Politecnico di Torino

DIGITAL SYSTEMS ELECTRONICS

04OIHNX - A.A. 2024/2025

Prof. G. Masera

Laboratory assignment no. 4 – Flip-flops and counters

DUE DATE: 08/04/2025 DELIVERY DATE: 08/04/2025

GROUP 08 - Contributions:

- Daniele Becchero (308299) – 33.3%
- Bohotici Ionut Viorel (300061) – 33.3%
- Simone Viola (310779) – 33.3%

The members of the group listed above declare under their own responsibility that no part of this document has been copied from other documents and that the associated code is original and have been developed expressly for the assigned project.

Introduction

This laboratory experience explores the fundamental building blocks of digital systems: latches, flip-flops, and counters. These components are the simpler example of sequential circuits, where the output depends not only on the current input but also on the history of previous states. Through this practical session, we will gain hands-on experience in designing and simulating these essential circuits using VHDL.

The experiment begins with the implementation of a latch, the simplest form of memory elements, to understand their basic operation and limitations. Building on this foundation, flip-flops are introduced as a more robust and synchronous alternative, enabling precise control over state transitions with a clock signal. Finally, counters are designed to demonstrate sequential operations, showcasing their applications in tasks such as frequency division, event counting, and timing.

The last section of this report is about a real-world application of a sequential circuit.

Section 1: Gated SR-latch

The first section of this laboratory experience is about the design of a gated SR-latch.

Delivered files

Two VHDL files have been delivered for the description of the gated SR latch and its simulation:

- *gated_SRlatch.vhd*, VHDL code for the top-level entity, which acts as the gated SR latch
- *gated_SRlatch_tb.vhd*, VHDL code for the testbench used to validate the functionality of the gated SR latch

Design entry

In this case, the VHDL code *gated_SRlatch.vhd* is the one provided in the assignment and it has not been divided into subcomponents, as it was implemented using simple and short lines of code with AND and XOR gates, written using the VHDL statement `not (R_g or Qb)`, and `not (S_g or Qa)`.

Next, we focus on the specifics, analyzing the individual blocks to be implemented and exploring the possible synthesis methods for the circuit. As illustrated in Figure 1, the circuit of a gated SR-latch serves as the foundation for this analysis and synthesis. The goal is to examine its behaviour and functionality through implementation using VHDL code.

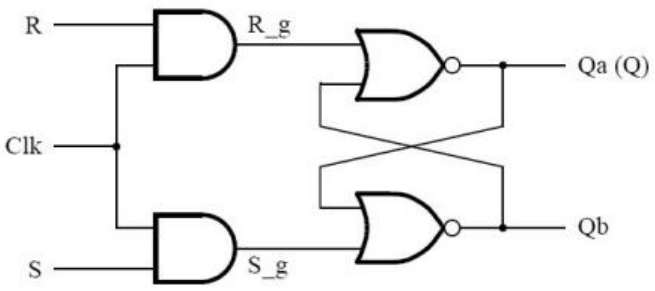

Figure 1 - Gated SR-latch circuit

The truth table of the following circuit is shown below.

Clk (clock)	S (Set)	R (reset)	Qb	Qa	Condition
1	0	1	1	0	*Reset*
1	1	0	0	1	*Set*
1	0	0	Qb	Qa	*memory*
1	1	1	0	0	*(“forbidden”)*
0	-	-	Qb	Qa	*memory*

Table 1 -Truth table for a gated SR-latch

The behavioural implementation of the latch using a process is trivial, but it does not provide access to its internal signals, such as R_g and S_g, making them unobservable. To ensure these signals can be observed and to retain the structure depicted in Figure 1, a compiler directive must be included in the code. As shown in Figure 2, the keep directive is added via a VHDL `ATTRIBUTE` statement, instructing the Quartus Prime compiler to allocate separate logic elements for each signal (R_g, S_g, Qa, and Qb). Following code compilation, the tool generates the circuit with four distinct 4-LUTs—one for each gate—as illustrated in Figure 3.

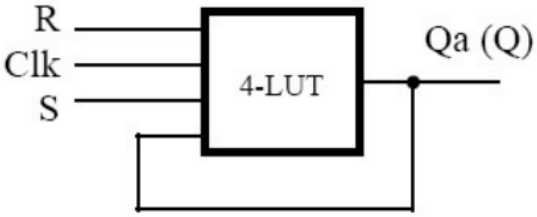

Figure 2 - SR-latch using one 4-input LUT

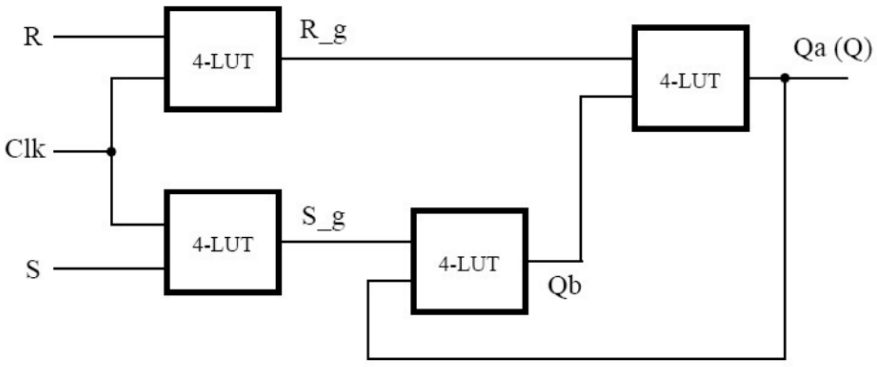

Figure 3 - Implementation of the SR-latch of Figure 1

Functional simulation

To verify the functionality of the code that implements this gated SR-latch using four 4-LUTs, a VHDL testbench code needs to be simulated on ModelSim, sequentially changing the states at time intervals of 10 ns. To fully ensure correct operation, the input states were not simulated just once but numerous times, confirming the reliability of the implementation after several iterations, resets, and memory operations. For each combination of R and S input states

simulated, a sequence of 4 clock cycles (0-1-0-1) is performed to verify when the device switches. The table of simulated states in chronological order is as follows.

S	R	Clk	Q	Comment
0	0	0-1-0-1	*Indeterminate (no previous value)*	Q retains previous value (1 or 0). In this case, there is no previous value, so it is indeterminate.
0	1	0-1-0-1	0	RESET test
1	0	0-1-0-1	1	SET test
0	0	0-1-0-1	1 *(previous value)*	HOLD state
0	1	0-1-0-1	0	Repeat RESET test
1	0	0-1-0-1	1	Repeat SET test
1	1	0-1-0-1	*Forbidden (invalid state)*	Test invalid state

Table 2 - SR-latch simulated states

The waveform screen capture is shown in Figure 4.

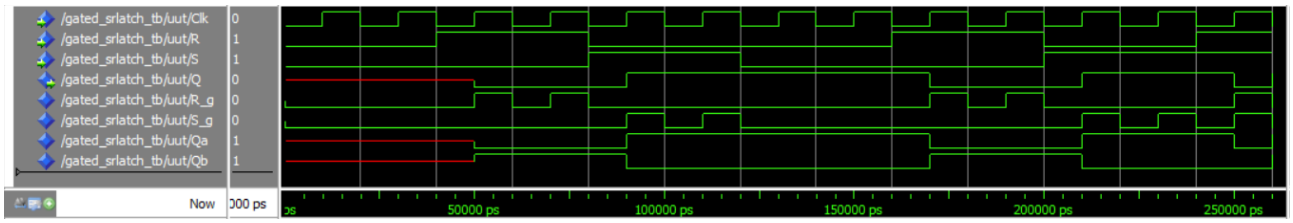

Figure 4 - Simulation results of SR-latch

As expected, the gated SR latch works properly, and the outputs match the combinations described in Table 2.

Synthesis & Register Transfer Level (RTL) Viewer

After creating a project in Quartus Prime and successfully compiling and synthesized the code, we can proceed by examining the gate-level circuit generated from the VHDL code, using *Tools -> Netlist Viewers -> RTL Viewer*.

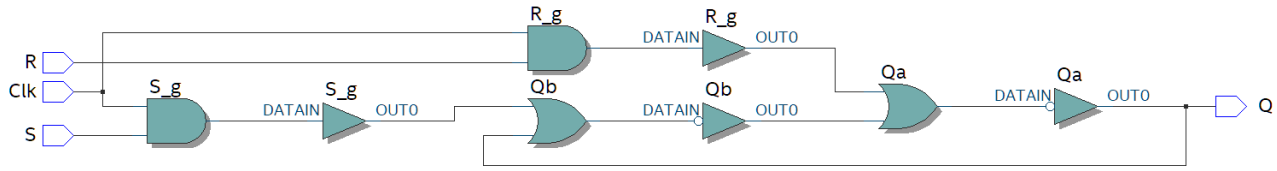

Figure 5 - RTL viewer result for SR-latch

Using “Technology Map Viewer Post-Mapping”, we can check also the correct implementation of the circuit we described, which returns us the architecture in Figure 6.

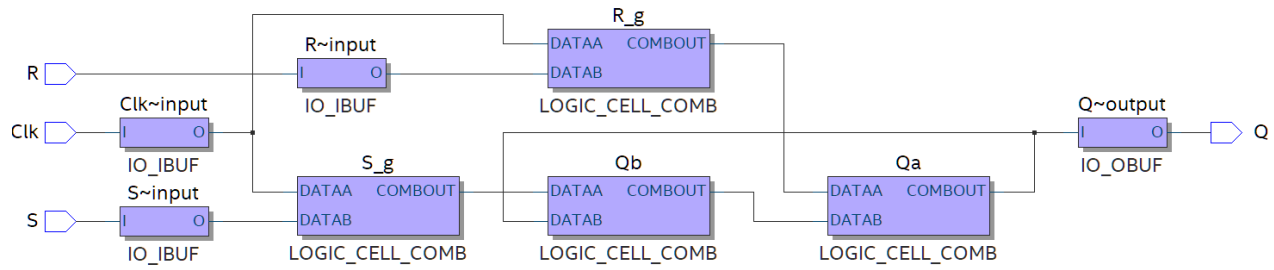

Figure 6 - Technology map viewer post-mapping results for SR-latch

A similar architecture can be observed using *"Technology Map Viewer Post-Fitting"*.

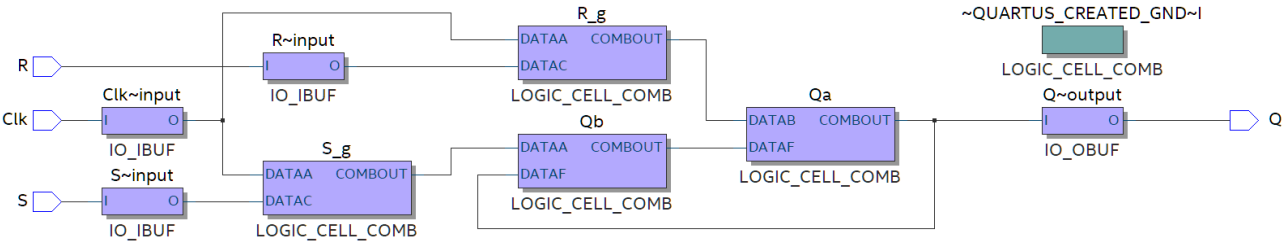

Figure 7 - Technology map viewer post-fitting result of SR-latch

The RTL display setting has proven particularly useful for verifying that the written code works according to the desired architecture.

Section 2: Structural 16-bit synchronous counter

This section covers the VHDL implementation of a 16-bit synchronous counter described with a structural approach, based on sixteen T-type flip-flops and some logic gates.

Delivered files

Five VHDL files have been delivered for the description of the structural 16-bit counter and its simulation:

- *device_counter16bit.vhd*: VHDL code for the top-level entity that connects the 16-bit structural counter with the 7-segment display decoder and the FPGA onboard devices.
- *counter16bit_struc.vhd*: VHDL code for a structural 16-bit counter implemented by using some T-FlipFlops and logic gates.
- *T_FlipFlop.vhd:* VHDL code for a T-type flip flop.
- *hexDecoder.vhd*: VHDL code for a binary to 7-segment display decoder.
- *counter16bit_struc_tb*.vhd: VHDL code for the testbench used to validate the functionality of the *counter16bit_struc*.

Design entry

In this section, the 16-bit structural counter must be designed by using a structural approach. The assignment provides an example for a 4-bit structural counter that involves four T-type flip-flops and three logic gates. For a 16-bit counter, the number of flip-flops rises to 16. They are arranged in the same way as the 4-bit counter, with each flip-flop connected to the previous one as shown in Figure 8.

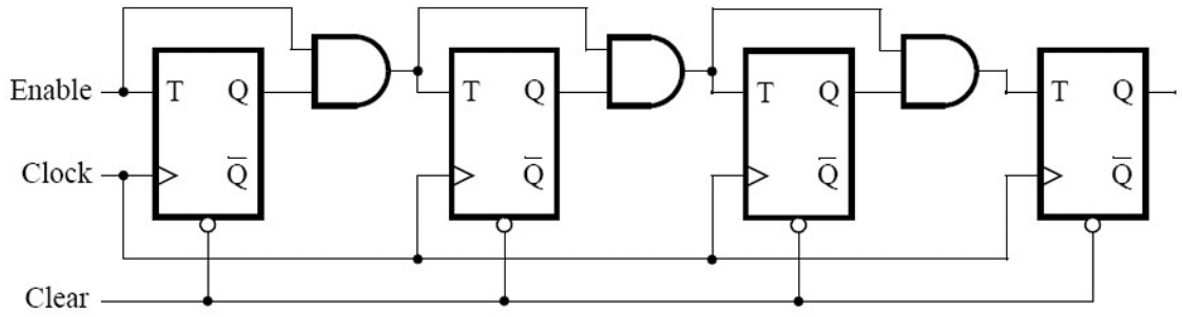

Figure 8 - 4-bit counter using T-type flip-flops

For this exercise is asked to design a fully synchronous circuit, which means that the clock signal must be the same for each flip flop.

16-bit counter (structural architecture)

The 16-bit structural counter is the top-level entity of this section. The architecture is made by sixteen blocks which contains one T-type flip-flops and one elementary logic AND gate. For a generic N-bit counter, the number of blocks to be implemented is given by the following formula.

$$\#Blocks = \lceil \log_2 N \rceil$$

To simplify the VHDL model of the counter, the four blocks have been generated with a `for … generate` statement, as shown.

```
TFF : T_FlipFlop
port map(
     CLK => CLK,
     CLR => CLR,
     T => EN,
     Q => C(0)
);

ff_generate : for i in 1 to 15 generate
     A(i) <= T(i - 1);
     T(i) <= A(i) and C(i - 1);
     TFF : T_FlipFlop
     port map(
          CLK => CLK,
          CLR => CLR,
          T => T(i),
          Q => C(i)
     );
end generate ff_generate;
```

As you can see from the code, to generate 16 of these components in cascade, the first one was initialized separately to handle correctly the indices in the `generate` statement. Then, using a for loop from 1 to 15, the remaining 15 T-flip flops were generated and connected. The elementary building block of the counter is shown in Figure 9.

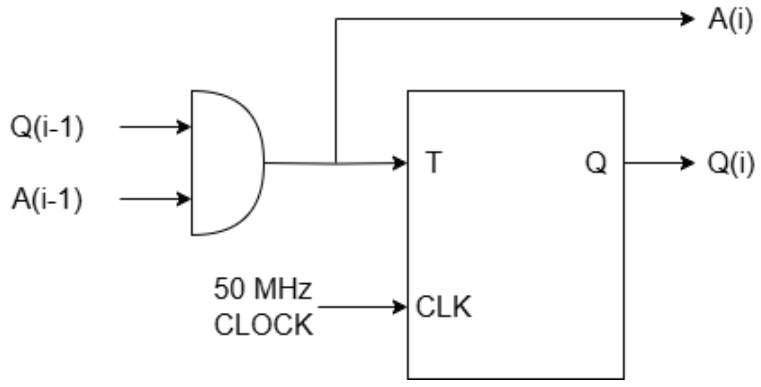

Figure 9 - Elementary building block of a N-bit counter

T-FlipFlop

A T-type flip-flop is a digital circuit that continuously changes its state at each positive clock edge when the input T is high. The resulting output is represented by the Q signal.

Its implementation in VHDL has been made by using a process.

```
process (CLR, CLK)
     begin
          if (CLR = '0') then
               t_tmp <= '0';
          elsif (rising_edge(CLK)) then
               t_tmp <= T xor t_tmp;
          end if;
     end process;
Q <= t_tmp;
```

This component features a clock input CLK, an active-low asynchronous clear input (CLR), the toggle input (T), and the output Q. The operation of the flip-flop is described using a process that is sensitive to both the clear signal and the clock signal. The clear input (CLR) was intentionally designed to be asynchronous, as we believe it should be capable of acting at every moment, independently of the clock signal.

Functional simulation

The file delivered which describes the testbench, necessary for the functional simulation, is named *counter16bit_struc_tb.vhd*. To ensure proper functional simulation of the counter in ModelSim, the testbench was written so that, at the initial time instant, the signals EN, CLK and CLR are set to a logic low value (0). Subsequently, all three signals are driven high (1), and the CLK signal is then toggled between '0' and '1'.

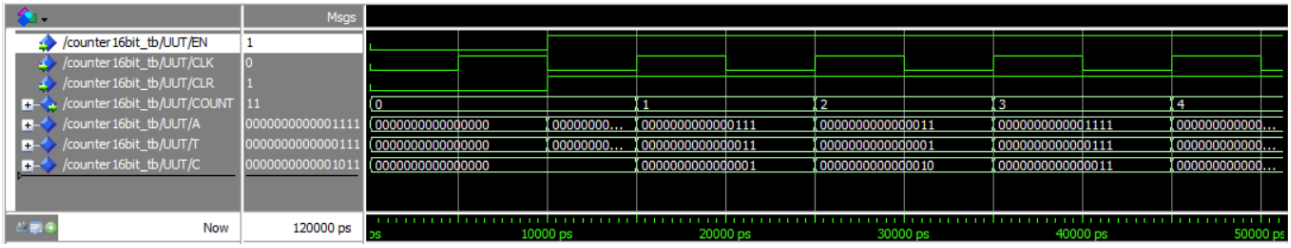

Figure 10 - 16-bit structural counter simulation results

As shown in the waveforms, the counter operates correctly and increments its value on each rising edge of the CLK signal. The COUNT output is shown in decimal format.

Synthesis and RTL Viewer

After importing the VHDL file into Quartus Prime project, we included the *device_counter16bit_struc* that contains the assignments between the counter and the push button KEY0 as the Clock input, switches SW1 and SW0 as Enable and Reset inputs respectively, and the 7-segment displays HEX3-0 to display the hexadecimal count as the circuit operates. Once the circuit is compiled and synthesized the circuit, we can proceed by

examining the gate-level circuit generated from the code, using *Tools-> Netlist Viewers -> RTL Viewer:*

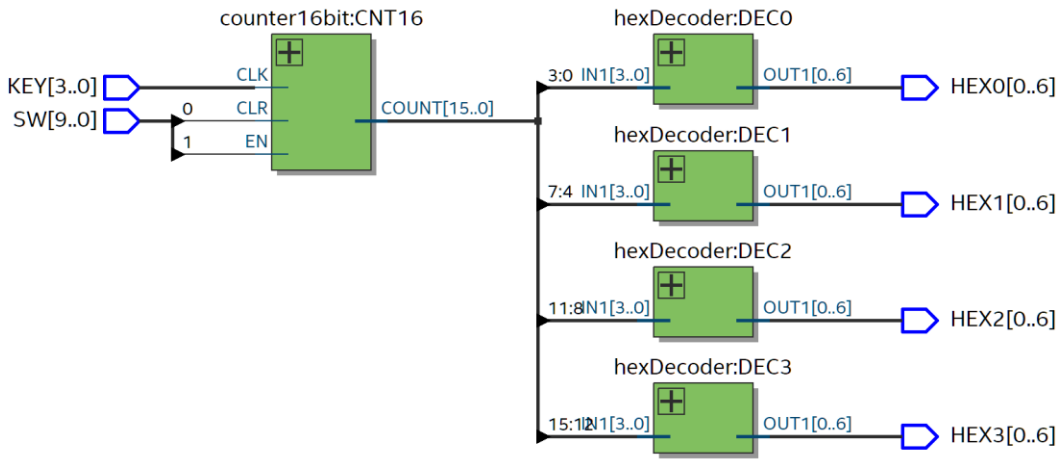

Figure 11 - RTL viewer results for 16-bit structural counter

Each block is composed of an internal sub-circuit. In particular, the circuit generated from the code for the *counter16bit* block is shown in Figure 12. The generation of all 16 flip-flops and their interconnections through logic gates can be easily observed.

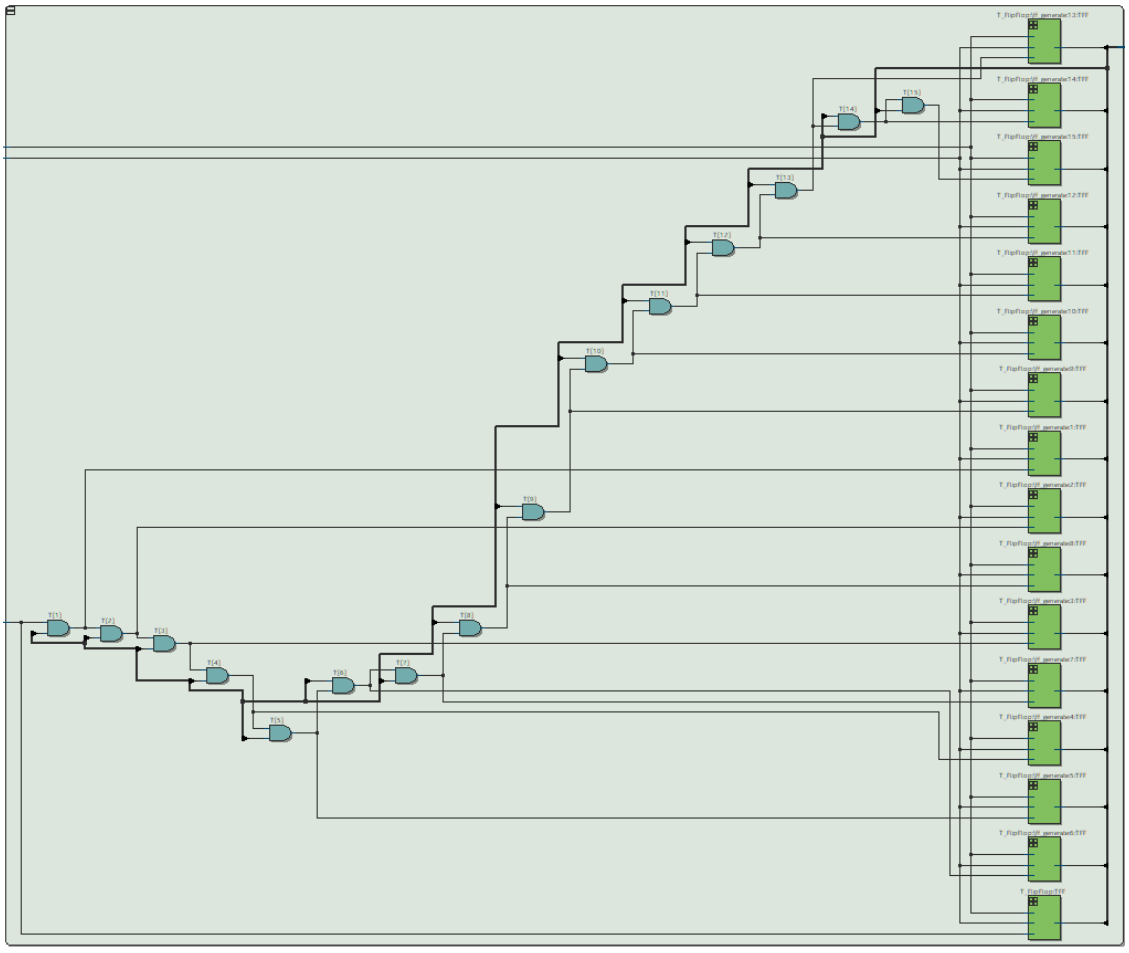

Figure 12 - Circuit generated by Quartus Prime

Looking even deeper, we can inspect the internal structure of each generated T-type flip-flop by clicking on one of the green blocks on the right.

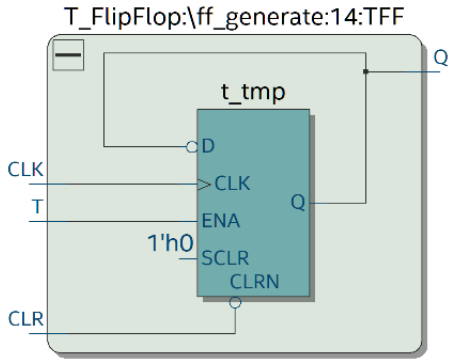

Figure 13 - Internal structure of T-type flip-flop

As expected, from the overall circuit view down to the structural level of the individual components, the code correctly generates the desired circuit to implement a 16-bit counter in a structural way.

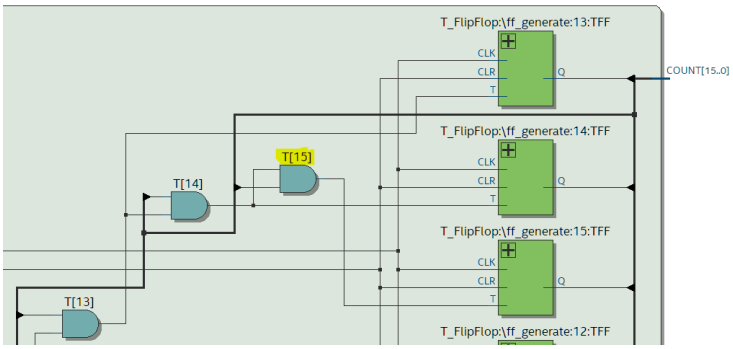

Figure 14 - Detailed view of the number of logic elements used

As shown in the detailed view above, the implementation of this counter requires 31 logic elements (LE) divided in 16 flip-flops and 15 AND gates (notice that the first flip-flop is starting from index 0). Moreover, the maximum operating frequency (Fmax) estimated by the Quartus Prime timing analysis is approximately 493 MHz, that is reasonable and coherent with expectations.

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	492.61 MHz	492.61 MHz	KEY[0]	

Figure 15 - Maximum operating frequency of 16-bit structural counter

Section 3: Behavioural 16-bit synchronous counter

The objective of this exercise is to design and implement a 16-bit counter that replicates the exact functionality of the counter described in the previous section. However, this implementation will focus on a behavioural approach instead of the structural methodology employed earlier. This enables a comparative analysis of the two approaches in terms of computational complexity, code length, and performance, based on an evaluation of the maximum operating frequency.

Delivered files

Two VHDL files have been delivered for the description of the behavioural 16-bit synchronous counter and its simulation:

- *counter16bit_behav.vhd*: VHDL code for a behavioural 16-bit counter simply based on the VHDL statement

```
Q <= Q + 1;
```

- *counter16bit_behav_tb.vhd*: VHDL code for the testbench used to validate the functionality of the *counter16bit_behav*

Design entry

Since this behavioural version does not require implementing the code using SW, HEX or KEY on the DE1-SoC board, the VHDL files have been structured specifically for the simulation in ModelSim to verify its functionality, and for synthesis in Quartus to perform compilation, RTL viewer, and timing analysis.

As result, the decoder code for the 7-segment displays has been omitted. Only a single VHDL file remains, implementing the simple VHDL statement `Q <= Q + 1`, along with the corresponding testbench file used in ModelSim.

Functional simulation

The provided testbench file, named *counter16bit_behav_tb.vhd*, is essential for the functional simulation of the counter in ModelSim. To ensure correct counter functionality, the testbench initializes the signals EN, CLK, and CLR to a logic low state (0) at the starting time instant. Following this, the CLK signal is alternately toggled between 0 and 1 multiple times.

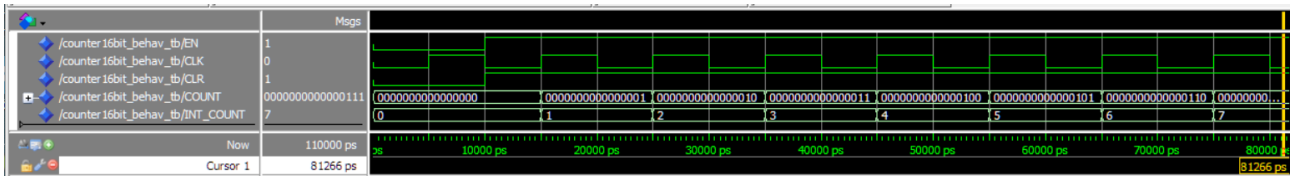

Figure 16 - Simulation results for 16-bit behavioural counter

Synthesis

After successfully compiling the design in Quartus, the RTL viewer can be utilized to inspect the resulting circuit generated by the behavioural version. This allows for a direct comparison with the structural version, providing valuable insights into their differences and functionality.

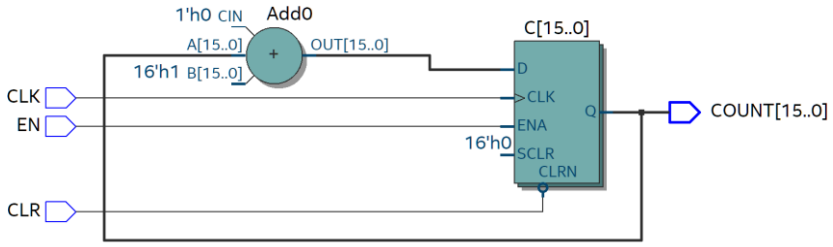

Figure 17 - RTL viewer of the 16-bit behavioural counter

We observe that this behavioural version is conceptually simpler, as is the VHDL code, but the performances are lower because a structural approach allows for a deeper optimization of the digital circuit. One demonstration of this is the maximum operating frequency estimated by the Quartus timing analysis, which can be seen below: when compared to the structural approach version, the maximum frequency in this case is lower, which means that the hardware implementation is more complex and therefore the performance is worse. The maximum operating frequency, also in this case, is reasonable and coherent with expectations.

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	437.83 MHz	437.83 MHz	CLK	

Figure 18 - Timing analysis results for 16-bit behavioural counter

As expected, the maximum operating frequency in the behavioural counter (437,83 MHz) is lower than that of the one of the structural version(492,61 MHz).

Section 4: Flashing digits from 0 to 9

In this section, a circuit was designed and implemented to iteratively display digits from 0 to 9 on the 7-segment display HEX0. Each digit is shown for approximately one second. To determine the one-second interval, a counter was used, incremented by the 50 MHz clock signal provided by the DE1 board (CLOCK_50 input signal, treated as an external input). Once a 1 Hz pulse was derived, it was used to control a second unit responsible for generating the sequence of ten digits to be displayed. It is important to note that no additional clock signals were derived in the design, and the entire circuit was implemented as a fully synchronous system, ensuring that all flip-flops are directly clocked by the 50 MHz clock signal.

Delivered files

Seven VHDL files have been provided to describe the structural implementation of the flashing digits from 0 to 9, along with an additional file dedicated to the testbench simulation:

- *comparator.vhd*, VHDL code for a comparator (type A).
- *digitFlashing.vhd*, VHDL code that links part of the elements used (structural based).
- *elementaryCounter.vhd*, VHDL code for a toggle counter (with load command).
- *flipflop.vhd*, VHDL code for a D-type flip-flop.
- *moduloCounter.vhd*, VHDL code for a modulo counter.
- *mux.vhd*, VHDL file for the 2-to-1 1-bit wide multiplexer.
- *digitFlashing_tb.vhd*, VHDL code used to implement the testbench

Two additional VHDL files have been delivered to implement the digit flashing circuit on the FPGA:

- *device_digitFlashing.vhd*, VHDL code for the top-level entity that defines the structure of the circuit, connecting block together.
- *hexDecoder.vhd*, VHDL code for a 7-segment-display (active low) decoder.

The synthesized *.sof* file for the 5CSEMA5F31C6 device has been delivered in the “Additional files” folder.

Design entry

To implement flashing digits, it is necessary to provide a pulse at a rate of 1 Hz and use it to increment a counter each time. By using the input signal CLOCK_50 available from the DE1-SoC board, and treating it as an external clock signal, we can use a modulo counter to count how many times the clock signal goes high. Since this counter needs to count up to $50 \cdot 10^6$, the next 2’s power greater than that is 2^{26}. For this reason, the counter module muse be (at least) 26-bit wide.

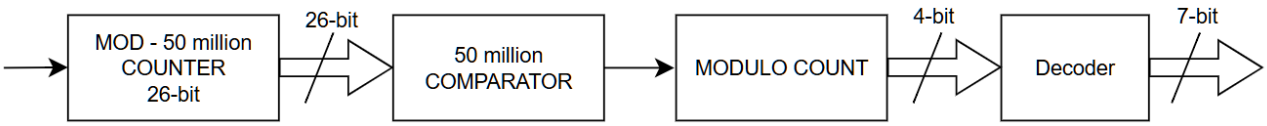

Figure 19 - Circuit structure for flashing digits

At this point, we need to compare the number of clock ticks counted using a comparator to detect when 50 million is reached, thereby identifying the 1 Hz pulse. After that, the output of the comparator will be a single pulse at a frequency of 1 Hz (T = 1 s), which will be the enabling input to a new modulo-10 counter block. This final counter counts how many 1 Hz pulses it receives – corresponding to the number of seconds elapsed – and outputs a 4-bit value (ranging from 0 to 9) to a decoder, which drives a 7-segment display. By analysing in more detail, we can describe and design each block individually. In particular, we will examine the internal structure and a possible implementation of the Modulo Counter.

Modulo counter

This entity is a programmable modulo counter. The modulus and bit-width parameters must be specified at compile-time using generic inputs. This ensures flexibility and allows the design to be easily adapted to different configurations without modifying the core code.

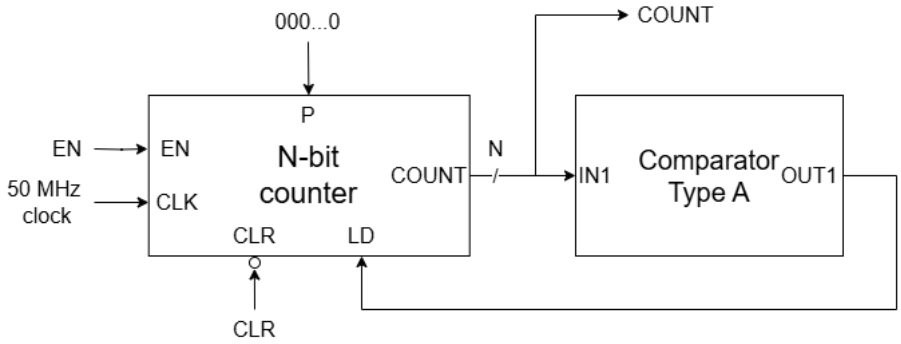

Figure 20 - Parametric modulo counter internal structure

The counter is based on a N-bit counter (declared at compile-time) with load command, which is driven by a comparator that checks if the output value of the counter is equal to another value specified at compile-time. In case of this modulo counter the modulus is 50000000 (50 million), so the comparator checks for the last valid number, which is 49999999 (in binary). The N-bit counter is a component that is made by N elementary-counters, the internal structure of the elementary-counter is shown below. Notice that the *prev_A* and *prev_Q* inputs of the first elementary counter are driven by the enable input of the entire counter.

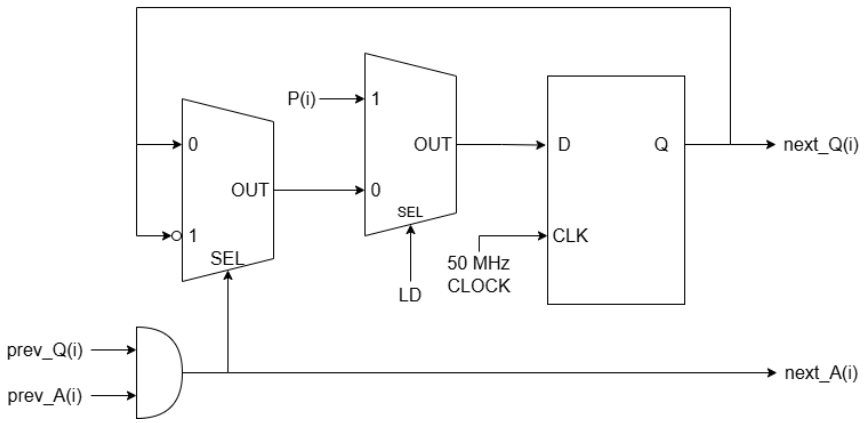

Figure 21 - Elementary building block of the N-bit counter

Comparator Type A & Type B

Two types of binary comparators have been used in this and in the next Section. Both check if the first input is equal to the second, but they differentiate by the second operand of the comparison. It is declared as a generic parameter, defined at compile time in the Comparator Type A, while it is a different input signal in Comparator Type B. For both of them, it is also needed to define the number of input bits of the component by using the `generic map` statement.

Functional simulation

The testbench for this circuit is trivial. The only input is a clock signal, which has been generated inside a process.

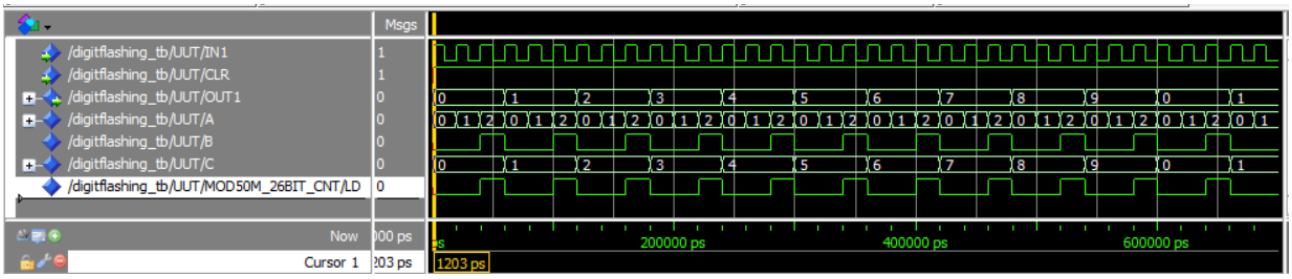

Figure 22 - Simulation results for Flashing digits circuit

The circuit is first reset using the CLR signal, and then it will works normally. For simulation purposes, the 50 million modulo counter has been replaced with a 3-modulo counter in order to speed up the simulation. The signal A is the output of the 3-modulo counter, while B is the output of the comparator. Notice that C (and so the overall output OUT1) is incremented just when B is high on the rising edge of the clock. As expected, the testbench simulation on ModelSim shows that the digit flashing and its related components work correctly, incrementing the count from 0 to 9 and then restarting from 0.

In this simulation of the clock signal is not exactly 50 MHz as requested in real implementation, but this does not affect the validation of the VHDL code, since on the DE1-SoC board the increment rate will be just different, but the overall behaviour will be the same (apart from the 3 modulo counter that will be set for 50 million modulo counter).

Synthesis

The flashing digit entity with all its sub-components have been imported as a component in a new Quartus project. This component has been used with the *hexDecoder* in a new top-level entity that includes also the connection between the components and the DE1-SoC onboard devices.

To show the performance on the board we have recorded a short video that can be reached at this Dropbox folder.

Section 5: Reaction timer

This section is about the design of a reaction timer circuit that measure how quickly a person responds to a visual stimulus. The setup is essential. It includes an LED that lights up after a wait time, which is selected using eight switches operating in binary. Once the wait time has elapsed, the LED will turn on, and the user must press KEY3 as quickly as possible to stop the timer that shows the reaction time in millisecond.

Delivered files

Eleven VHDL files have been delivered. In details:

- *reactionTimer.vhd*, VHDL file for the timer (main entity)
- *reactionTimer_tb.vhd*, VHDL file for the testbench of the timer
- *moduloCounter.vhd,* VHDL file for a parametric modulo counter
- *counter16bit_struc.vhd,* VHDL file for a 16-bit counter (the same of Section 2)
- *elementaryCounter.vhd,* VHDL file for the elementary component of the 16-bit counter
- *T_FlipFlop.vhd,* VHDL code for T-type flip-flop
- *mux.vhd,* VHDL file for the 2-to-1 1-bit wide multiplexer
- *flipflop.vhd,* VHDL code for D-type flip-flop
- *comparator.vhd,* VHDL code for a comparator (type A)
- *comparator_typeB.vhd,* VHDL code for comparator (type B)
- *gated_SRlatch.vhd,* VHDL code for a SR latch made of logic gates

Another two VHDL files have been delivered to model and test the binary-to-BCD converter.

- *binToBcd.vhd,* VHDL code for the binary-to-BCD converter
- *binToBcd_tb.vhd*, VHDL code for the testbench

Two additional files have been delivered to program the FPGA:

- *device_reactionTimer.vhd*, VHDL file for the main entity (general)
- *hexDecoder.vhd,* VHDL code for the 7-segment display decoder

The synthesized *.sof* file for the 5CSEMA5F31C6 device has been delivered in the “Additional files” folder.

Design entry

Reaction timer

The main entity for the timer is made by some counters, a SR-latch, some logic gates and a modulo counter that generates a 1 kHz pulse. The entity's inputs are the 50 MHz clock, the reset and the timer stop signals and the wait-time in binary, while COUNT is a 16-bit wide output, which is the value of reaction time in millisecond, and OUT2 is the output signal for the LED.

Since the circuit must count the time-interval millisecond, a sort of frequency divider is needed to provide a 1 kHz pulse starting from the 50 MHz on-board clock. The circuit is based on a modulo counter followed by a comparator that checks if the counter's output is equal to 49999 and generates a high value if true. Notice that the comparator needs to check for '50000 -1' because it starts counting from '0'. The output signal of the comparator is a pulse at a rate of 1 kHz. It is important to notice that all the components are driven by the 50 MHz clock signal, and the 1 kHz pulse is used to enable the circuit functionality. As requested, the circuit is fully synchronous.

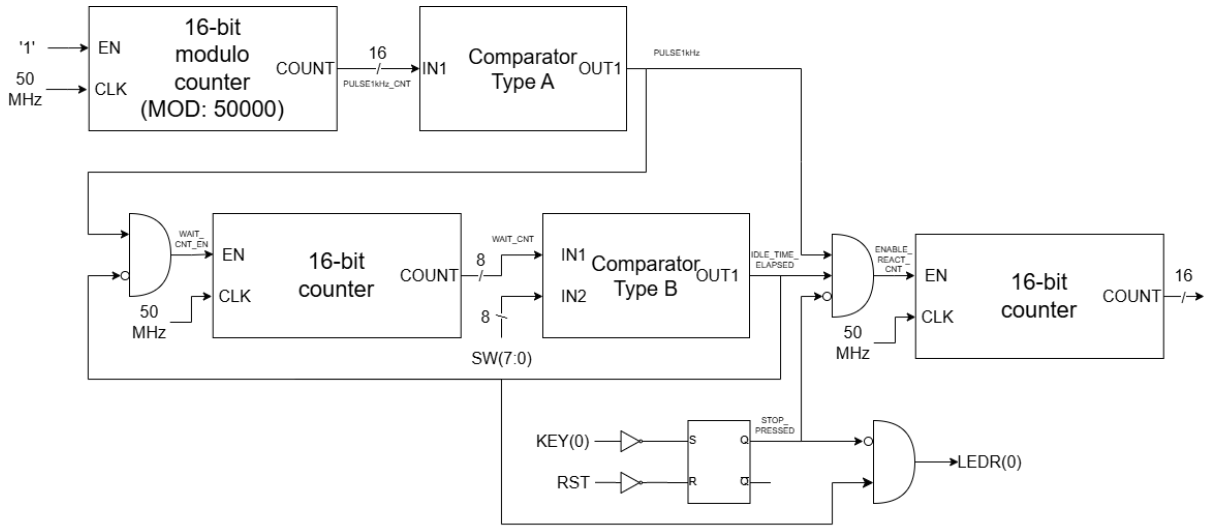

Figure 23 - Reaction timer internal architecture

Once the 1 kHz pulse is available, it is used to feed the "enable" input of the first 16-bit counter, which is used as a timer for the wait phase. The "enable" input of the counter is also controlled by the output of the following comparator that checks if the output value of the counter is equal to the value defined by the switches. Notice that only the first eight LSBs of the counter are needed, because the switches are eight, according to the specifications. A 16-bit counter have been chosen in order to avoid creating another VHDL component with just different number of bits.

When the counter has reached the value defined by the user, the following comparator's output is forced high. This signal disables the "enable" of the first counter and activate the "enable" input of the second counter, which counts the reaction time in millisecond. The increment is stopped when KEY3 is pressed. The change in the status of KEY3 is stored in the

SR-latch, which must be reset at the circuit start-up. The output of the latch, which is high after the STOP button has been pressed, is used to disable the input of the second counter.

Modulo Counter

The Modulo Counter is the same entity already described in *Section 4*, under the *Design Entry* paragraph.

16-bit counter

The counter is equal to the one described in *Section 2*. The counter is made by sixteen T-type flip-flops connected in cascade.

Comparator Type A & Type B

These two types of comparators are already explained in details in *Section 4*, under the *Design Entry* paragraph.

SR-Latch

The SR-latch is the same component described in *Section 1*.

Binary-to-BCD converter

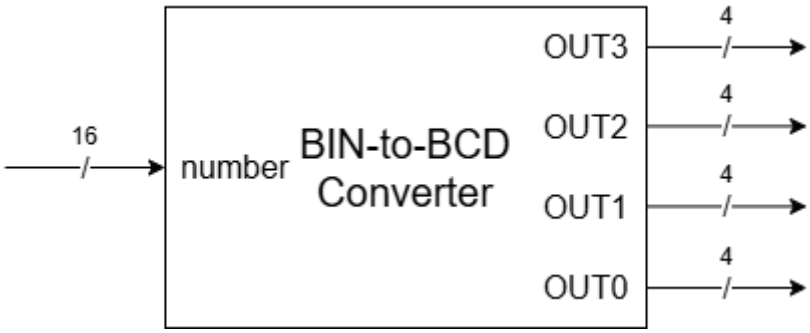

Figure 24 - Binary-to-BCD converter

The binary-to-BCD converter is used to display the reaction time, represented in binary from the *reactionTimer*, on four 7-segment displays. Its architecture relies on a series of `when...else` assignments to determine each digit and process the next one. For instance, with the number 1234, the thousands digit is identified first. The circuit then calculates the remainder by subtracting the largest possible multiple of a thousand from the original number (1234 - 1000 = 234). The hundreds digit is then processed, followed by the tens and ones digits.

Functional simulation

Reaction timer

The validate the VHDL model of the circuit described above, the parametric blocks used inside have been set to work at a lower speed than the operating frequency of the real circuit, to achieve a quicker simulation. The modulo counter and the following comparator generates the “edited 1 kHz” pulse every three clock cycles. The overall functionality is not changed, because in the real implementation the parameters will be set accordingly to generate a pulse at the specified rate.

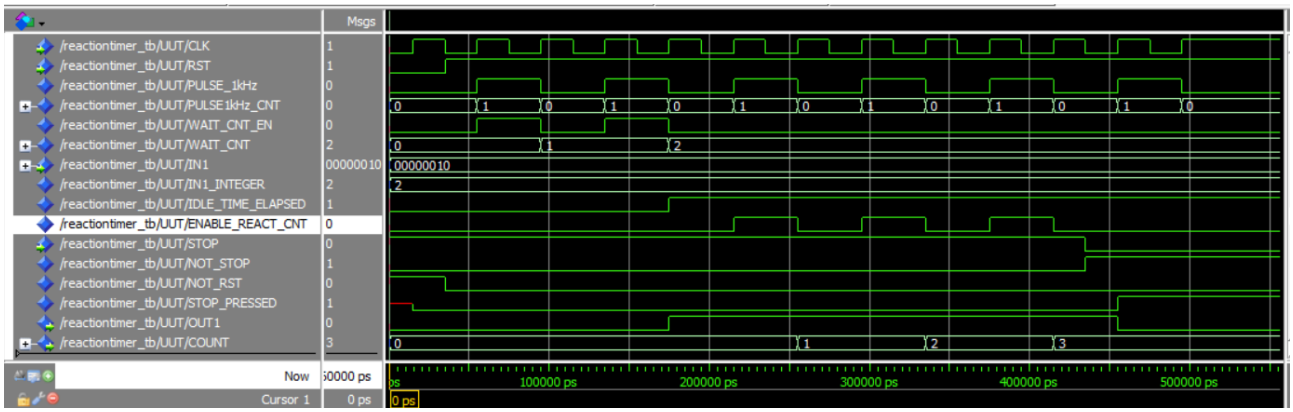

Figure 25 - Simulation results for the Reaction timer circuit

The waveform result shows the clock and reset signals, the output of the first modulo-counter that should provide the 1 kHz pulse and the effective 1 kHz pulse. Then, there is the wait phase counter's output and the following comparator's output, with the wait time set by the signal IN1. After the wait phase has elapsed, the second counter is enabled, again with the 1 kHz pulse. Below, there are the SR-latch signals and the two outputs of the reaction timer. OUT1 will be connected to the LED0, while COUNT will be connected to the binary-to-BCD converter and then to the displays. The intermediate binary signals have been represented in unsigned format by selecting "Unsigned" in the "Radix" menu of the corresponding signal.

Binary-to-BCD converter

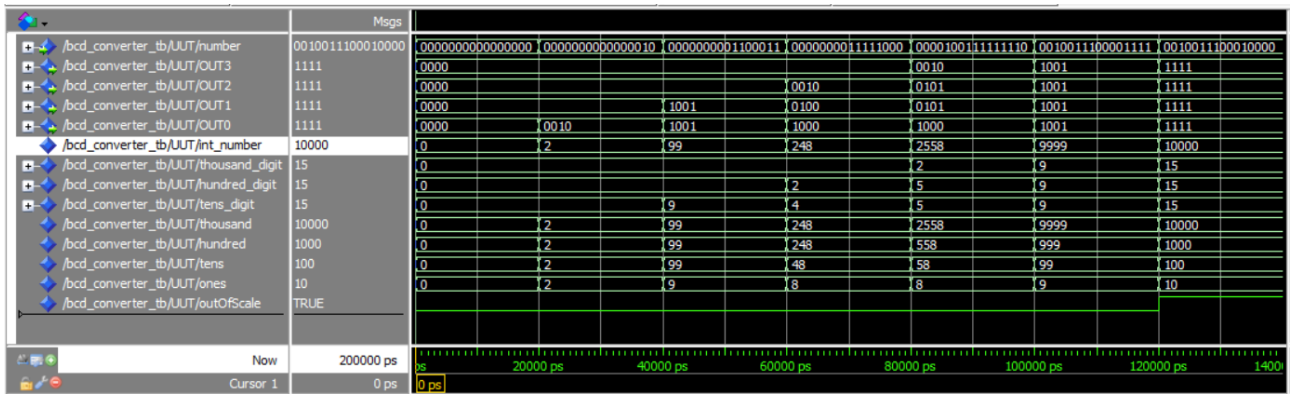

Figure 26 - Simulation result of the BIN-to-BCD converter

The waveform results for the binary-to-BCD converter are shown above. Notice that for numbers greater than 9999, every output is forced to '1111', which is decoded into a blank digit by the 7-segment decoder.

Synthesis

The *reactionTimer* and *binToBcd* entities were integrated into the *device_reactionTimer* (new main entity) to implement the circuit on the FPGA. The architecture of the main entity establishes the connections between the DE1-SoC on-board components, the internal 50 MHz clock, and the *hexDecoder* component. This integration ensures that the reaction time is displayed on four 7-segment displays.

The circuit operates as intended; however, certain operational clarifications are necessary. The circuit requires resetting at startup, achieved by pressing KEY0, to ensure proper functionality. Additionally, if the STOP pushbutton (KEY3) is activated before the LED lights up, the reaction time displayed is 0. This scenario is classified as a "false start," representing a condition equivalent to cheating. The last thing to notice is that after “9999”, no number is displayed because it is assumed that the user is away from the device.

To show the performance on the board we have recorded a short video that can be reached at this Dropbox folder.

Conclusions

No relevant difficulties have been encountered during the development of the VHDL code for the first four sections.

At the beginning of the VHDL code development for *Section 5*, some difficulties were encountered in the implementation of the circuit, mainly due to the choice of adopting a sequential approach and a relatively high level of abstraction. With this section, we have understood deeply about the importance of drawing the building block of a circuit before start writing the VHDL code.

This experience has proven to be both highly stimulating and highly important to learn how to design larger digital circuits and effectively describing them using VHDL.

Sources and notes

For all the VHDL files provided, an online tool has been used to format the code to ensure coherence between all the different files. The tool is free to use, and it is available at this link: VHDL Beautifier.

Part of the VHDL codes were implemented on a board purchased by a member of the group.

The VHDL files were developed using course materials produced by Prof. G. Masera (available on the course webpage) and the following eBooks recommended by the professor:

1. Mealy, Bryan, and Fabrizio Tappero. *Free Range VHDL*. 2016.
2. Ashenden, Peter J. *The VHDL Cookbook*. 1990.