Instituto Tecnológico de Aeronáutica

Divisão de Ciência da Computação - IEC

Departamento de Sistemas de Computação - IEC-SC

EEA-25 - Sistemas Digitais Programáveis

Lista de Exercícios Nº 6

Prof. André da Fontoura Ponchet

São José dos Campos - SP

23 de outubro de 2023

Sumário

1 Exercise 1 - Modeling a Generic Counter (5,0)

The VeriRISC CPU contains a program counter and a phase counter. One generic counter definition can serve both purposes.

In this exercise, you create a simple register design as per the specification and verify it using the provided testbench. Please make sure to use the same variable name as the ones shown in block diagram:

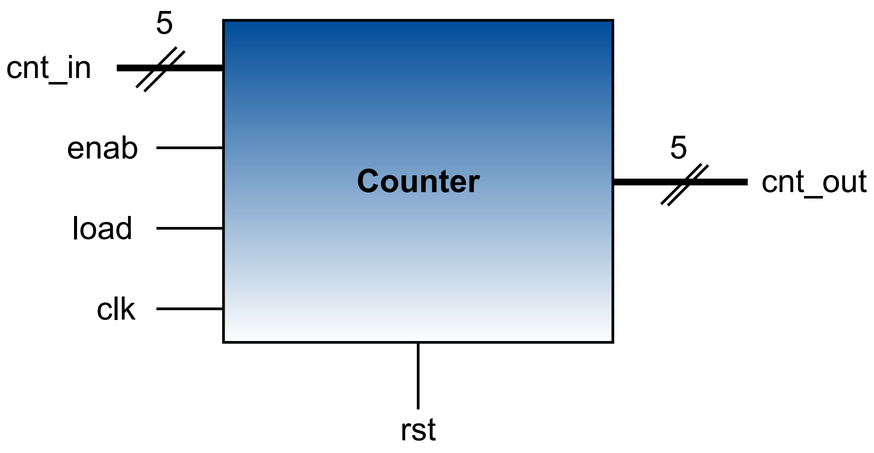

Figura 1: Counter diagram.

① Specifications:

- The counter is clocked on the rising edge of clk.
- rst is active high.
- cnt_in and cnt_out are both 5-bit signals.
- If rst is high, output will become zero.
- If load is high, the counter is loaded from the input cnt_in.
- Otherwise, if enab is high, cnt_out is incremented, and cnt_out is unchanged.

② Designing a Generic Counter

- Examine the file ***counter_test.v***. Use your favorite editor to create the counter.v file and to describe the counter module named as “counter”.
- Parameterize the counter data input and output width so that the instantiating module can specify the width of each instance. Assign a default value to the parameter.
- Describe the counter behavior in separate combinational and sequential procedures.

③ Verifying the Counter Design

- Check counter design using the following commands in the terminal:

$$\begin{cases} iverilog\ counter\_test.v \\ vvp\ a.out \\ gtkwave\ counter\_test.vcd \end{cases}$$

- You should see the following results:

```
At time 20 rst=0 load=1 enab=1 cnt\_in=10101 cnt\_out=10101
At time 30 rst=0 load=1 enab=1 cnt\_in=01010 cnt\_out=01010
At time 40 rst=0 load=1 enab=1 cnt\_in=11111 cnt\_out=11111
At time 50 rst=1 load=1 enab=1 cnt\_in=11111 cnt\_out=00000
At time 60 rst=0 load=1 enab=1 cnt\_in=11111 cnt\_out=11111
At time 70 rst=0 load=0 enab=1 cnt\_in=11111 cnt\_out=00000
TEST PASSED
```

- Present terminal and *gtkwave* screenshots.

2 Exercise 2 - Modeling the Counter Using Functions (5,0)

You typically use a function to encapsulate an operation performed multiple times with multiple different sets of operands. Encapsulating functionality reduces code “bloat” to make it more understandable and thus more reusable.

In this exercise, you create a counter design as per the specification using functions to practice the construct and verify it using the provided testbench. Please make sure to use the same variable name as the ones shown in Figure 1.

1 Specifications:

- The same specifications from exercise 1.

2 Designing a Counter

- Examine the following files: ***counter_test.v*** and ***counter.v*** (incomplete counter module).
- Use your favorite editor to modify the counter.v file name it as “counter”.
- Describe the combinational logic of the counter using functional block. You can use an ’if’ construct to model the counter combinational block.

3 Verifying the Counter Design

- Check counter design using the following commands in the terminal:

$$\begin{cases} iverilog\ counter\_test.v \\ vvp\ a.out \\ gtkwave\ counter\_test.vcd \end{cases}$$

- You should see the following results:

```
At time 20 rst=0 load=1 enab=1 cnt\_in=10101 cnt\_out=10101
At time 30 rst=0 load=1 enab=1 cnt\_in=01010 cnt\_out=01010
At time 40 rst=0 load=1 enab=1 cnt\_in=11111 cnt\_out=11111
At time 50 rst=1 load=1 enab=1 cnt\_in=11111 cnt\_out=00000
At time 60 rst=0 load=1 enab=1 cnt\_in=11111 cnt\_out=11111
At time 70 rst=0 load=0 enab=1 cnt\_in=11111 cnt\_out=00000
TEST PASSED
```

- Present terminal and *gtkwave* screenshots.