FH Vorarlberg
University of Applied Sciences

Assignment 03

Debouncer

Embedded Systems 3

Fachhochschule Vorarlberg

Master Mechatronics

Eingereicht bei

Dr. Andrè Mitterbacher

Vorgelegt von

Roman Passler

Dornbirn, 08.01.2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

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

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.

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.

```
/*-----------------------------------------
Project:    debounce
Purpose:
Author:     rpa2306
Version:    00, 01.06.2018
-----------------------------------------*/

`include "debounce.sh"

module debounce (
    input   logic           rst_n,
    input   logic           clk50m,
    input   logic           sw,
    output  logic           sw_hi,
    output  logic           sw_lo,
    output  logic           sw_dbnc
);

// (1) counter for high
    logic [`BITS-1:0]   sw_hi_cnt;
    logic               sw_hi_cnt_zero;
    logic               hi_edge;

// (2) counter for low
    logic [`BITS-1:0]   sw_lo_cnt;
    logic               sw_lo_cnt_zero;
    logic               lo_edge;

// (1) Implementation of counter high
    assign sw_hi_cnt_zero = ~|sw_hi_cnt;

    always_ff @ (negedge rst_n or posedge clk50m) begin //
    end
        if (~rst_n) begin
            sw_hi_cnt <= '1;
        end
        else if (sw & sw_hi_cnt_zero) begin
            sw_hi_cnt <= '0;
        end
        else if (sw) begin
            sw_hi_cnt <= sw_hi_cnt - `BITS'd1;
        end
        else begin
            sw_hi_cnt <= '1;
        end
    end                                                 //

// edge detection up
    always_ff @ (negedge rst_n or posedge clk50m) begin //
        if (~rst_n) begin
            hi_edge = 1'b0;
        end
        else if(sw_hi_cnt == `BITS'd1)begin
            hi_edge = 1'b1;
        end
        else begin
            hi_edge = 1'b0;
        end
    end                                                 //

```

```
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 debounce       DUT (.*);
28
29 /*
30 *   (3) Create stimuli for all inputs
31 */
32
33 logic run_sim = 1'b1;
34
35 initial begin: clk_gen
36     clk50m = 1'b0;
37
38     while(run_sim) begin
39         #10ns;
40         clk50m = !clk50m;
41     end
42 end
43
44 initial begin
45
46     $display("------------------------------");
47     $display("    debounce started    ");
48     $display("------------------------------");
49
50     @(negedge clk50m);
51     rst_n = 1'b0;
52     sw = 1'b0;
53     #50ns;
54     @(negedge clk50m);
55     rst_n = 1'b1;
56     #50ns;
57
58     @(negedge clk50m);
59     sw = 1'b0;
60     repeat (10) begin
61             @ (posedge clk50m);
62     end
63     #50ns;
```

```
64      @(negedge clk50m);
65      sw = 1'b1;
66
67      repeat (5) begin
68              @ (posedge clk50m);
69      end
70      @ (negedge clk50m);
71      sw = 1'b0;
72      //@ (posedge clk50m);
73
74      @ (negedge clk50m);
75      sw = 1'b1;
76      repeat (3) begin
77              @ (posedge clk50m);
78      end
79
80      @(negedge clk50m);
81      sw = 1'b0;
82      repeat (5) begin
83              @ (posedge clk50m);
84      end
85
86      @ (negedge clk50m);
87      sw = 1'b1;
88      repeat (2) begin
89              @ (posedge clk50m);
90      end
91
92      @ (negedge clk50m);
93      sw = 1'b0;
94
95      repeat (8) begin
96              @ (posedge clk50m);
97      end
98
99      #200ns;
100
101     @ (negedge clk50m);
102     sw = 1'b1;
103
104     repeat (8) begin
105             @ (posedge clk50m);
106     end
107
108     #200ns;
109
110
111     @ (negedge clk50m);
112     sw = 1'b0;
113
114     repeat (8) begin
115             @ (posedge clk50m);
116     end
117
118     #50ns;
119     @ (negedge clk50m);
120     run_sim = 0;
121     rst_n = 1'b0;
122     $display("------------------------------");
123     $display("    debounce finished     ");
124     $display("------------------------------");
```

```
125
126 end
127 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 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.

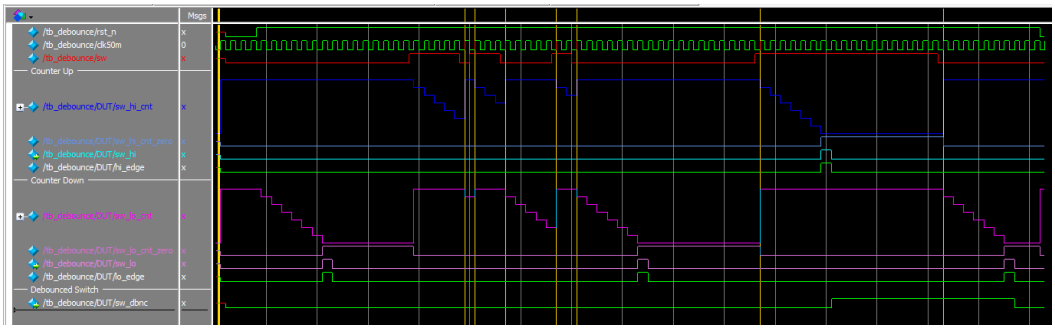

Abbildung 1.1: Waveform Window
Quelle: eigene Ausarbeitung