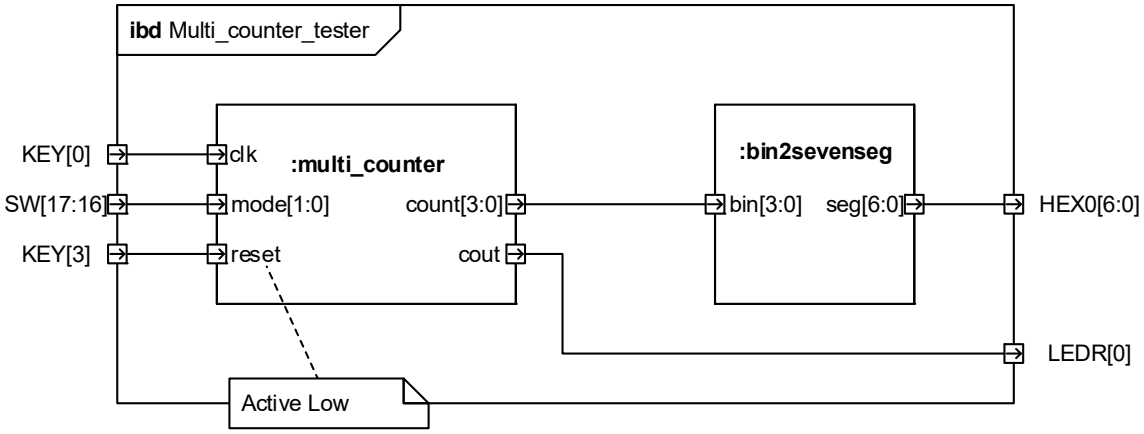

Fig. 1: *Multi counter.*

1 Counter - One Digit

Your very first VHDL counter! The counter must have different modes, so it can be used to count seconds, minutes and hours. The clock input is just a tactile button for now.

a) Design a binary circular counter (returns to zero after reaching maximum value) that increments on every positive edge of the clock, see figure 1 Display the current counter value on the 7-segment display. Use the 7-segment decoder component from former exercises.

- clk, when positive edge is applied the binary counter are incremented by one.
- reset, asynchronous reset. When '0' is applied: reset counter value to 0.
- mode, when:
  - "00" count from 0-9.
  - "01" count from 0-5.
  - "10" or "11" count from 0-2.
- count, this output is the current binary counter value.
- cout, set output to '1' when the current counter turns to zero - otherwise set output to '0'.
- seg, display current counter value on the 7-segment display.

b) Create a functional simulation of the `multi_counter`. How does it react to changes in "mode"? How does cout work?

c) Download and test the design on the DE2 board.

2 Clock - One Digit

Now we'll use the 50MHz crystal oscillator on the DE2-board to generate a real clock signal. We will use this to drive our counter from last exercise step.

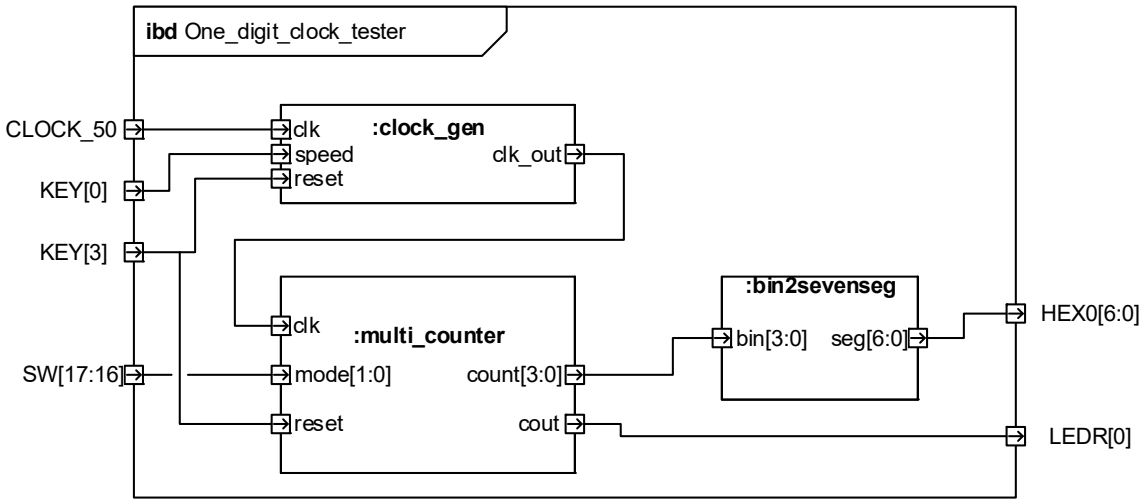

Fig. 2: *One digit clock.*

a) Design a new `clock_gen` component that generates a pulse every second when "speed" is '1' and every 5 millisecond when "speed" is '0'. The pulse must be one 50MHz clock pulse wide. Connect the `clk_out` output of `clock_gen` to the "clk" input of the `multi_counter` component as shown in figure 2.

`CLOCK_50` is a 50MHz, 50% duty-cycle signal generated by a clock oscillator on the DE2-board. The reset signal must be an active low asynchronous reset.

HINT: use a variable integer type in the `clock_gen` component for the counter and set `clk_out` high every time the counter reaches 50.000.000 - otherwise the `clk_out` should be set to low

b) Download and test the design on the DE2 board. Does it output as expected? Does reset work correctly?3)

3 Clock - Six Digit

To get a 24-hour clock, we'll extend it to six digits.

a) Create a new design entity, watch, move the design from step 2 into it and extend the design to 6 cascade coupled instances of the `multi_counter` component, so it will end up as a 6 digit 24 hours clock (eg.. 23 : 59 : 59). Refer to the figure 3 for design inspiration. Use the "count" values, from the hour counters, to control when to reset all counters to 00:00:00 (when they hit 24).

b) Create a tester and instantiate the watch in it and connect it as shown in figure 4. Leave the time, "tm", ports on the watch open for now, by using the "open" keyword (`tm=>open`). Download and test the design on the DE2 board. The output should be a watch as shown in figure 5.

4 Alarm Watch

Let's add an alarm by comparing the watch output with a predefined alarm time.

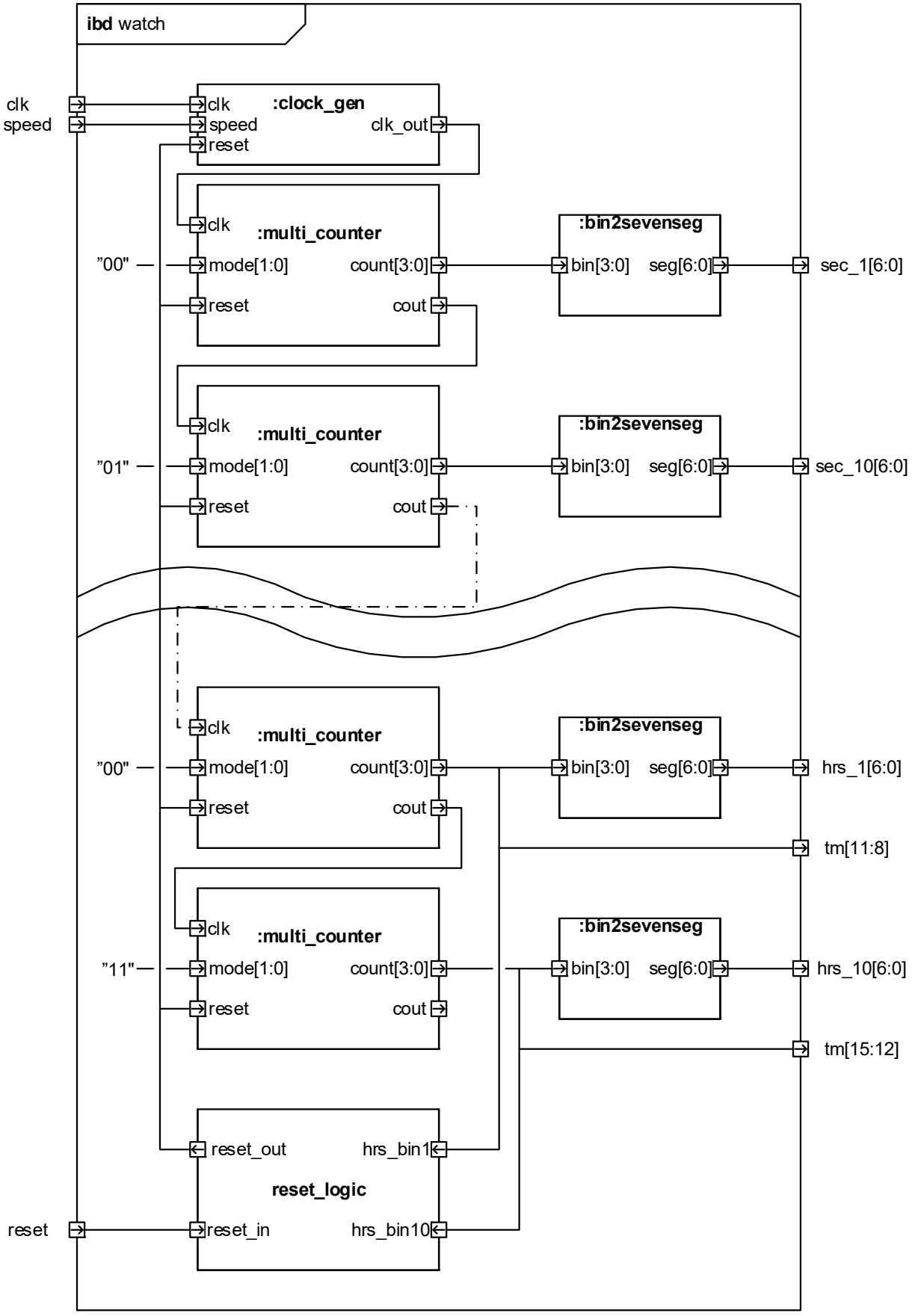

Fig. 3: *Watch IBD (minutes not shown).*

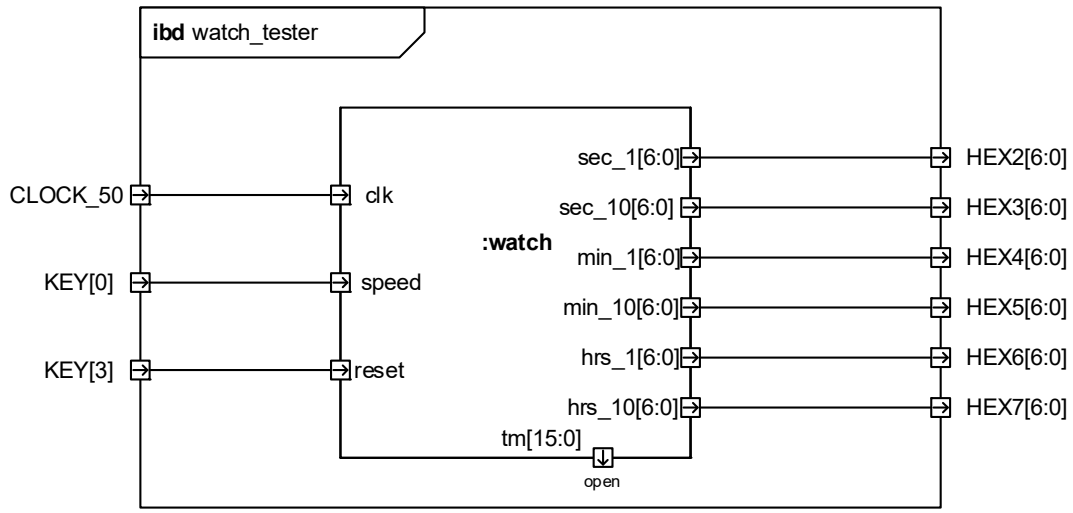

Fig. 4: *Watch tester.*

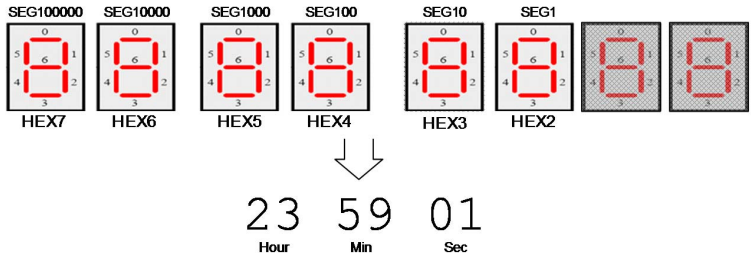

Fig. 5: *Watch output.*

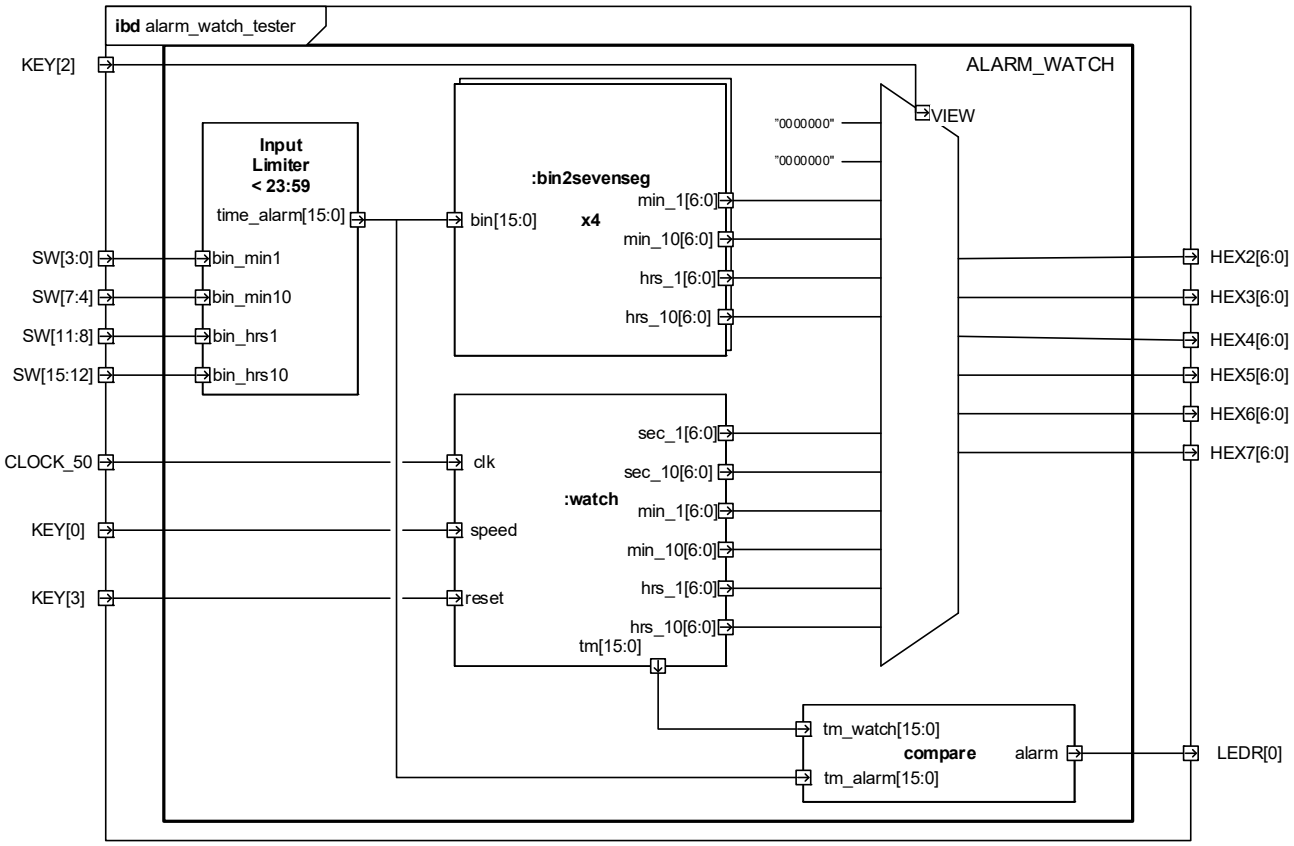

Fig. 6: *Alarm clock.*

a) Extend the watch component with an alarm feature with a one minute resolution (meaning: only hours and minutes can be set). Use the time output from the counters to detect an alarm. A roughly sketched drawing for inspiration is shown in figure 6.

b) Update the tester and instantiate (as shown in figure 6). Download and test the design on the DE2 board.

5 Alarm Watch with Buzzer (OPTIONAL)

Extend the design with a more advanced alarm output.

a) If an alarm occurs, the speaker plays an approximately 400Hz sound for an interval of approx. 700 ms and then silence for approx. 700 ms. Repeat this pattern until the one minute alarm period is finished. Use GPIO 0 connector pin 1 (`PIN_D25`) for the red wire and pin 12 (GND) for the black wire. Why it is not so easy to choose an arbitrary frequency?

b) Download and test the design on the DE2 board.

6 Peer Feedback: Journal 2

You need to document your feedback *to* the two other groups, you have reviewed (tree reviews in case of a review-group with four DSD groups).

a) Insert the data from your two(/three) review-spreadsheets (from `J2_rubric.xlsx` for the feedback you have given) with feedback points and comments into the journal in text or table form.

b) If you meet the J2 deadline: also insert the feedback(s), you have received, into this journal.

If you have not received two(/three) journals to review from the other groups after the deadline, then document here, that you have contacted the course-lecture about the problem (insert email text).