



ASSIGNMENT 04

DAC

EMBEDDED SYSTEMS 3

FACHHOCHSCHULE VORARLBERG

MASTER MECHATRONICS

EINGEREICHT BEI

DR. ANDRÈ MITTERBACHER

VORGELEGT VON

ROMAN PASSLER

DORNBIRN, 14.01.2018

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	III
Listings	IV
1 Debouncer	1
1.1 Einleitung	1
1.2 Implementierung	1
1.3 Test Bench	3
1.4 Simulationsscript	6
1.5 Transkript und Waveform Window	6

Abbildungsverzeichnis

1.1	Waveform Window	7
-----	---------------------------	---

Tabellenverzeichnis

Listings

1.1	Implementierung	2
1.2	Testbench	4
1.3	Simulationsscript	6

1 Debouncer

1.1 Einleitung

Der Switch, der für den Betrieb der Stoppuhr bestimmt ist, muss entprellt werden, da ansonsten die Statemachine nicht ordnungsgemäß funktioniert. Dieses Beispiel entprellt die „high“ sowie „low“ Flanke.

- Wenn der Switch „low“ ist, wird der erste Counter gestartet von einem Initialwert (alle Bits auf eins). Das Signal ist solange gültig, bis der Counter null erreicht hat. Wenn der Switch „high“ schaltet, so wird der erste Counter sofort rückgesetzt auf seinen Initialwert.
- Wenn der Switch „high“ ist, wird der zweite Counter gestartet von einem Initialwert (alle Bits auf eins). Das Signal ist solange gültig, bis der Counter null erreicht hat. Wenn der Switch „low“ schaltet, so wird der zweite Counter sofort rückgesetzt auf seinen Initialwert.
- Die Bitbreite des Counters muss in einem Header File mit einem Makro („BITS“) definiert werden. Für diese Implementierung wird „BITS“ auf drei gesetzt.
- Die Flankenerkennung der fallenden und steigenden Flanke des entprellten Signals wird als „sw_hi“ und „sw_lo“ ausgegeben. Beide Ausgänge sind nur für einen Clock Zyklus auf „high“.

1.2 Implementierung

Als erstes wurde eine Implementierung mit einem Flipflop erstellt¹. Da dies unübersichtlich und für das Field Programmable Gate Array (FPGA) zu viele Bauelemente beinhaltet, wurde versucht die Aufgaben in Flipflops aufzuteilen. Die Implementierung erfolgt wie in Listing 1.1 dargestellt ist.

In Codezeile 32 bis 46 ist der Counter für die positive Flanke in einem Flipflop dargestellt. In Codezeile 66 bis 79 ist der Counter für die negative Flanke in einem Flipflop dargestellt.

Um den Übergang der Counter auf null zu erkennen wurde jeweils für die zwei Counter ein eigenes Flipflop erstellt, dies ist in Codezeile 49 bis 59 (positive Flanke) und in Codezeile 82 bis 92 (negative Flanke) dargestellt.

Für den entprellten Ausgang wurde ebenfalls ein eigenes Flipflop implementiert, dies ist in Codezeile 97 bis 107 zu sehen.

¹Siehe https://github.com/pasrom/Hardware-Description-Languages/tree/master/code/debounce_2

Die Ausgänge „sw_lo_cnt_zero“ und „sw_hi_cnt_zero“ werden mit den jeweiligen Countern zugewiesen, wenn der Counter null ist, so ist der Ausgang eins.

```

1  /*-----
2  Project:    debounce
3  Purpose:
4  Author:     rpa2306
5  Version:    00, 01.06.2018
6  -----*/
7
8  'include "debounce.sh"
9
10 module debounce (
11     input    logic          rst_n ,
12     input    logic          clk50m ,
13     input    logic          sw ,
14     output   logic          sw_hi ,
15     output   logic          sw_lo ,
16     output   logic          sw_dbnc
17 );
18
19 // (1) counter for high
20 logic ['BITS-1:0]  sw_hi_cnt;
21 logic              sw_hi_cnt_zero;
22 logic              hi_edge;
23
24 // (2) counter for low
25 logic ['BITS-1:0]  sw_lo_cnt;
26 logic              sw_lo_cnt_zero;
27 logic              lo_edge;
28
29 // (1) Implementation of counter high
30 assign sw_hi_cnt_zero = ~|sw_hi_cnt;
31
32 always_ff @ (negedge rst_n or posedge clk50m) begin //
33 end
34     if (~rst_n) begin
35         sw_hi_cnt <= '1;
36     end
37     else if (sw & sw_hi_cnt_zero) begin
38         sw_hi_cnt <= '0;
39     end
40     else if (sw) begin
41         sw_hi_cnt <= sw_hi_cnt - 'BITS'd1;
42     end
43     else begin
44         sw_hi_cnt <= '1;
45     end
46 end //
47
48 // edge detection up
49 always_ff @ (negedge rst_n or posedge clk50m) begin //
50     if (~rst_n) begin
51         hi_edge = 1'b0;
52     end
53     else if (sw_hi_cnt == 'BITS'd1) begin
54         hi_edge = 1'b1;
55     end
56     else begin
57         hi_edge = 1'b0;
58     end

```

```

59     end //
60
61     assign sw_hi = (sw_hi_cnt_zero & hi_edge) ;
62
63 // (2) Implementation of counter low
64     assign sw_lo_cnt_zero = ~|sw_lo_cnt;
65
66     always_ff @ (negedge rst_n or posedge clk50m) begin //
67         if (~rst_n) begin
68             sw_lo_cnt <= '1;
69         end
70         else if (~sw & sw_lo_cnt_zero) begin
71             sw_lo_cnt <= '0;
72         end
73         else if (~sw) begin
74             sw_lo_cnt <= sw_lo_cnt - 'BITS'd1;
75         end
76         else begin
77             sw_lo_cnt <= '1;
78         end
79     end //
80
81 // edge detection down
82     always_ff @ (negedge rst_n or posedge clk50m) begin //
83         if (~rst_n) begin
84             lo_edge = 1'b0;
85         end
86         else if (sw_lo_cnt == 'BITS'd1) begin
87             lo_edge = 1'b1;
88         end
89         else begin
90             lo_edge = 1'b0;
91         end
92     end //
93
94     assign sw_lo = (sw_lo_cnt_zero & lo_edge);
95
96 // (3) output FF
97     always_ff @ (negedge rst_n or posedge clk50m) begin //
98         if (~rst_n) begin
99             sw_dbnc <= 1'b0;
100        end
101        else if (sw_lo) begin
102            sw_dbnc <= 1'b0;
103        end
104        else if (sw_hi) begin
105            sw_dbnc <= 1'b1;
106        end
107    end //
108
109 endmodule

```

Listing 1.1: Implementierung

1.3 Test Bench

In Listing 1.2 ist die Testbench ersichtlich.


```

1  /*-----
2  Project:    debounce
3  Purpose:
4  Author:     rpa2306
5  Version:    00, 01.06.2018
6  -----*/
7
8  module tb_debounce();
9
10 /*
11 *.  (1) Create wires to connect the DUT
12 *.  Like footprints for an IC on a PCB
13 */
14
15 logic      rst_n;
16 logic      clk50m;
17 logic      sw;
18
19 logic      sw_hi;
20 logic      sw_lo;
21 logic      sw_dbnc;
22
23 /*
24 *   (2) Create an instance of the DUT
25 */
26
27 localparam TBWIDTH = 4;
28
29 debounce    #(.WIDTH (TBWIDTH))    DUT (. *);
30
31 /*
32 *   (3) Create stimuli for all inputs
33 */
34
35 logic run_sim = 1'b1;
36
37 initial begin: clk_gen
38     clk50m = 1'b0;
39
40     while(run_sim) begin
41         #10ns;
42         clk50m = !clk50m;
43     end
44 end
45
46 initial begin
47
48     $display("-----");
49     $display("    debounce started    ");
50     $display("-----");
51
52     @(negedge clk50m);
53     rst_n = 1'b0;
54     sw = 1'b0;
55     #50ns;
56     @(negedge clk50m);
57     rst_n = 1'b1;
58     #50ns;
59
60     @(negedge clk50m);

```

```
61     sw = 1'b0;
62     repeat (10) begin
63         @ (posedge clk50m);
64     end
65     #50ns;
66     @(negedge clk50m);
67     sw = 1'b1;
68
69     repeat (5) begin
70         @ (posedge clk50m);
71     end
72     @ (negedge clk50m);
73     sw = 1'b0;
74     //@ (posedge clk50m);
75
76     @ (negedge clk50m);
77     sw = 1'b1;
78     repeat (3) begin
79         @ (posedge clk50m);
80     end
81
82     @(negedge clk50m);
83     sw = 1'b0;
84     repeat (5) begin
85         @ (posedge clk50m);
86     end
87
88     @ (negedge clk50m);
89     sw = 1'b1;
90     repeat (2) begin
91         @ (posedge clk50m);
92     end
93
94     @ (negedge clk50m);
95     sw = 1'b0;
96
97     repeat (8) begin
98         @ (posedge clk50m);
99     end
100
101     #200ns;
102
103     @ (negedge clk50m);
104     sw = 1'b1;
105
106     repeat (8) begin
107         @ (posedge clk50m);
108     end
109
110     #200ns;
111
112
113     @ (negedge clk50m);
114     sw = 1'b0;
115
116     repeat (8) begin
117         @ (posedge clk50m);
118     end
119
120     #50ns;
121     @ (negedge clk50m);
```

```
122     run_sim = 0;
123     rst_n = 1'b0;
124     $display("_____");
125     $display("    debounce finished    ");
126     $display("_____");
127
128 end
129 endmodule
```

Listing 1.2: Testbench

1.4 Simulationsscript

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

```
1 # Create simulation environment
2 vlib work
3 vmap work work
4
5 # Compile desing files -> use file names
6 vlog ../src/debounce.sv
7 # Compile the test bench
8 vlog tb_debounce.sv
9 # Init simulation -> use module name
10 vsim tb_debounce
11 # -r recursive
12 log -r *
13 do wave_tb_debounce.tcl
14
15 # Run simulation
16 run -all
17 # run 100us
18 # Show results
19 view wave
```

Listing 1.3: Simulationsscript

1.5 Transkript und Waveform Window

In Abbildung 1.1 ist das Waveform Window dargestellt. Es zeigt die geforderten Testfälle. Wie zu sehen ist, wurden die Anforderungen erfüllt.

1 Debouncer

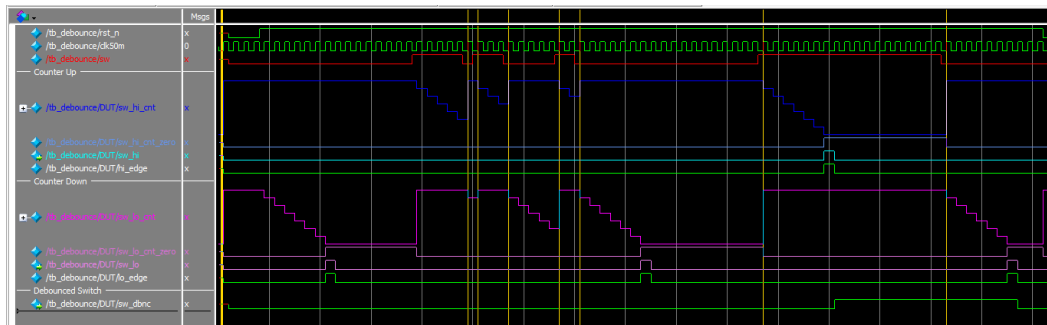


Abbildung 1.1: Waveform Window
Quelle: eigene Ausarbeitung