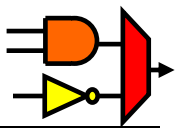


Elettronica dei Sistemi Digitali
Digital Systems Electronics

Lab#4

Flip-flops and counters

This lab aims at investigating the operation of latches, flip-flops, registers and counters. For example, these components are normally used in the Arithmetic Logic Units of the DSPs and, generally, in many different sub-units of every microprocessor. **Please note that for exercise 4 and 5 you have first to design your circuit on a paper and then to write the VHDL.**

Contents:

Abbreviations and acronyms:

IC – Integrated Circuit
SR – Set Reset
LE – Logic Element
LED – Light Emitting Diode
LUT – Look Up Table
MUX – Multiplexer
RCA – Ripple Carry Adder
RTL – Register Transfer Level
VHDL – Very high speed integrated circuits Hardware Description Language
[VHDL cookbook: http://www.onlinefreeebooks.net/engineering-ebooks/electrical-engineering/the-vhdl-cookbook-pdf.html]

1 – Gated SR latch

The Altera FPGAs already include internally some flip-flops that are available for implementing a sequential circuit if required by the user. Here we will show how to use these flip-flops in sections from 4 to 6 of this laboratory

manual. Anyway, at first we will show how storage elements can be created in an FPGA without using its dedicated flip-flops.

Figure 1 depicts a gated SR latch. A style of VHDL code that uses logic expressions to describe this circuit is given in Figure 2. If this latch is implemented in an FPGA that has 4-input Look-Up Tables (LUTs), then only one LUT is needed as shown in Figure 3a.

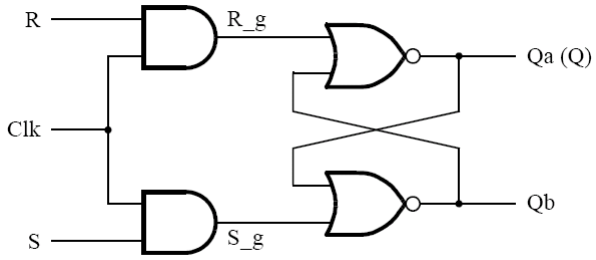


Figure 1. A gated RS latch circuit.

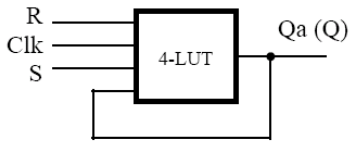
```
- - A gated RS latch described the hard way
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part1 IS
   PORT ( Clk, R, S : IN STD_LOGIC;
   Q : OUT STD_LOGIC);
END part1;

ARCHITECTURE Structural OF part1 IS
   SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC ;
   ATTRIBUTE keep : boolean;
   ATTRIBUTE keep of R_g, S_g, Qa, Qb : SIGNAL IS true;
BEGIN
   R_g <= R AND Clk;
   S_g <= S AND Clk;
   Qa <= NOT (R_g OR Qb);
   Qb <= NOT (S_g OR Qa);
   Q <= Qa;
END Structural;
```

Figure 2 - Specifying the RS latch by using logic expressions.

Although the latch can be correctly synthesized in one single 4-input LUT, this implementation does not allow its internal signals, such as *R_g* and *S_g*, to be observed, because they are not provided as outputs from the LUT. To preserve these internal signals it is necessary to include a *compiler directive* in the code. In Figure 2 the directive *keep* is included by using a `VHDL ATTRIBUTE` statement; it instructs the Quartus Prime compiler to use separate logic elements for each of the signals *R_g*, *S_g*, Q*a*, and Q*b*. After the code compilation, the tool generates the circuit with four 4-LUTs depicted in Figure 3b.



(a) Using one 4-input lookup table for the RS latch.

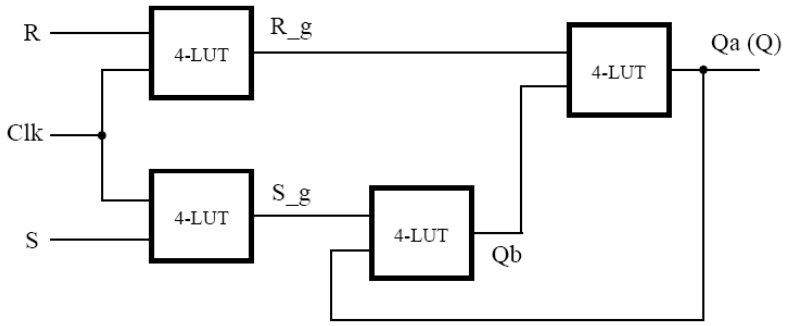


Figure 3 - Implementation of the RS latch from Figure 1.

Implement the SR latch circuit as follows:

1. **Create a new project for the SR latch**.
2. **Generate a VHDL file with the code** in Figure 2 and include it in the project.
3. **Compile the code**. Use the Quartus Prime RTL Viewer tool to examine the gate-level circuit generated from the code (**Tools -> Netlist Viewers -> RTL Viewer**) and use the Technology Viewer tool (**Tools -> Netlist Viewers -> Technology Map Viewer**) to verify that the latch is implemented in the way shown in Figure 3b.
4. ***Create an ad-hoc testbench to test the correct operation of the circuit.*** Simulate the behavior of the circuit with Modelsim.

2 – 16-bit synchronous counter

Consider the circuit in Figure 4. It is a 4-bit synchronous counter, which uses four T-type flip-flops. The counter increments the count signal on each positive edge of the clock if the Enable signal is asserted. The counter is reset to 0 by using the Reset signal. You need to implement a 16-bit synchronous counter.

1. **Write a VHDL file** that defines the 16-bit counter by using the structure depicted in Figure 4 and compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, f_{max}, at which your circuit can be operated?
2. **Simulate your circuit** to verify its correctness.
3. **Augment your VHDL** file to use the pushbutton KEY0 as the Clock input, switches SW1 and SW0 as Enable and Reset inputs, and 7-segment displays *HEX3-0* to display the hexadecimal count as your circuit operates. Make the necessary pin assignments and compile the circuit.
4. **Implement your circuit** on the DE1 board and test its functionality by operating the implemented switches.
5. **Implement a 4-bit version of your circuit** and use the Quartus Prime RTL Viewer to see how the tool synthesizes your circuit. What are the differences with respect to Figure 4?

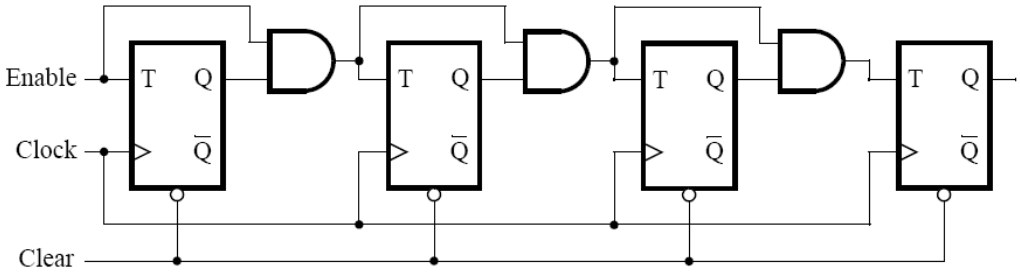


Figure 4 - A 4-bit counter.

3 – 16-bit synchronous counter version 2

Simplify your VHDL code so that the counter specification is based on the VHDL statement

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

In order to allow the addition of `unsigned` numbers, you need to add the numeric_std package of the IEEE library as to the standard 1164 package:

```
USE ieee.numeric_std.all;
```

Compile a 16-bit version of this counter and compare the number of LEs needed and the f_{max} required. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part II.

4 – Flashing digits from 0 to 9

Design and implement a circuit that successively flashes digits from 0 to 9 on the 7-segment display HEX0. Each digit should be displayed for about one second. Use a counter to determine the one-second interval. The counter should be incremented by the 50 MHz clock signal provided on the DE1 board (`CLOCK_50` input signal, to be managed as a clock input coming from the external world). Do not derive any other clock signals in your design; make sure that all the flip-flops in your circuit are clocked directly by the 50 MHz clock signal. **Note: design your circuit on a paper before starting the VHDL implementation.**

5 – Reaction timer

Design on a paper and then implement on the DE1 board a reaction-timer circuit. The circuit should operate as follows:

1. **The circuit is reset by pressing the push button switch *KEY*$_0$.**
2. After an elapsed time, the red light labeled ***LEDR*$_0$ turns on and a four-digit *HEX3-0* counters starts counting in intervals of 1 millisecond**. The amount of time in milliseconds from when the circuit is reset until *LEDR* $_0$ is turned on is set by switches *SW*$_{7-0}$.
3. A person whose speed is being tested must **press the pushbutton *KEY*$_3$ as quickly as possible** to turn the LED off and freeze the counter in its present state. The counter stays frozen until the ***KEY*$_0$** is pressed. **The count which shows the reaction time will be displayed on the 7-segment displays *HEX3-0*.**
4. ***Create an ad-hoc testbench to test the correct operation of the circuit.*** The timing and the values of the input signals are up to you.