



---

**ASSIGNMENT 02**  
UP / DOWN COUNTER

---

**EMBEDDED SYSTEMS 3**

FACHHOCHSCHULE VORARLBERG  
MASTER MECHATRONICS

EINGEREICHT BEI

DR. ANDRÈ MITTERBACHER

VORGELEGT VON

ROMAN PASSLER

DORNBIRN, 31.12.2017

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>II</b>
<b>Tabellenverzeichnis</b>	<b>III</b>
<b>Listings</b>	<b>IV</b>
<b>1 Up / Down Counter</b>	<b>1</b>
1.1 Einleitung . . . . .	1
1.2 Implementierung . . . . .	1
1.3 Test Bench . . . . .	2
1.4 Simulationsscript . . . . .	4
1.5 Transkript und Waveform Window . . . . .	5
1.6 Vor- und Nachteile der Implementierung . . . . .	6

# Abbildungsverzeichnis

1.1	Waveform Window . . . . .	5
-----	---------------------------	---

# Tabellenverzeichnis

# Listings

1.1	Implementierung . . . . .	1
1.2	Testbench für den Up / Down Counter . . . . .	2
1.3	Simulationsscript . . . . .	4
1.4	Commandline Output A . . . . .	5

# 1 Up / Down Counter

## 1.1 Einleitung

In der zweiten Aufgabe soll ein n-Bit Up / Down Counter implementiert werden. Folgende Anforderungen werden an den Up / Down Counter gestellt:

- synchrone Logik  $\rightarrow$  „clk50m“ ist die Clock aller Flipflops.
- Aktiver Low-Reset auf Null („rst\_n“).
- Zähler wird inkrementiert oder dekrementiert, wenn  $en == 1'b1$ .
  - If ( $down == 1'b1$ )  $\rightarrow cnt = cnt - 1$
  - sonst  $\rightarrow cnt = cnt + 1$

Zusätzlich zu den Anforderungen muss die Implementierung getestet und verifiziert werden. Dies bedeutet, dass alle Eingänge stimuliert und alle Signale im „Wave Window“ angezeigt werden.

## 1.2 Implementierung

Die Implementierung erfolgt wie in Listing 1.1 dargestellt ist. In Codezeile Zeile 13 wird der Ausgang Counter „cnt“ mit variabler Bitlänge definiert definiert.

```
1  /*-----
2  Project:      Up Down Cunter
3  Purpose:      Flexible up down counter
4  Author:       rpa2306
5  Version:      00, 31.12.2017
6  -----*/
7
8  module counter_updn #(parameter WIDTH=8) (
9      input  logic          rst_n ,
10     input  logic          clk50m , // 50 Mhz Clock
11     input  logic          en ,
12     input  logic          down ,
13     output logic [WIDTH-1:0] cnt //
14 );
15
16 // ----- n-bit counter up and down -----
17
18 always_ff @ (negedge rst_n or posedge clk50m) begin
19     if (!rst_n) begin
20         cnt <= 1'b0;
21     end
22     else if (en && !down) begin
```

```
23         cnt <= cnt + 1'b1;
24     end
25     else if (en && down) begin
26         cnt <= cnt - 1'b1;
27     end
28 end
29
30 endmodule
```

Listing 1.1: Implementierung

## 1.3 Test Bench

In Listing 1.2 ist die Testbench ersichtlich. Es wurde ein automatisiertes Testscript erstellt, welches einen Fehler ausgibt, sobald der Up oder Down Test fehlschlägt (Codezeile 98 bis 104 und 121 bis 128).

```
1  /*-----
2  Project:    Up Down Cunter
3  Purpose:    Flexible up down counter
4  Author:     rpa2306
5  Version:    00, 31.12.2017
6  -----*/
7  `timescale 10ns/10ns
8
9  module tb_counter_updn();
10
11  `define counterSize 6
12
13  // (1) Prepare DUT wiring
14
15  logic          rst_n;
16  logic          clk50m;
17  logic          en;
18  logic          down;
19  logic  ['counterSize-1:0] cnt;
20
21  // (2) Instantiate the DUT
22
23  counter_updn #(.WIDTH ('counterSize)) dut (.*);
24
25  // (3) Create test stimuli
26  logic run_sim = 1'b1;
27
28  // all initial are running in parallel
29  initial begin
30      clk50m = 1'b0;
31      while (run_sim) begin
32          #10ns;
33          clk50m = !clk50m;
34      end
35  end
36
37  initial begin
38      automatic int cntSoftware = 0;
39      automatic int cntHardware = 0;
40      automatic int overflowCnt = 0;
```

```

41     automatic int maxValue = 2 << ('counterSize-1);
42     automatic int startValue = 0;
43     automatic int countUp = 100;
44     automatic int countDown = 200;
45
46     rst_n = 1'b0;
47     en = 1'b0;
48     down = 1'b0;
49
50     $display("-----");
51     $display(" TB_COUNTER_UPDN started ");
52     $display("-----");
53     $display("");
54
55     #100ns;
56
57     $display(" Check: en == 1'b1 and down == 1'b0\n");
58     down = 1'b0;
59     rst_n = 1'b1;
60     @ (negedge clk50m); // wait for negedge
61     en = 1'b1;
62     @ (negedge clk50m); // wait for negedge
63     en = 1'b0;
64     rst_n = 1'b0;
65     @ (negedge clk50m);
66
67     #100ns;
68
69     $display(" Check: en == 1'b1 and down == 1'b1\n");
70     down = 1'b1;
71     @ (negedge clk50m); // wait for negedge
72     en = 1'b1;
73     rst_n = 1'b1;
74     @ (negedge clk50m); // wait for negedge
75     en = 1'b0;
76     rst_n = 1'b0;
77     @ (negedge clk50m);
78
79     #100ns;
80
81     rst_n = 1'b1;
82     down = 1'b0;
83     en = 1'b1;
84     $display("---- Stimulate the up count for 100 ----\n");
85
86     startValue = cnt;
87     repeat (countUp) begin
88         @ (negedge clk50m); // wait for negedge
89         cntSoftware++;
90         if (cnt == maxValue-1) begin
91             overflowCnt++;
92         end
93     end
94
95     en = 1'b0;
96     down = 1'b1;
97     cntHardware = overflowCnt * maxValue + cnt + 1 - startValue;
98     if (cntHardware == cntSoftware) begin
99         //
100         $display("---- Test passed with counting up: %d ----\n",
101                 countUp);

```



```

100     end
101   else begin
102     $display("—— Test failed with counting up: %d ——", countUp);
103     $display("—— cntHardware %d, cntSoftware %d ——\n",
104             cntHardware, cntSoftware);
105   end
106   //
107   #100ns;
108   cntSoftware = 0;
109   overflowCnt = 0;
110   startValue = cnt;
111   @ (negedge clk50m);
112   en = 1'b1;
113   $display("—— Stimulate the down count for 200 ——\n");
114   repeat (countDown) begin
115     @ (negedge clk50m); // wait for negedge
116     cntSoftware++;
117     if (cnt == 0) begin
118       overflowCnt++;
119     end
120   end
121   cntHardware = (overflowCnt - 1) * maxValue + maxValue - cnt +
122               startValue;
123   if (cntHardware == cntSoftware) begin
124     //
125     $display("—— Test passed with counting up: %d ——\n",
126             countUp);
127   end
128   else begin
129     $display("—— Test failed with counting up: %d ——", countDown);
130     ;
131     $display("—— cntHardware %d, cntSoftware %d ——\n",
132             cntHardware, cntSoftware);
133     $display("");
134   end
135   //
136   en = 1'b0;
137   rst_n = 1'b0;
138   run_sim = 1'b0; // stop clock generator
139
140   $display("—————");
141   $display(" TB_COUNTER_UPDN finished ");
142   $display("—————");
143 end
144 endmodule

```

Listing 1.2: Testbench für den Up / Down Counter

## 1.4 Simulationsscript

In Listing 1.3 ist das Simulationsscript dargestellt. Es beinhaltet dieselben Befehle wie in der letzten Lehrveranstaltung, natürlich angepasst an den Up / Down Counter.

```

1 # Create simulation environment
2 vlib work
3 vmap work work

```



```
18 |# TB_COUNTER_UPDN finished
19 |# _____
```

Listing 1.4: Commandline Output A

## 1.6 Vor- und Nachteile der Implementierung

Folgend sind die Vor- und Nachteile der Implementierung gelistet:

### Vorteile

- Variable Bitlänge

### Nachteile

- Die Clock Frequenz darf nur so hoch gewählt werden, wie es das „Back Propagation Delay“ erlaubt. Somit ist die maximale Zählfrequenz begrenzt.
- Bei Anwahl des Down-Bits sollte beim Reseten der Counter Wert nicht auf „0“ initialisiert werden, sondern mit dem maximalen Wert

$$2 \ll ('counterSize - 1) - 1 \quad (1.1)$$

- Ebenfalls wäre es gut, den Startwert des Counters selbst wählen zu können.