FPGA tutorial

Lecture 12

27.01.2021

Jochen Steinmann

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

Organisatorical

Discussion of Evaluation

How does the exam look like?

I uploaded you a previous exam.

Note:

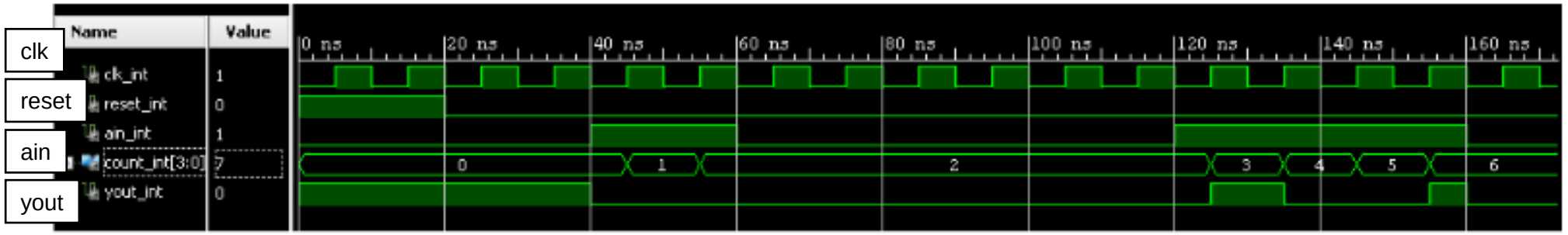
1. I changed the language from VERILOG to VHDL of the course.
   There are language specific differences.
2. Previous courses used real hardware, this time „just“ simulation.
   Topics might be different.


Solution

ToDo Lecture 11

A. Implement the three process **Marley** FSM presented in these slides

B. Design a sequence detector implementing a Mealy state machine using three blocks.
The Mealy state machine has one input (ain) and one output (yout).
The output yout is 1 if and only if the total number of 1s received is divisible by 3 (hint: 0 is inclusive, however, reset cycle(s) do not count as 0- see in simulation waveform time=200).
Develop a testbench and verify the model through a behavioral simulation.



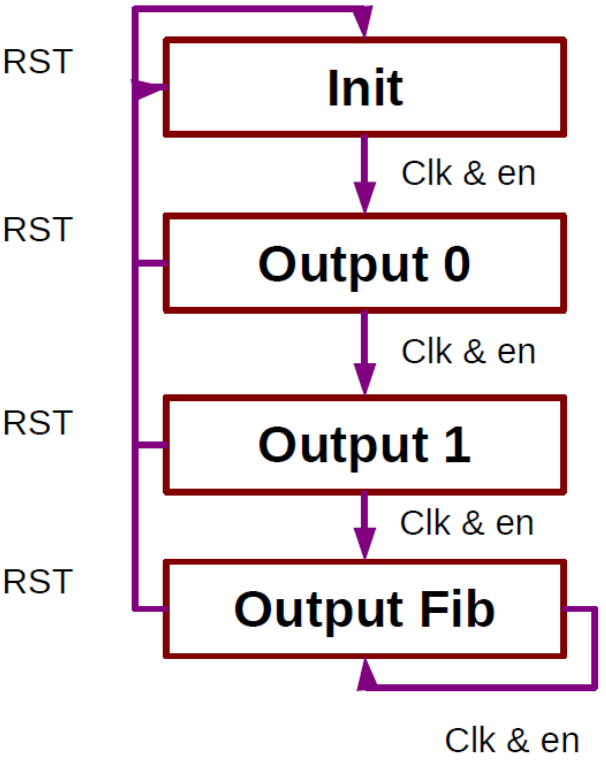
ToDo Lecture 11

C. Implement a FSM outputting the Fibonacci numbers
Reminder: 0 – 1 – 1 – 2 – 3 – 5 – … – (n-1 + n-2)

- States:
  - 1 init
  - 2 output 0
  - 3 output 1
  - 4 create Fibonacci number
  - 5 keep output

  5 can be ignored depending on your implementation

- Events:
  - 1: CLK = btnD
  - 2: en
  - 3: Reset



There are various ways, how to approach this.

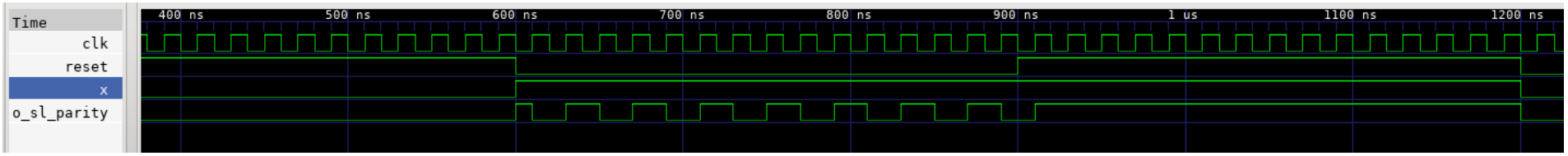
Please let me know, whether you completed C!

There is time to continue this exercise today…

Lecture 11 a

```
SYNC_PROC : process (i_sl_clk) begin
    if rising_edge(i_sl_clk) then
        if (i_sl_reset = '1') then
            state <= S0;
        else
            state <= next_state;
        end if;
    end if;
end process;
```

```
OUTPUT_DECODE : process (state, i_sl_x) begin
    o_sl_parity <= '0';
    case (state) is
        when S0 =>
            if (i_sl_x = '1') then
                o_sl_parity <= '1';
            end if;
        when S1 =>
            if (i_sl_x = '0') then
                o_sl_parity <= '1';
            end if;
        when others =>
            o_sl_parity <= '0';
    end case;
end process;

NEXT_STATE_DECODE : process (state, i_sl_x) begin
    next_state <= S0;
    case (state) is
        when S0 =>
            if (i_sl_x = '1') then
                next_state <= S1;
            end if;
        when S1 =>
            if (i_sl_x = '0') then
                next_state <= S1;
            end if;
        when others =>
            next_state <= S0;
    end case;
end process;
```

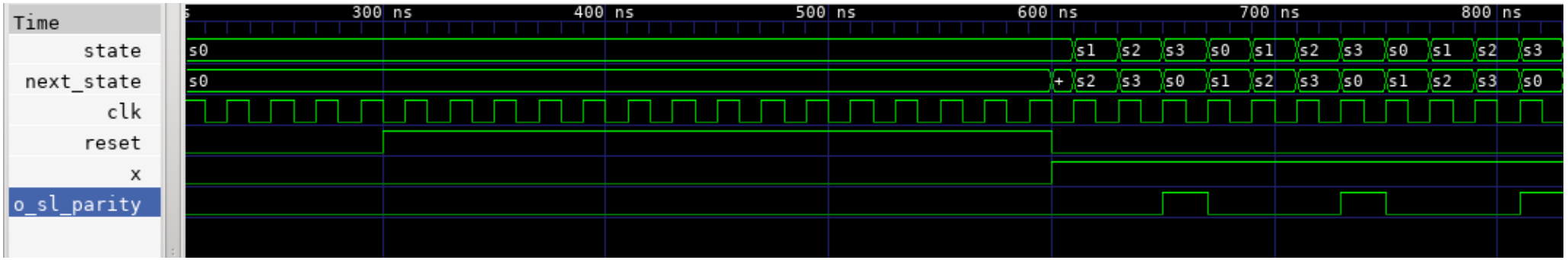


Lecture 11 b

```
architecture behavior of FSM_02 is

    type state_type is (S0, S1, S2, S3);
    signal state, next_state : state_type;

    begin

    SYNC_PROC : process (i_sl_clk) begin
        if rising_edge(i_sl_clk) then
            if (i_sl_reset = '1') then
                state <= S0;
            else
                state <= next_state;
            end if;
        end if;
    end process;

    OUTPUT_DECODE : process (state, i_sl_x) begin
        o_sl_parity <= '0';
        case (state) is
            when S3 =>
                o_sl_parity <= '1';
            when others =>
                o_sl_parity <= '0';
        end case;
    end process;
```

```
NEXT_STATE_DECODE : process (state, i_sl_x) begin
    next_state <= S0;
    case (state) is
        when S0 =>
            if (i_sl_x = '1') then
                next_state <= S1;
            end if;
        when S1 =>
            if (i_sl_x = '1') then
                next_state <= S2;
            end if;
        when S2 =>
            if (i_sl_x = '1') then
                next_state <= S3;
            end if;
        when S3 =>
            if (i_sl_x = '1') then
                next_state <= S0;
            end if;
        when others =>
            next_state <= S0;
    end case;
end process;
```

Lecture 11 b


Time
state
next_state
clk
reset
x
o_sl_parity
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
s0
s1
s2
s3
s0
s1
s2
s3
s0
s1
s2
s3
s0
+
s2
s3
s0
s1
s2
s3
s0
s1
s2
s3
s0


NEW

VHDL

III. Physikalisches Institut B
RWTH AACHEN UNIVERSITY

Using a component

- In the past, we always used
  COMPONENT test

  UUT : test
  PORT MAP

- There is an alternative way of doing this

  UUT : entity work.test(behavior) <- this is using the behavior architecture of module test, which is the library work (our default work space)

Useful, if you have many inputs / outputs

Reset

How to reset?

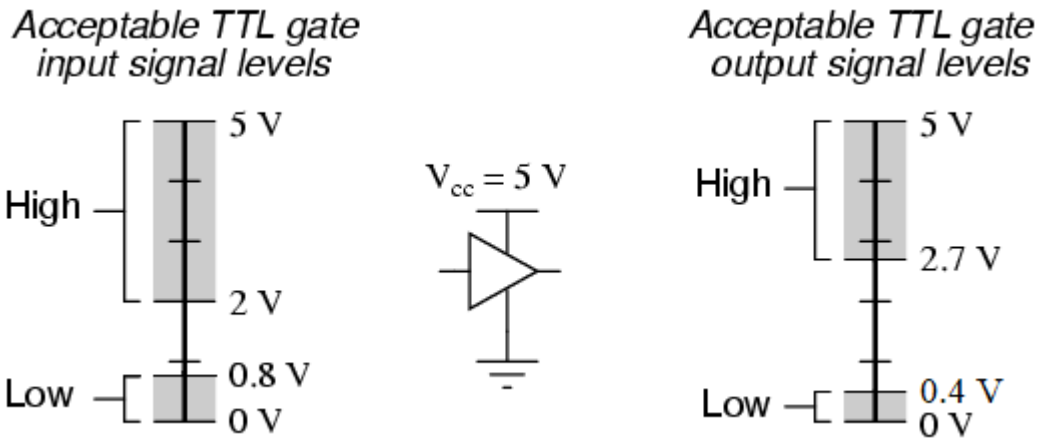
- Basically there are two ways of how to reset?
  - Reset if reset signal is = 1
  - Reset if reset signal is = 0 (sometimes also called non reset / nReset / nRST)
- Both are doing their job
- Why is resetting on 0 is a smart way?

Not reset?

- When to reset?
  - Power on, to get a well defines state
    - Note: setting a default value for signals / variables does just work in the simulation.
    - Inside the FPGA you might end up with a strange behavior, if not using reset.
  - Any situation, which requires a reset…

Not reset!

Power on reset by just two additional components

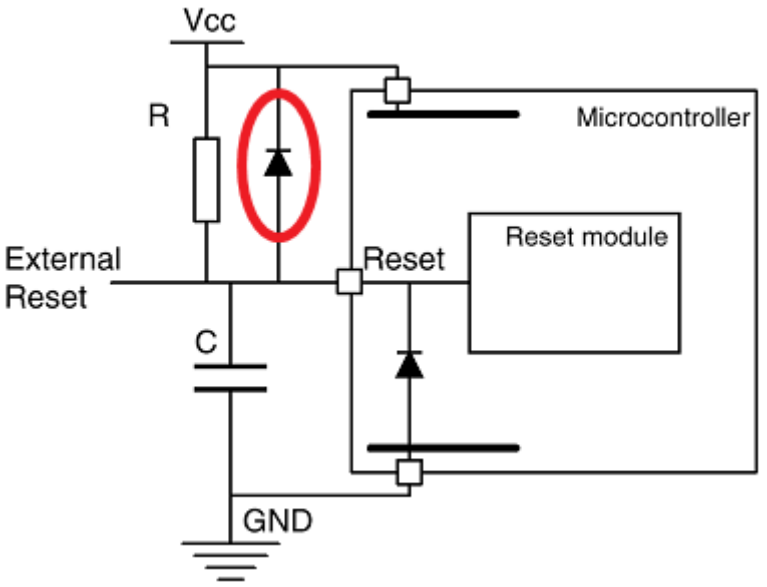


Most of the ICs used in electronics are using a non-reset.

It is easier to use.

Inside the FPGA it does not matter, which type of reset you are using

Figure 3-1. Recommended Reset Pin connection.



Thinking in states…

- Example: Exercise 11B
  - The output yout is 1 if and only if the total number of 1s received is divisible by 3 (hint: 0 is inclusive, however, reset cycle(s) do not count as 0- see in simulation waveform time=200).

- Which states can we identify?
  a) Zero ones
  b) Single one
  c) Double one
  d) Triple one

- Which transitions are needed?
  - A -> B -> C -> D -> A

- When to do the transition?
  - If input x = 1

Finite State Machines

Doing different tasks

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

Finite State Machine

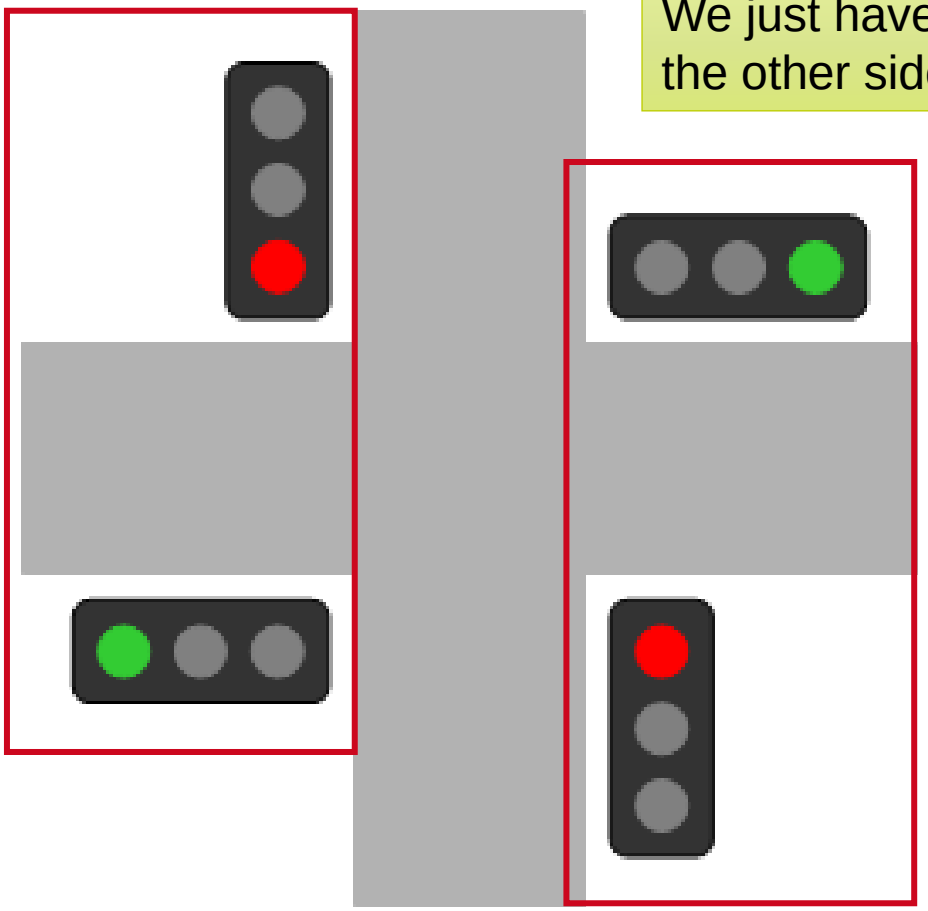
- The state changing part…

```
process(Clk) is
begin
    if rising_edge(Clk) then
        if nRst = '0' then
            State <= <reset_state>;
        else
            case State is
                when <state_name> => <set_outputs_for_this_state_here>
                    if <state_change_condition_is_true> then
                        State <= <next_state_name>;
                    end if;
                ...
            end case;
        end if;
    end if;
end process;
```

Continuation of FSM

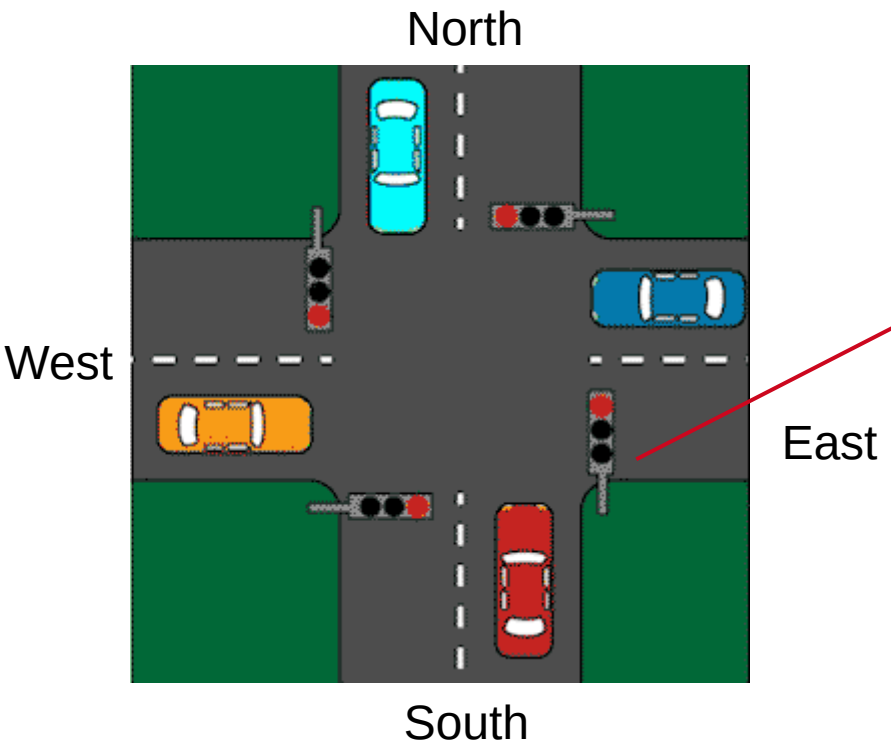
Situation given

- Intersection with traffic lights



We just have to care about one side, the other side is identical

How does the intersection behave?



Traffic lights:

- red
- red – orange
- green
- orange
- red

Intersection:

All red – one directions gets green – All red – next direction gets green.

Designing the FSM

- Think about which states can you observe / think of
- Which transitions are needed?
- Draw a diagram of the FSM (this helps implementing the states)
- Implement the FSM
- Do you have any idea how to increase the length of the green state?

ToDo Lecture 12

A. Implement the intersections FSM

B. Continue implementation of last weeks Fibonacci FSM

Few words regarding the testbench skeleton

```
-- Testbench sequence
process is
begin
    wait until rising_edge(Clk);    -- wait for rising edge
    wait until rising_edge(Clk);    --

    -- Take the DUT out of reset
    nRst <= '1';

    wait;   -- wait forever
end process;
```

Thank you!