Assignment Homework 1

Due date: Check Canvas

STUDENT VERSION

Please team up with a fellow student and start working on the following tasks (100 pts total). In addition, you *must* consider the following:

- Vivado creates a default template including a comment block with:
  - "Company" – insert team name/number
  - "Engineer" – put team member names here
  - "Design Name" – based on task description (*do not leave empty*)
  - "Project Name" – Assignment 1
  - "Description" – based on task description (*do not leave empty*)
- To prepare your assignment for submission through Canvas, you must use: `File > Project > Archive`
- You work as a team. Please alternate between one student creating the testbench, the other student creating the implementation.

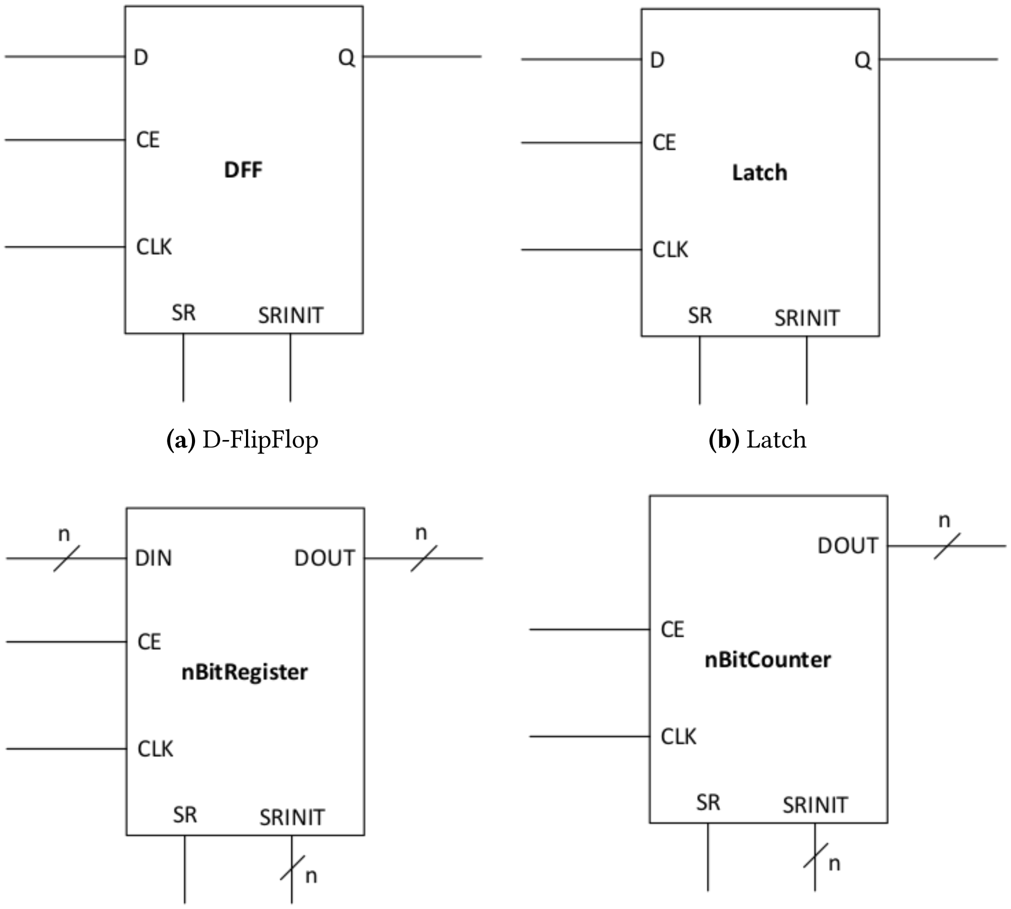
We will start this homework assignment together through Zoom as a lab. Please try to follow along and ask questions if you get stuck somewhere. Your homework assignment includes all tasks mentioned here (even those, that you manage to complete during our joint lab time).

1 Task

As hardware engineers, you enjoy starting your projects using a bottom-up approach. Therefore, you feel like specifying and implementing a couple of basic modules you may need for your other designs. After some thoughts, you are positive that you need the following:

1. A basic D-FlipFlop,
2. A basic Latch,
3. An n-Bit register,
4. An n-Bit counter

After a moment of thought, you realize that while your first impulse was to do this all by yourself, it is better to split the workload with your colleague. Greater things can only be achieved in a team (despite all disagreements that occur as part of team work)! The content of this assignment should be mostly known due to taking ECE 272.



(a) D-FlipFlop **(b)** Latch **(c)** n-Bit register **(d)** n-Bit counter

Figure 1: Overview of the interface (VHDL: port) definitions.

2 General Design

For now, we mostly ignore the recommended Verilog coding style to keep things as simple as possible. In order to keep the modules consistent, your clock signal is always named **CLK**, the data input is called **D** in case of a single bit and **DIN** in case of a bus, while the output values have the name **Q** and **DOUT**, respectively. You always use a set/reset signal **SR**, which initializes the storage element(s) with the value of the input signal/bus **SRINIT**. In addition, you use an enable signal **CE** to enable or disable the module. Thinking ahead, you define that **SR** is always more important than **CE**. As you recall two ways to reset a module (clock synchronously and asynchronously), you will add the suffix **SSR** and **ASR** to the entity names.

Please ensure that you match the signal name correctly. **Failure to do so will cause your code to not run on the TA side, resulting in 0 points.**

3 Output

3.1 Basic modules

Create the Verilog files for the following eight entities:

- DFF_ASR, nBitRegister_ASR, DLatch_ASR, nBitCounter_ASR (all asynchronous)
- DFF_SSR, nBitRegister_SSR, DLatch_SSR, nBitCounter_SSR (all synchronous)

In your group, one of you is responsible for, e.g., creating the first four files. The other will handle the implementation of the other modules (or any other combination to share the workload). Toss a coin to determine your roles.

Utilize the “File-Add Sources..-Add or create design sources” Navigator to generate the Verilog files (Verilog Module). While using the wizard is an option for entering input/output signals, you are encouraged to write the entity from scratch if possible.

3.2 Test benches

Once you have completed the basic modules, proceed to develop test benches for each. To streamline this process, formulate a master plan outlining the essential steps required:

1. You use your favorite drawing tool (if you use MS Paint, the angry testing gods will never forgive you ~~and you will suffer a very slow and horrible death by paintbrushes~~[*]) and create a control flow graph for each module.
2. You want to do a C_2 whitebox test [†] for the register and the counter and document your test cases (including the test input and correct output) together with the control flow graph.
3. You want to do a full blackbox test for the Latch and FlipFlop.
4. To save some copy & paste work, you always test a **uut_ssr** and **uut_asr** module together in one test bench (UUT is for “unit under test”).
   As you simulate the modules, you want to color the internal state and output value in different colors and group the signals belonging to each uut. As the counter contains a hex value, you change the radix of the output signal to hex. You decide to save the resulting wave form configuration and name it after the test bench.

Now, create four Verilog test benches: DFF_tb, DLatch_tb, nBitCounter_tb, and nBitRegister_tb. Use the "File-Add Sources..-Add or create simulation sources" Navigator to generate Verilog Testbench files. If you identify any errors in your previous modules, ensure they are corrected to fulfill their intended usage.

In your group, the individual who implemented DFF_ASR, DFF_SSR, DLatch_ASR, and DLatch_SSR files is responsible for creating nBitCounter_tb and nBitRegister_tb. The other individual will handle DFF_tb and DLatch_tb. For example:

[*] It is safe to assume that OSU’s legal department would not approve which is why we had to modify this German-style task description. Please use open-source tools such as: Inkscape, Ipe drawing editor, LibreOffice Draw, . . .

[†] `https://en.wikipedia.org/wiki/Code_coverage`

Person A: Create the basic modules (DFF_ASR, DFF_SSR, DLatch_ASR, and DLatch_SSR files), and develop the test benches (nBitCounter_tb and nBitRegister_tb). Person B: Create the basic modules (nBitRegister_ASR, nBitRegister_SSR, nBitCounter_ASR, and nBitCounter_SSR files), and develop the test benches (DFF_tb and DLatch_tb).

Please ensure that you match the module/file name correctly. **Failure to do so will cause your code to not run on the TA side, resulting in 0 points.**

3.3 Submission

Submit your report/program code on Canvas (all files compressed in one `.zip` in addition to the archived project). The archive you submit must contain all .v files. Please make sure that your code passes the syntax check and comment anything which is not obvious (use the Verilog comments // comment).

Please also include the control flow graphs, as well as the test case documentation for the two whitebox tests and the wave form configuration files.

4 Hints and Tips

The following is an (incomplete!) example of the module DFF_SSR.

Listing 1: Example of the module DFF_SSR

```verilog
module DFF_SSR(
   input CLK,   // CLOCK
   input SR,    // RESET
   input CE,    // ENABLE
   input SRINIT, // INITIAL VALUE
   input D,     // D
   output reg Q // Q
);

always @(posedge CLK)
begin
   if (SR == 1'b1) // synchronous reset
      Q <= SRINIT;
   else
      Q <= D;
end

endmodule
```

In order to create n-Bit modules, use the *parameter* to define the width n, then use it in the port definition. Please do not use inout signals, as these basic modules do not need them.

If you test a sub-module, always make sure that you use registers to make the input signals clock synchronous. If you have a variable bit width, test your design with a 4 bit bus. If you want to test a synchronous reset together with an asynchronous reset in one test bench, you will probably need different reset test cases (and thus signals).