"FPGA-Programming" Exercise Sheet III:

Latches, FlipFlops und Register

The integrated logic block of FPGA devices include flip-flops, which can be utilized for the description of sequential logic and storage circuits. Goal of this exercise is to examine latches, flip-flops and registers.

As target hardware always choose FPGA chip Cyclone V SoC 5CSEMA5F31C6

1) *Gated RS-latch*

Within this part, it should be demonstrated how storage elements can be described in FPGA devices, without using dedicated flip-flops of the chip. Following figure illustrates a *gated RS-latch*.

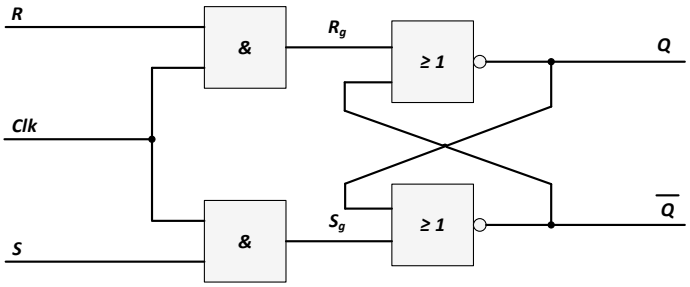

Figure 1.1: Circuit of a gated RS-latch

i) From the given circuit, derive the complete truth table and describe the logic functions for outputs Q and $\bar{Q}$.

If this gated RS-latch should be implemented inside of a FPGA chip, which contains Lookup Tables (LUTs) with 4 inputs, then only one LUT block will be needed in order to realize the functionality as following figure shows.

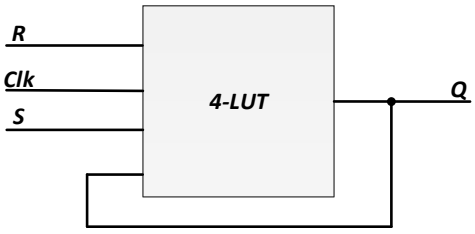

Figure 1.2: Implementation of a gated RS-latch in a single LUT

Although a gated RS-latch can be implemented in a single LUT, this implementation does not allow to observe the internal signals R_g and S_g. The internal signals will be cleaned out by the compiler for optimization. It the signals must be observed, for instance during simulation, a so called *compiler assignment* must be used, which instructs Quartus to use discrete LUTs for the implementation of the signals R_g, S_g, Q and $\bar{Q}$. Following figure illustrates the LUT implementation when using the compiler assignment **keep** by a VHDL **attribute** statement.

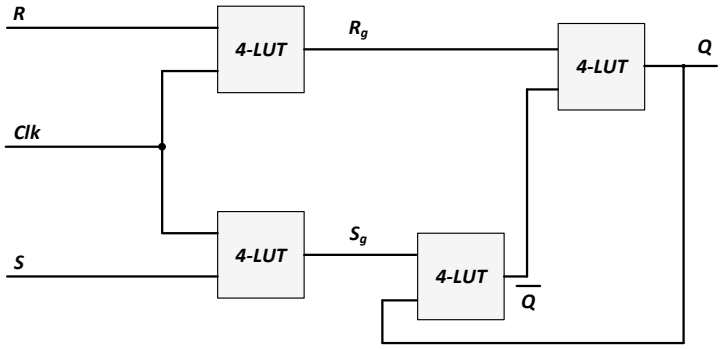

Figure 1.3: Implementation of a gated RS-latch in four discrete LUTs

Implement the circuit of the gated RS-latch. Perform following steps:

ii) Create a new Quartus project for the circuit, named **e_my_gatedRSlatch**.

iii) Describe the architecture (**a_my_gatedRSlatch_1**) for the circuit of the gated RS-latch using boolean expressions **without** using the **keep** attribute. Compile the circuit and analyse it with Quartus **RTL Viewer** and **Technology Map Viewer**.

iv) Add an additional architecture description (**a_my_gatedRSlatch_2**) in which the **keep** attribute for the internal signals R_g, S_g, Q and $\bar{Q}$ is used. Describe a configuration block named **c_my_gatedRSlatch** and configure it is use the second architecture description. Compile the circuit and analyse it using Quartus **RTL Viewer** and **Technology Map Viewer**.

2) *Gated D-latch*

Following figure shows the circuit of a gated D-latch which should be examined similar to the one of part 1).

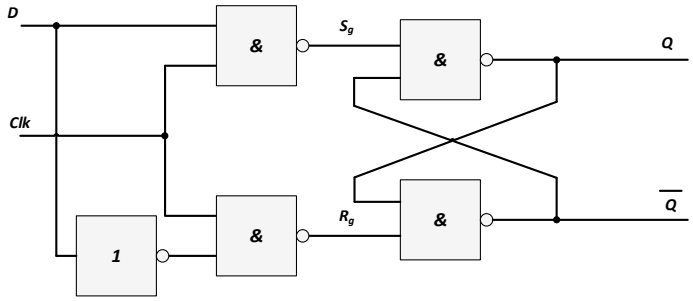

Figure 2.1: Circuit of a gated D-latch

Perform following steps:

i) From the given circuit, derive the complete truth table and describe the logic function for the outputs Q and $\bar{Q}$.

ii) Create a new Quartus project for the circuit, named **e_my_gatedDlatch**.

iii) Create a VHDL sub-level entity named **e_gatedDlatch**. Describe the architecture of the gated D-latch using boolean expressions both **without** using the **keep** attribute (**a_gatedDlatch_1**) and **with** using the **keep** attribute (**a_gatedDlatch_2**).

iv) Create the VHDL top-level entity named **e_my_gatedDlatch** in which the component **e_gatedDlatch** will be instantiated. Use the **configuration specification** syntax to first include the instance named **I_DLatch** using the architecture description without the keep attribute set (**a_gatedDlatch_1**). Compile the circuit and analyse it with Quartus **RTL Viewer** and **Technology Map Viewer**.

v) Configure the circuit to use architecture description **a_gatedDlatch_2** for the instance **I_DLatch**. Compile the circuit and analyse it with Quartus **RTL Viewer** and **Technology Map Viewer**.

vi) For input D use switch $SW[0]$, for input Clk use switch $SW[1]$. Output Q should be assigned to $LEDR[0]$. Import the necessary pin assignments, compile the circuit and download the compiled circuit into the FPGA.

vi) Test the functionality of the circuit in hardware.

3) *Gated Master-Slave D-latch*

Following figure shows the circuit of a *gated MS-D-latch*, which should be analysed in this part.

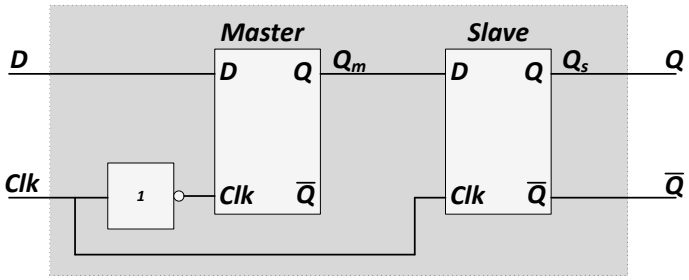

Figure 3.1: Circuit of a gated Master-Slave D-latch

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_gatedMSDlatch**. Create a VHDL top-level entity which includes two instances (**I_MDLatch** and **I_SDLatch**) of the gated D-latch (**e_gatedDlatch**) from part 2). The two instances should be configured to use the architecture description which does not use the **keep** attribute (**a_gatedDlatch_1**).

ii) For input D use switch $SW[0]$, for the input Clk use switch $SW[1]$. Output Q should be assigned to $LEDR[0]$. Furthermore the Output of the Master Q_m should be assigned to $LEDR[1]$. Import the necessary pin assignments and compile the circuit.

iii) Analyse the circuit with Quartus **Technology Map Viewer**.

iv) Download the circuit and test the functionality in hardware.

4) *Comparison of different D-type flip-flops*

Following figure shows a circuit, consisting of three different storage elements, one gated D-latch, as well as a positive-edge triggered D-flip-flop and a negative-edge triggered D-flip-flop.

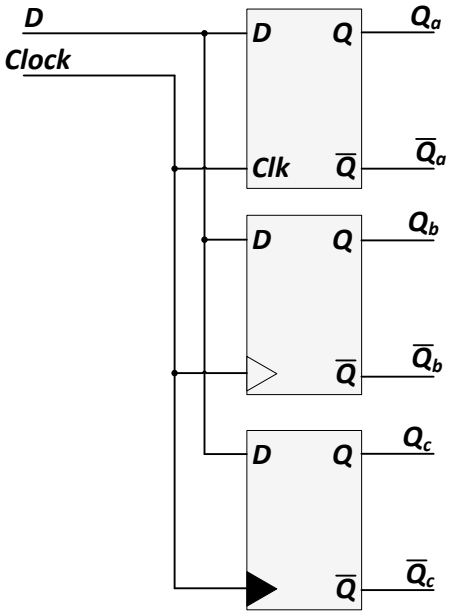

Figure 4.1: Three different D-flip-flop types

Following timing diagram clarifies the behaviour of the individual flip-flops.

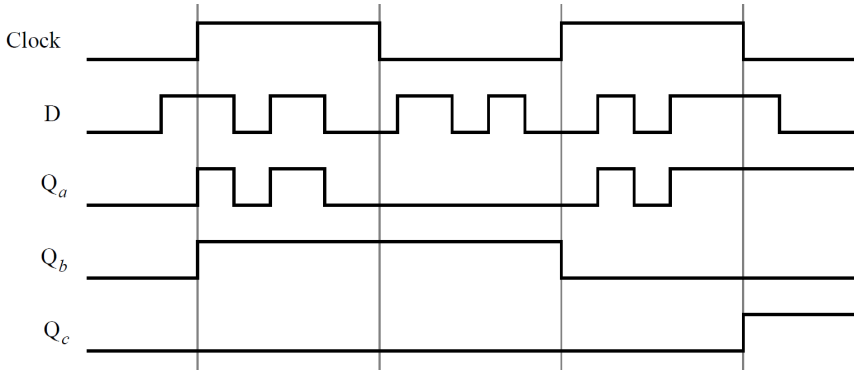

Figure 4.2: Timing diagram for the three types of D-flip-flop

Implement and simulate this circuit. Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_DtypeFFs**.

ii) Write a VHDL sub-level entity (**e_gatedDlatch**) for the description of a gated D-latch **without** usage of the **keep** attribute.
Describe the behaviour by using a *combinatorial process*.

iii) Write a VHDL sub-level entity (**e_posedgeDFF**) for the description of a positive-edge triggered D-flip-flop. Describe the behaviour by using a *sequential process*.

iv) Write a VHDL sub-level entity (**e_negedgeDFF**) for the description of a negative-edge triggered D-flip-flop. Describe the behaviour by using a *sequential process*.

v) Create a top-level entity (**e_my_DtypeFFs**) which includes all three components. Compile the circuit.

vi) By using the Quartus **Technology Map Viewer** verify that the gated D-latch will be implemented in one single LUT, whereas the positive-edge triggered D-flip-flop and negative-edge triggered D-flip-flop will be implemented in discrete flip-flop blocks of the target FPGA.

vii) Perform a functional simulation with ModelSim and examine the behaviour of the three D-type flip-flops.

5) *Simple example to storage elements*

Within this part the hexadecimal value of a 8-bit number A should be displayed on the 7-segment displays $HEX3$ and $HEX2$. Furthermore the hexadecimal value of a 8-bit number B should be displayed on the 7-segment display $HEX1$ and $HEX0$. The values of A and B are the inputs of the circuit which will both be provided by the switches $SW[7-0]$.

To input the values of A and B, first the value for A should be set by the switches and stored in a register, before the value of B will be set by the very same switches.

Furthermore an adder circuit should be described, which calculates the arithmetic sum $S = A + B$, and displays this sum on the 7-segment display $HEX5$ and $HEX4$. The carry output produced by the adder should be shown on $LEDR[0]$.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_reg**.

ii) Create a VHDL sub-level entity named **e_hex7seg**. Describe a 7-segment decoder which interprets a 4-bit input vector as hexadecimal number and decodes it for displaying it on a 7-segment display. Use a *combinatorial process* named **p_Decode** together with a **case** statement.

iii) Create a VHDL sub-level entity named **e_reg_arn**. Describe a 8-bit wide register with *low-active asynchronous* reset. Use a *sequential process*.

iv) Write the VHDL top-level entity (**e_my_reg**) which implements the requested functional range. Use $KEY[0]$ as *low-active* reset and $KEY[1]$ as manual clock input.

v) Import the necessary pin assignments and compile the circuit.

vi) Perform a functional simulation with ModelSim and examine the behaviour of the circuit.

vii) Download the compiled circuit and test the functionality in hardware.