Department of Electronics and Telecommunications
Bachelor degree in Electronic Engineering

Digital Systems Electronics

LAB 4: Flip-flops and counters.

Due date: April 22, 2024
- extended to: April 24, 2024
Delivery date: April 24, 2024

Group: 19

Contributions:

- Valtorta Alexander James
- Tedesco Angelo
- Urgu Sara

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 has been developed expressly for the assigned project.

Contents

0 Introduction **2**
0.1 Aim of this project . . . 2
0.2 Content and structure of the document . . . 2

1 Gated SR latch **2**
1.1 Design entry . . . 2
1.2 Functional simulation . . . 2
1.3 Synthesis . . . 2

2 16-bit synchronous counter **3**
2.1 Design entry . . . 3
2.2 Functional simulation . . . 4
2.3 Synthesis . . . 4

3 16-bit synchronous counter version 2 **5**
3.1 Design entry . . . 5
3.2 Synthesis . . . 5

4 Flashing digits from 0 to 9 **6**
4.1 Design entry . . . 6
4.2 Functional simulation . . . 7
4.3 Synthesis . . . 7

5 Reaction timer **8**
5.1 Design entry . . . 8
5.2 Functional simulation . . . 9
5.3 Synthesis . . . 9

0 Introduction

0.1 Aim of this project

The purpose of this laboratory is to investigate the operation of latches, flip-flops, registers and counters.

0.2 Content and structure of the document

The report is divided into four sections, each for every exercise of the laboratory assignment. Moreover, each section contains the key parts in order to design and implement each circuit:
- Design Entry, which describes the circuit to be implemented on the FPGA device, in particular it is indicated the overall architecture of the circuit, by reporting the name of the source file or files and the key role of each of them.
- Functional Simulation, which describes the testbench and the approach used to assess the completed design. (provided when requested)
- Synthesis, which shows the results of Modelsim functional simulation. (provided when requested)

1 Gated SR latch

1.1 Design entry

The design in the *gated_sr_latch.vhd* is provided by the assignment.

1.2 Functional simulation

The testebench (*tb_gated_sr.vhd*) allows to test the functionalities of the design by simulating the behavior of the latch.
In particular after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *gated_sr_latch.vhd*), and the internal signals for the simulation (*Clk_t, R_t, S_t, Q_t*), initialized to '0'.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by means of two processes: one for the simulation of the set reset, and the other for the simulation of the clock. The former is performed by assigning to the set internal signal (*S_t*) first the logic value '1' and then '0', waiting a short amount of time in between, and then again for the reset internal signal (*R_t*). The latter is performed by assigning to the internal signal (*CLK_t*) first the logic value '1' and then '0', waiting a short amount of time in between.

1.3 Synthesis

In order to examine the gate-level circuit generated from the code the Quartus Prime RTL Viewer tool is used, while the Technology Viewer tool is used to verify that the latch is implemented correctly.

(a) Gate-level circuit from RTL Viewer tool

(b) Implementation verified from the Technology Viewer tool

Figure 1: Verification the Gated SR latch

Once the circuit is successfully synthesized by Quartus it is possible to run the Modelsim functional simulation.

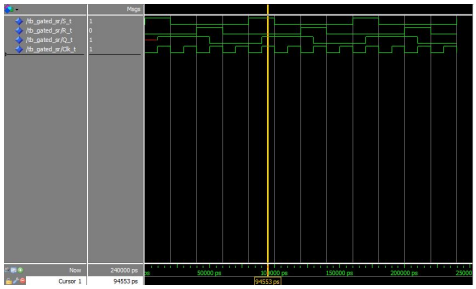


Figure 2: Modelsim functional simulation of the Gated SR latch

2 16-bit synchronous counter

2.1 Design entry

In order to defines the 16-bit counter by using the structure depicted some elements need to be specified:

-The Toggle flip-flop (*T_flipflop.vhd*) is defied by means of a behavioural architecture where once the internal signal *Q1* is declared and initialized to '0', it is assigned to the output port *Q* of the t-type FF, thus a process is set up, whose sensitivity list contains the clock *Clock* and reset *Resetn* inputs, thus the process defines a sequential process sensitive to changes in the Clock and Resetn signals and describes how the flip-flop behaves based on these signals. First it is tested the *Resetn*, if it is asserted, so it is equal to '0', *Q1* is asynchronously cleared by setting it to '0', otherwise it is tested the presence of the rising edge of the clock in order to evaluate *Q1* based on the input *T*, in particular if *T* is '1' then *Q1* toggles its state, otherwise it remains unchanged.

- For the 4-bit hexadecimal decoder with seven segments (*hex_decoder.vhd*) it is used the same design previously described for Lab3.

- The 4-bit synchronous counter is implemented in (*n_bit_counter.vhd*) by setting in the entity the generic parameter *N* to 4 and using a structural architecture where the T-flipflop (specified in *T_flipflop.vhd*) is defined as a component and the internal signals *C, r* are declared as vectors (they represent the counter output and the intermediate signals used for generating the counter, respectively). Once the *C* internal signal and the *Enable* input are assigned to the *Cout* output and the first bit of the *r* internal signal, the first flipflop is instantiated by mapping the inputs and output of the flipflop to those of the counter, the same is done for the remaining flipflop, using a for-generate

statement, in particular the each bit i of the r internal signal is now computed based on the previous bit *r(i-1)* and the corresponding counter bit *C(i-1)*.

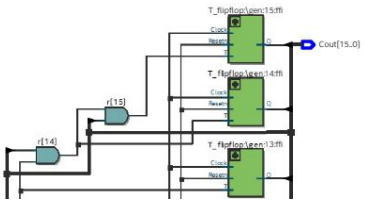
- Lastly in the *counter_map.vhd* it takes place the mapping of the inputs and outputs of the counter (defined in *n_bit_counter.vhd*) and that of the decoder (defined in *hex_decoder.vhd*) to those assigned to the DE1 board, in order to visualize the hexadecimal numbers in the displays. In particular are assigned *KEY1* as a manual clock input, switches *SW0, SW1* as Reset and Enable respectively and, finally, the seven segment displays *HEX0, HEX1, HEX2, HEX3* to display the hexadecimal count as the circuit operates. Noticing that the generic parameter N is set to 16 since the counter is a 16-bit one, and the internal signal *out_tmp* is used to temporarily store the output of the counter.

2.2 Functional simulation

With the testbench *tb_n_bit_counter.vhd* it is possible to check the functionalities of the completed design, in particular, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *n_bit_counter.vhd*), and the internal signals for the simulation (*En_t*, *Clk_t*, *CLR_t*, all initialized to '0' and *out_t*).
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a two processes: one for the simulation of the enable and clear, and the other for the simulation of the clock. The former is performed by assigning to the clear internal signal (*CLR_t*) and to the enable signal *En_t* first the logic value '1', then toggling *En_t* from '1' to '0' and again to '1', and finally assigning '0' to both *CLR_t* and *En_t*, all done by waiting a short amount of time in between each commutation. The latter is performed by assigning to the internal signal (*CLK_t*) first the logic value '1' and then '0', waiting a short amount of time in between.

2.3 Synthesis

It is possible to observe how many logic elements (LEs) are used to implement the circuit and what is the maximum frequency at which it can be operated.



(a) 15 logic elements used

Slow 1100mV 85C Model Fmax Summary

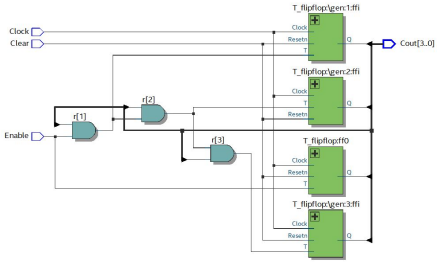
<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	361.01 MHz	361.01 MHz	Clock	

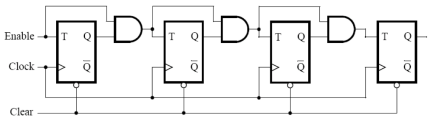
(b) Maximum frequency

Figure 3: LEs and Fmax

Finally using the Quartus Prime RTL Viewer to see how the tool synthesizes the circuit it is possible to compare it with the circuit provided in the assignment:



(a) 4-bit counter synthesized by Quartus Prime RTL Viewer



(b) Provided circuit

Figure 4: Resulting and provided circuit

3 16-bit synchronous counter version 2

3.1 Design entry

In order to simplify the previous design it is only a matter of adding the *numeric_std package* of the IEEE library to the *hex_decoder.vhd* and to the *counter_map.vhd* file to allow addition of unsigned number, and modify the design of the counter.
In particular, aside from the addition of the library, the change consists in the implementation of a behavioural architecture whose process is sensitive to changes in the *Enable, Clock, Clear* input signals. Moreover, the process checks if *Clear* is '0' thus resets the counter output value *Cout* to '0', otherwise, if *Enable* is asserted, it checks for a rising edge of the clock, thus increments *Cout* by one.

3.2 Synthesis

Once the compilation is successful it is possible to observe what is the maximum frequency at which it can be operated.

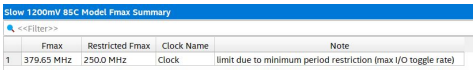
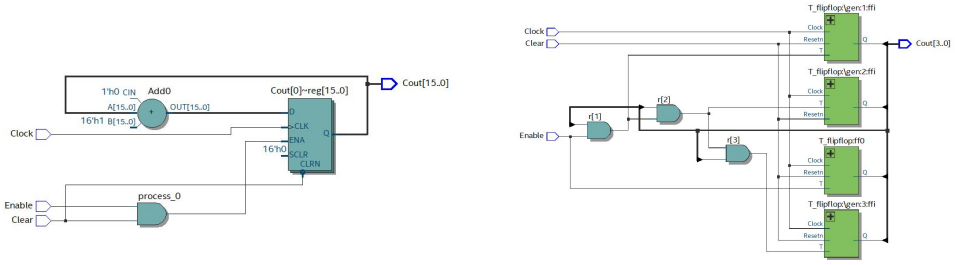


Figure 5: Maximum frequency

Finally using the Quartus Prime RTL Viewer to see how the tool synthesizes the circuit it is possible to observe how much the complexity is improved with respect to the previously designed circuit:



(a) current 4-bit counter synthesized by Quartus Prime RTL Viewer

(b) Previous circuit

Figure 6: Differences with the circuits

4 Flashing digits from 0 to 9

4.1 Design entry

In order to implement a circuit that successively flashes digits from 0 to 9 on the 7-segment display HEX0, where each digit is displayed for about one second, the idea is to have two counters, both running at 50 MHz. One counter increments on every clock cycle up to 50 MHz and then restarts. Additionally, it sends a signal to enable the second counter. The enable signal for the second counter activates only for one cycle and then remains inactive until it receives another enable signal. This way, the second counter increments by 1 only when it receives 50 million clock pulses, which happens once per second.

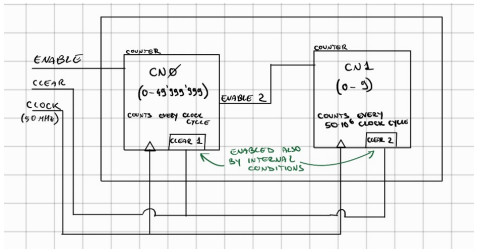


Figure 7: Design of the circuit

In particular it is composed by:

- The 7-segment decoder is defined in *decoder.vhd* with the dataflow structure already mentioned for previous labs.

- The T-type flipflop is defined in *Tflipflop.vhd* with a behavioural architecture that also makes use of a flag.

- The first counter (defined in *digit_counter*) generates a 4-bit count sequence representing individual digits (0 to 9). In order to do so the FF (defined in *Tflipflop.vhd*) is defined as a component in the declarative part of the structural architecture where *Tv,Qv* are defined as internal signals so that, in the descriptive part, the inputs and outputs of four FFs can be properly mapped to those of the counter (defined in the entity) and the internal signals, making use of a for-generate statement for the

intermediate FFs.

- The same approach is used to design the second counter, which is defined in *counter26bit.vhd* and generates a 26-bit binary count sequence, thus the only difference is in the declaration of the output Q (here is a 26-bit vector, while for the *digit_counter* if was a 4-bit vector) and, as a consequence also in the number of iteration for the for-generate statement.

- Finally to control the two counters and regulate the display of the seven-segment LED digit the *digit_flasher.vhd* is defined, where the two counters (*digit_counter, counter26bit.vhd*) are defined as component along with the initialized internal signals *Enable2,Clear1,Clear2,Qv1,Qv2,U1,U2* in the declarative part of the architecture. Thus, once the components are properly mapped to the ports of the *digit_flasher* and the internal signals, and the *Qv1, Qv2* are converted to integers the behaviour of the counters and the display can be controlled by means of a process. In particular when *Clear* is asserted, both counters are cleared (*Clear1, Clear2* set to '1') and the enable signal for the second counter (*Enable2*) is set to '0'; on the other hand (if *Clear* is equal to '0') first it checks for the right conditions in order to enable the second counter and reset the first one (*Enable2, Clear1* set to '1'), then it checks for the right condition needed reset the second counter (*Clear2* set to '1').

4.2 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.

- In the *counter26bit_tb.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *counter26bit.vhd*), and the initialized internal signals for the simulation (*Enable_tb, Clock_tb, Clear_tb, Q_tb.*
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a two processes: one for the simulation of the enable and clear, and the other for the simulation of the clock. The former is performed by assigning to the enable signal *Enable_tb* the logic value '1', then toggling the clear signal *Clear_tb* from '1' to '0', waiting in between each commutation. The latter is performed by assigning to the internal signal (*Clock_tb*) first the logic value '0' and then '1', waiting a short amount of time in between.

- In the *flasher_tb.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *digit_flasher.vhd*), and the initialized internal signals for the simulation (*Enabletb, Clocktb, Cleartb, Htb.*
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a two processes: one for the simulation of the enable and the other for the simulation of the clock. The former is performed by assigning to the enable signal *Enabletb* the logic value '1' then wait some time. The latter is performed by assigning to the internal signal (*Clock_tb*) first the logic value '0' and then '1', waiting a short amount of time in between.

4.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run the Modelsim functional simulation.
For the simulation it was used a counter up to 1000 instead of 50M, otherwise Modelsim would reach the maximum number of possible iterations

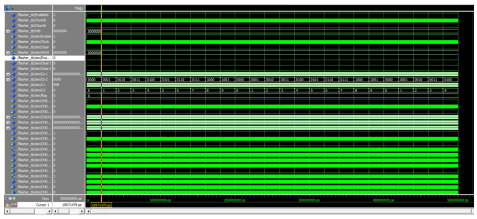
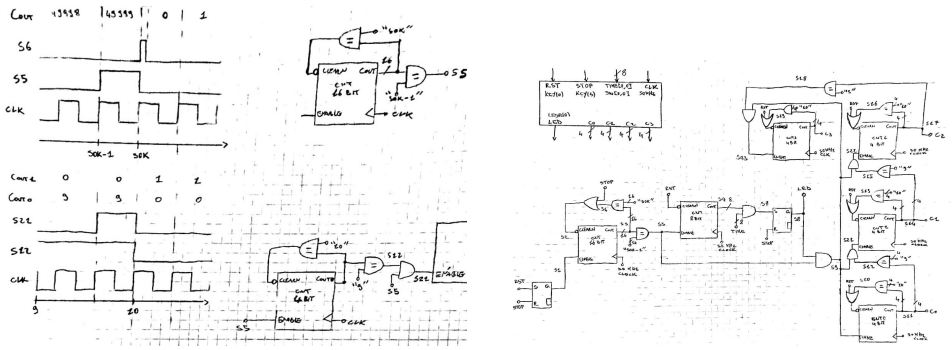


Figure 8: Modelsim functional simulation

5 Reaction timer

5.1 Design entry

The reaction-timer is implemented through the following design:



(a) (b)

Figure 9: Design of the circuit

In particular it is possible to define:

- The T-type flipflop is defined in *T_flipflop.vhd* with the behavioural architecture previously described.

- The 7-segment decoder is defined in *hex_decider.vhd* with the behavioural architecture previously used and described in other labs' assignments.

- The counter defined in *n_but_counter.vhd* is the same used for the 4-bit counter of the second assignment.

- The gated SR latch is defined in *gated_sr_latch.vhd* with a structural architecture that makes use of the attribute statement in order to prevent the internal signals *R_g, S_g, Qa, Qb* to be optimized by the synthesis tool.

- The n-bit AND-gate is defined in *and_nbit.vhd* with a behavioural architecture that though means of a process it compares the input vectors A,B: if they are equal, the output Y is set to '1', otherwise, to '0'.

- The completed circuit is defined in *reaction_timer.vhd*, where in the declarative part of the architecture the *n_but_counter.vhd*, the *gated_sr_latch.vhd* and the *and_nbit.vhd* are defined as components

and *s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16, s17, s18, s19, s20, s10n, s13n, s16n, s19n, s21, s22, s23* as internal signals. In the descriptive part all the components can be connected following the design by mapping their ports with the appropriate internal signal and the inputs and outputs declared in the entity (*RST, STOP, CLK, TIME, C0, C1, C2, C3, LED*), making use of the generic statement to define the correct number of bits.

- Lastly in the *timer_map.vhd* it takes place the mapping of the inputs and outputs of the whole circuit (defined in *reaction_timer.vhd*) to those assigned to the DE1 board, in order to visualize the hexadecimal numbers in the displays and press the push buttons. In particular *KEY0* to reset the circuit; the red lights *LEDR7-0* that turns on after the elapsed time; *HEX3-0* to show the reaction time; switches *SW7-0* to set the elapsed time; *KEY1* to turn the LED off and freeze the counter in its present state; *CLOCK_50* to manage the clock from the external.

5.2 Functional simulation

With the testbench *tb_reaction_timer.vhd* it is possible to check the functionalities of the completed design, in particular, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *reaction_timer.vhd*), and the internal signals for the simulation (those initialized to '0': *STOP_t, CLK_t, LED_t, RST_t, TIME_t*; and *c0_t, c1_t, c2_t, c3_t*).
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones and *TIME_t* is set to the equivalent of 1ms ('00000001') the simulation is performed through means of a two processes: one for the simulation of the reset and stop, and the other for the simulation of the clock. In particular, the first one is performed by changing the value of *RST_t* from '1' to '0', do the same for *STOP_t* and repeating again for both the signals, waiting a short amount of time in between each commutation.

5.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run the Modelsim functional simulation.
For the simulation it was used a counter up to 1000 instead of 50000, otherwise Modelsim would reach the maximum number of possible iterations.

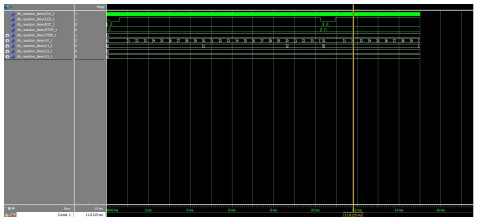


Figure 10: Modelsim functional simulation