



---

**ASSIGNMENT 05**  
PROJEKTZUSAMMENFASSUNG

---

**EMBEDDED SYSTEMS 3**

FACHHOCHSCHULE VORARLBERG

MASTER MECHATRONICS

EINGEREICHT BEI

DR. ANDRÈ MITTERBACHER

VORGELEGT VON

FLORIAN BURTSCHER  
ROMAN PASSLER

DORNBIRN, 31.01.2018

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>II</b>
<b>Tabellenverzeichnis</b>	<b>III</b>
<b>Listings</b>	<b>IV</b>
<b>1 DAC</b>	<b>1</b>
1.1 Einleitung . . . . .	1
1.2 Top-Level Design . . . . .	1
1.3 Toplevel Design und Simulation . . . . .	2
1.4 Bench Verifikation . . . . .	7

# Abbildungsverzeichnis

1.1	Top-Level Design . . . . .	2
1.2	Top-Level Design . . . . .	2
1.3	Sinus Verifikation in der Simulation . . . . .	7
1.4	Tiefpass erster Ordnung . . . . .	8
1.5	Sinus Verifikation . . . . .	8
1.6	Spektrum des Sinussignals . . . . .	9
1.7	Sägezahn Verifikation . . . . .	9
1.8	Dreieck Verifikation . . . . .	10
1.9	Rechteck Verifikation . . . . .	10

# Tabellenverzeichnis

# Listings

1.1	Toplevel Design . . . . .	2
1.2	Testbench . . . . .	5

# 1 DAC

## 1.1 Einleitung

Um die entwickelte Logik in ein reales System einzubetten, müssen einige Aufgaben erledigt werden:

- Geräteauswahl
- Pinbelegung
- Fertige Module (IP) hinzufügen
- Toplevel-Routing
- Top-Level-Simulation
- Synthese des Projekts in die Zielhardware
- Testbench Überprüfung
- Validierung (wenn möglich)

## 1.2 Top-Level Design

In Abbildung 1.1 ist das Top-Level Design ersichtlich. Die Verbindungen der Clock („clk50m“) und des Reset Eingangs („rst\_n“) wurden der Übersichtlichkeit nicht eingezeichnet.

## 1 DAC

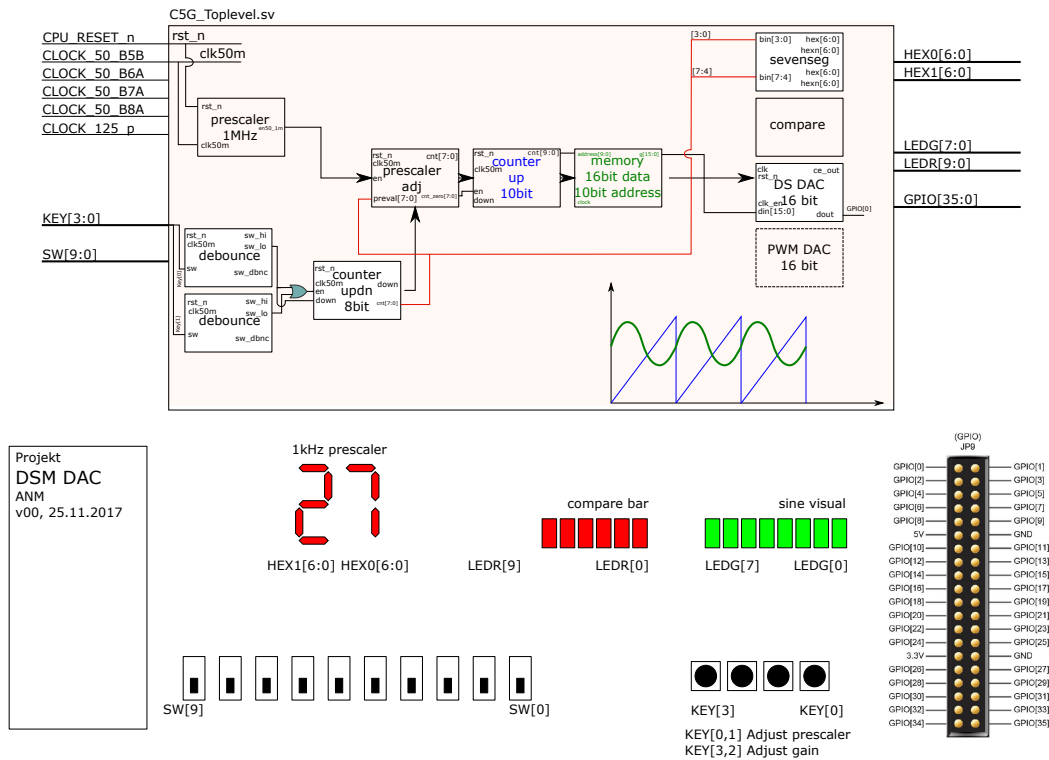


Abbildung 1.1: Top-Level Design  
Quelle: Mitterbacher Andr  und eigene Erweiterung

In Abbildung 1.2 ist das im Quartus implementierte Top-Level Design ersichtlich.

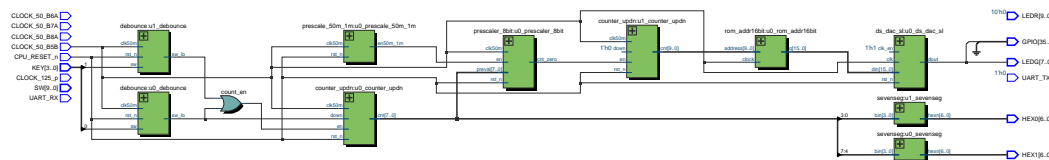


Abbildung 1.2: Top-Level Design  
Quelle: eigene Ausarbeitung

## 1.3 Toplevel Design und Simulation

In Listing 1.1 ist das Toplevel Design ersichtlich.

```

1  /* -----
2  Project : DS DAC
3  Purpose : Toplevel delta sigma DAC
4  Author  : ANM
5  Date   : 25.11.2017
6  ----- */
7
8  module toplevel_c5g_led_switch_7segx2_gpio_uart(
9
10
11

```

```

12  //////////// CLOCK ////////////
13  input  logic                                CLOCK_125_p,
14  input  logic                                CLOCK_50_B5B,
15  input  logic                                CLOCK_50_B6A,
16  input  logic                                CLOCK_50_B7A,
17  input  logic                                CLOCK_50_B8A,
18
19  //////////// LED ////////////
20  output logic [ 7:0 ] LEDG,
21  output logic [ 9:0 ] LEDR,
22
23  //////////// KEY ////////////
24  input  logic                                CPU_RESET_n,
25  input  logic [ 3:0 ] KEY,
26
27  //////////// SW ////////////
28  input  logic [ 9:0 ] SW,
29
30  //////////// SEG7 ////////////
31  output logic [ 6:0 ] HEX0,
32  output logic [ 6:0 ] HEX1,
33
34  //////////// Uart to USB ////////////
35  input  logic                                UART_RX,
36  output logic                                UART_TX,
37
38  //////////// GPIO, GPIO connect to GPIO Default ////////////
39  output logic [ 35:0 ] GPIO
40 );
41
42
43
44 //=====
45 // REG/WIRE declarations
46 //=====
47
48 logic                                rst_n;
49 logic                                clk50m;
50 logic                                ds_bitstream;
51 logic                                count_up;
52 logic                                count_low;
53 logic                                count_en;
54 logic [ 7:0 ] cnt_8bit;
55 logic                                en50m_lm;
56 logic                                cnt_zero;
57 logic [ 9:0 ] cnt_10bit;
58 logic [ 15:0 ] analogue_sin;
59
60 //=====
61 // Structural coding
62 //=====
63
64 // — Map outputs —
65
66 assign UART_TX = 1'b0;
67 assign LEDG = {8{ds_bitstream}};
68 assign LEDR[9:0] = '0;
69 assign GPIO[35:1] = '0;
70 assign GPIO [0] = ds_bitstream;
71
72 // — Map inputs —

```



```

73
74     assign                                rst_n      = CPU_RESET_n;
75     assign                                clk50m      = CLOCK_50_B5B;
76     assign                                count_en     = count_low || count_up
77         ;
78 // --- Modules ---
79
80 prescale_50m_1m u0_prescale_50m_1m(
81     .rst_n ,
82     .clk50m ,
83     .en50m_1m(en50m_1m)
84 );
85
86 debounce u0_debounce(
87     .rst_n ,
88     .clk50m ,
89     .sw(KEY[0]) ,
90     .sw_hi() ,
91     .sw_lo(count_low) ,
92     .sw_dbnc()
93 );
94
95 debounce u1_debounce(
96     .rst_n ,
97     .clk50m ,
98     .sw(KEY[1]) ,
99     .sw_hi() ,
100    .sw_lo(count_up) ,
101    .sw_dbnc()
102 );
103
104 counter_updn #(WIDTH (8)) u0_counter_updn(
105     .rst_n ,
106     .clk50m ,
107     .en(count_en) ,
108     .down(count_low) ,
109     .cnt(cnt_8bit)
110 );
111
112 prescaler_8bit u0_prescaler_8bit(
113     .rst_n ,
114     .clk50m ,
115     .en(en50m_1m) ,
116     .preval(cnt_8bit) ,
117     .cnt() ,
118     .cnt_zero(cnt_zero)
119 );
120
121 counter_updn #(WIDTH (10)) u1_counter_updn(
122     .rst_n ,
123     .clk50m ,
124     .en(cnt_zero) ,
125     .down(1'b0) ,
126     .cnt(cnt_10bit)
127 );
128
129 rom_addr16bit u0_rom_addr16bit(
130     .address(cnt_10bit) ,
131     .clock(clk50m) ,
132     .q(analogue_sin)

```

```

133 );
134
135 ds_dac_sl u0_ds_dac_sl(
136     .clk(clk50m) ,
137     .rst_n ,
138     .clk_en(1'b1) ,
139     .din(analogue_sin) ,
140     .ce_out() ,
141     .dout(ds_bitstream)
142 );
143
144 sevenseg u0_sevenseg(
145     .bin(cnt_8bit[7:4]) ,
146     .hex() ,
147     .hexn(HEX1[6:0])
148 );
149
150 sevenseg u1_sevenseg(
151     .bin(cnt_8bit[3:0]) ,
152     .hex() ,
153     .hexn(HEX0[6:0])
154 );
155
156 endmodule

```

Listing 1.1: Toplevel Design

In Listing 1.2 ist der Code der Testbench ersichtlich.

```

1  /* -----
2  Project : DS DAC
3  Purpose : Toplevel delta sigma DAC
4  Author  : ANM
5  Date    : 25.11.2017
6  ----- */
7  `timescale 10ns/10ns
8  module tb_toplevel_c5g_led_switch_7segx2_gpio_uart();
9      // (1) DUT wiring
10     /////////////// CLOCK ///////////////////
11     logic                                CLOCK_125_p;
12     logic                                CLOCK_50_B5B;
13     logic                                CLOCK_50_B6A;
14     logic                                CLOCK_50_B7A;
15     logic                                CLOCK_50_B8A;
16
17     /////////////// LED ///////////////////
18     logic [7:0]                          LEDG;
19     logic [9:0]                          LEDR;
20
21     /////////////// KEY ///////////////////
22     logic                                CPU_RESET_n;
23     logic [3:0]                          KEY;
24
25     /////////////// SW ///////////////////
26     logic [9:0]                          SW;
27
28     /////////////// SEG7 ///////////////////
29     logic [6:0]                          HEX0;
30     logic [6:0]                          HEX1;
31
32     /////////////// Uart to USB ///////////////////

```

```

33     logic                                UART_RX;
34     logic                                UART_TX;
35
36     //////////// GPIO; GPIO connect to GPIO Default ////////////
37     logic                                [35:0]    GPIO;
38
39
40     // (2) DUT instance
41     toplevel_c5g_led_switch_7segx2_gpio_uart    dut(. *);
42
43     // (3) DUT stimuli
44     logic run_sim = 1'b1;
45     int error_cnt = 0;
46     string action = "init";
47
48     // — Clocks and Reset —
49     initial begin : clk_gen_125m
50         CLOCK_125_p = 1'b0;
51         while (run_sim) begin
52             #4ns;
53             CLOCK_125_p = ~CLOCK_125_p;
54         end
55     end
56
57     initial begin : clk_gen_50m
58         CLOCK_50_B5B = 1'b0;
59         while (run_sim) begin
60             #10ns;
61             CLOCK_50_B5B = ~CLOCK_50_B5B;
62         end
63     end
64     assign CLOCK_50_B6A = CLOCK_50_B5B;
65     assign CLOCK_50_B7A = CLOCK_50_B5B;
66     assign CLOCK_50_B8A = CLOCK_50_B5B;
67
68     initial begin : rst_gen
69         CPU_RESET_n = 1'b0;
70         #99ns
71         CPU_RESET_n = 1'b1;
72     end
73
74     initial begin : load_memory
75         $readmemh("../fpga/Toplevel_C5G_led_switch_7segx2_gpio_uart/ip/
76             mem_sine_01.txt", dut.u0_rom_addr16bit.altsyncram_component.
77             m_default.altsyncram_inst.mem_data);
78     end
79
80     // — Stimulate inputs —
81
82     initial begin
83         $display("_____");
84         $display("tb_toplevel_c5g_led_switch_7segx2_gpio_uart started."
85             );
86         $display("_____");
87         KEY = '1;
88         SW = '0;
89         UART_RX = '0;
90         #1us;
91         action="Push KEY[0] 300 times";
92         $display("\t%s", action);

```

```

91     repeat(300) begin
92         @ (negedge CLOCK_50_B5B);
93         KEY[0] = 1'b0;
94         #1us;
95         @ (negedge CLOCK_50_B5B);
96         KEY[0] = 1'b1;
97     end
98
99     #100us;
100
101     action="Push KEY[1] 555 times";
102     $display("\t%s", action);
103     repeat(555) begin
104         @ (negedge CLOCK_50_B5B);
105         KEY[1] = 1'b0;
106         #1us;
107         @ (negedge CLOCK_50_B5B);
108         KEY[1] = 1'b1;
109     end
110     CPU_RESET_n = 1'b0;
111     #99ns
112     CPU_RESET_n = 1'b1;
113     #2000us;
114
115     run_sim = 1'b0;
116     $display("_____");
117     $display("tb_toplevel_c5g_led_switch_7segx2_gpio_uart finished.");
118     $display("_____");
119 end
120
121 endmodule

```

Listing 1.2: Testbench

Abbildung 1.3 zeigt den simulierten Sinus.

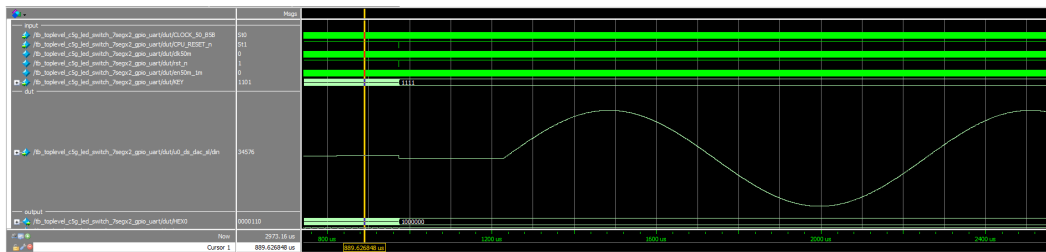


Abbildung 1.3: Sinus Verifikation in der Simulation  
Quelle: eigene Ausarbeitung

## 1.4 Bench Verifikation

Der Bitstream wird mit einem Tiefpassfilter erster Ordnung gefiltert (Abbildung 1.4).

$$f_g = \frac{1}{2 \cdot \pi \cdot R \cdot C} \quad (1.1)$$

Mit  $C = 1 \text{ nF}$  und  $R = 10 \text{ k}\Omega$  folgt für  $f_g$

$$f_g = 15,91 \text{ kHz} \quad (1.2)$$

Der erzeugte Sinus hat eine Frequenz von  $f_{max} = 976,56 \text{ Hz}$  unter der Annahme, dass der Prescaler eine Frequenz von  $10 \text{ MHz}$  und der gespeicherte Sinus 1024 Werte für eine Periode hat.

$$f_{max} = \frac{10^6 \text{ Hz}}{1024} \quad (1.3)$$

$$= 976,56 \text{ Hz} \quad (1.4)$$

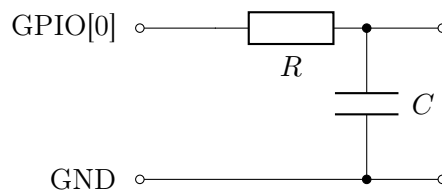


Abbildung 1.4: Tiefpass erster Ordnung  
Quelle: eigene Ausarbeitung

In Abbildung 1.5 ist der generierte Sinus dargestellt. Channel 1 zeigt die gemessene Spannung über die Zeit am Ausgang des Tiefpassfilters. Channel 2 zeigt den Bitstream am Ausgang des GPIO[0]. Die Funktion F1 zeigt das Tracking des Dutycycles. Durch das Ändern des Speichers lässt sich die Art des Ausgangsbitstromes und somit der generierten Spannung beliebig wählen. Abbildung 1.7 zeigt eine generierte Sägezahnspannung, Abbildung 1.8 zeigt eine generierte Dreiecksspannung und Abbildung 1.9 zeigt eine generierte Rechteckspannung.

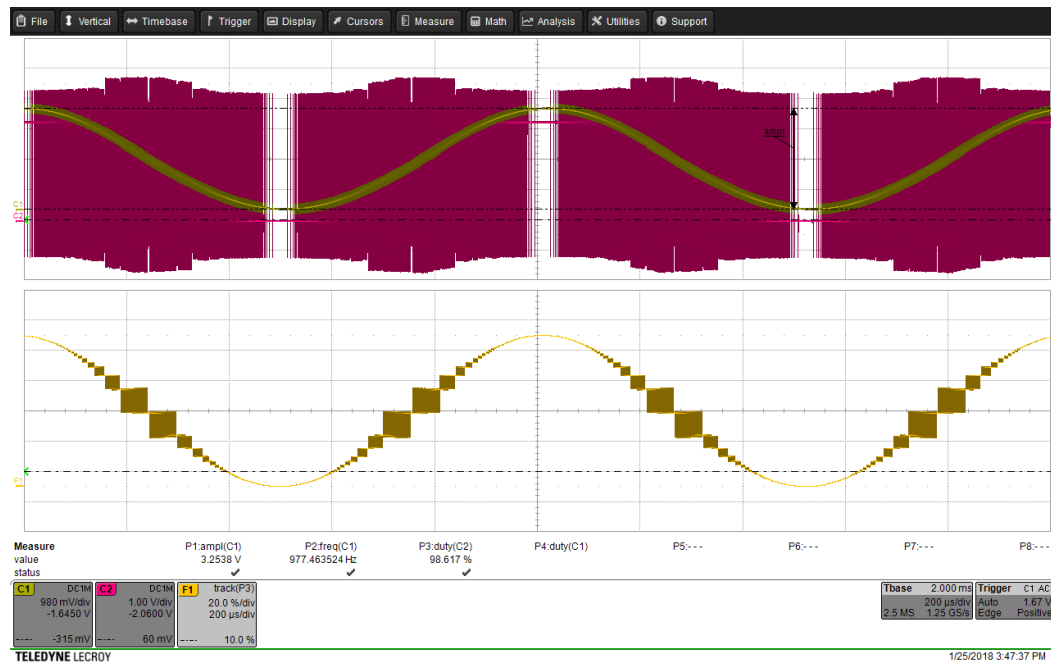


Abbildung 1.5: Sinus Verifikation  
Quelle: eigene Ausarbeitung

Abbildung 1.6 zeigt das Spektrum des erzeugten Sinussignals mit Prescaler 7. Die Grundschwingung mit  $139\text{ Hz}$  hat eine deutlich höhere Amplitude (siehe linke Tabelle) als die erste Oberschwingung mit  $279\text{ Hz}$ .

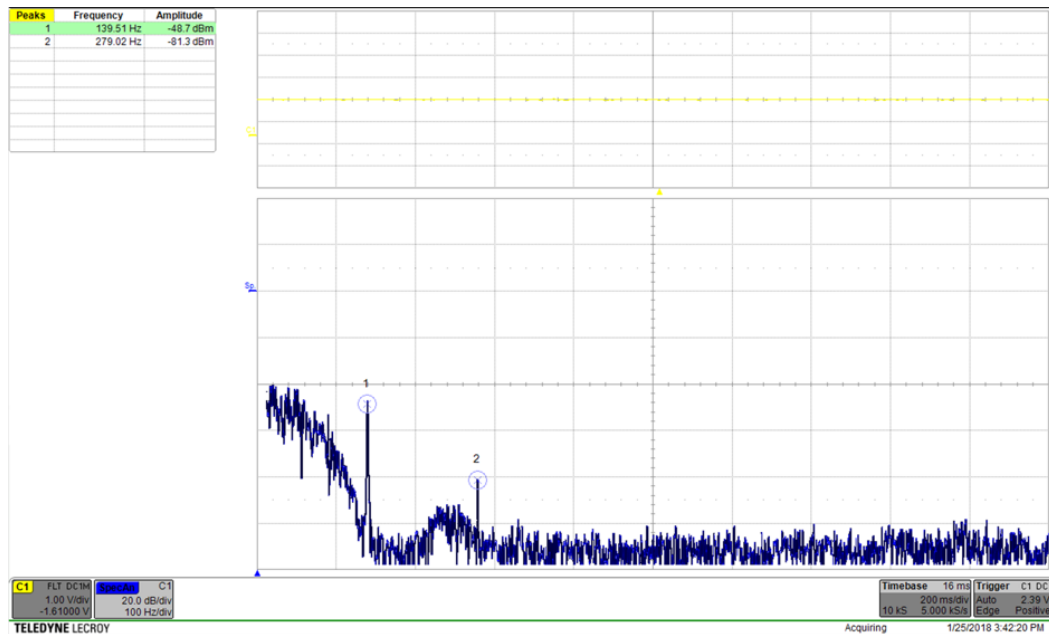


Abbildung 1.6: Spektrum des Sinussignals  
Quelle: Hermann Haag

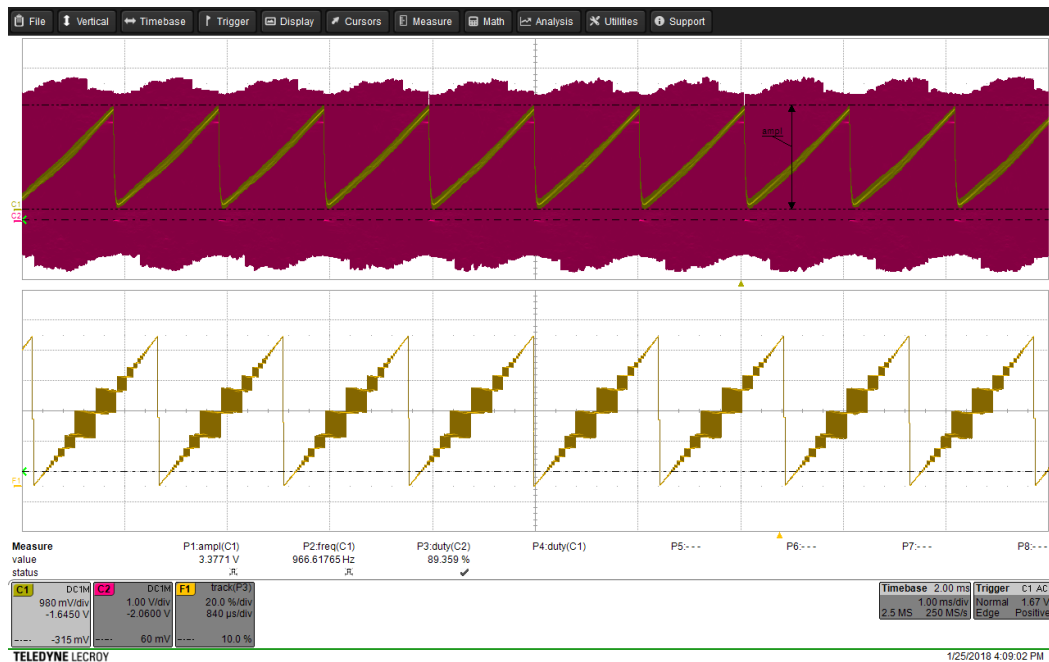


Abbildung 1.7: Sägezahn Verifikation  
Quelle: eigene Ausarbeitung

## 1 DAC

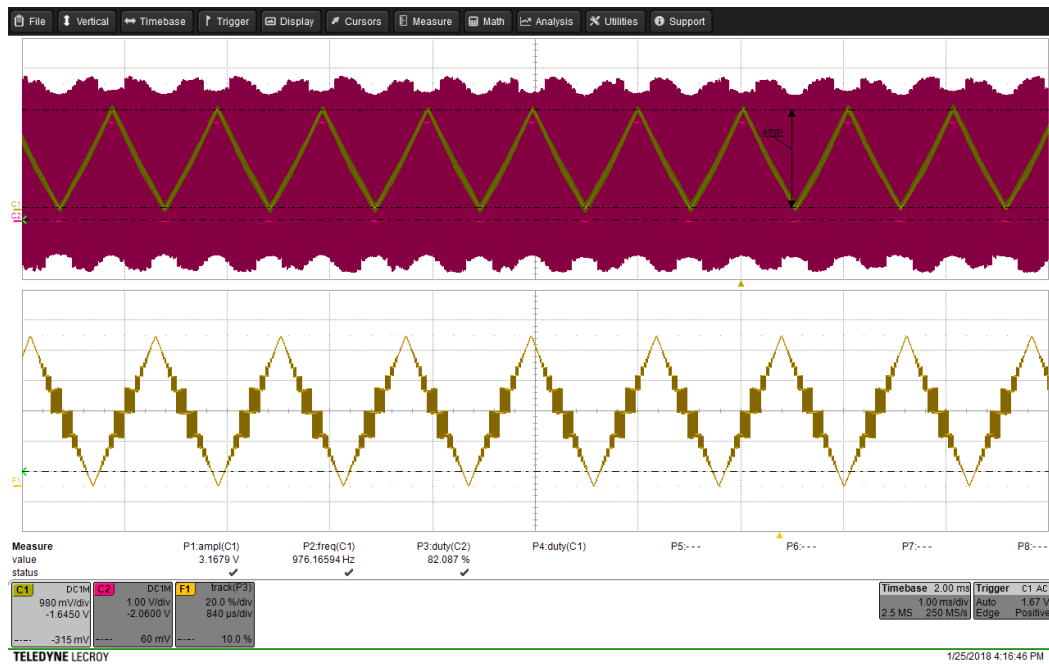


Abbildung 1.8: Dreieck Verifikation  
Quelle: eigene Ausarbeitung

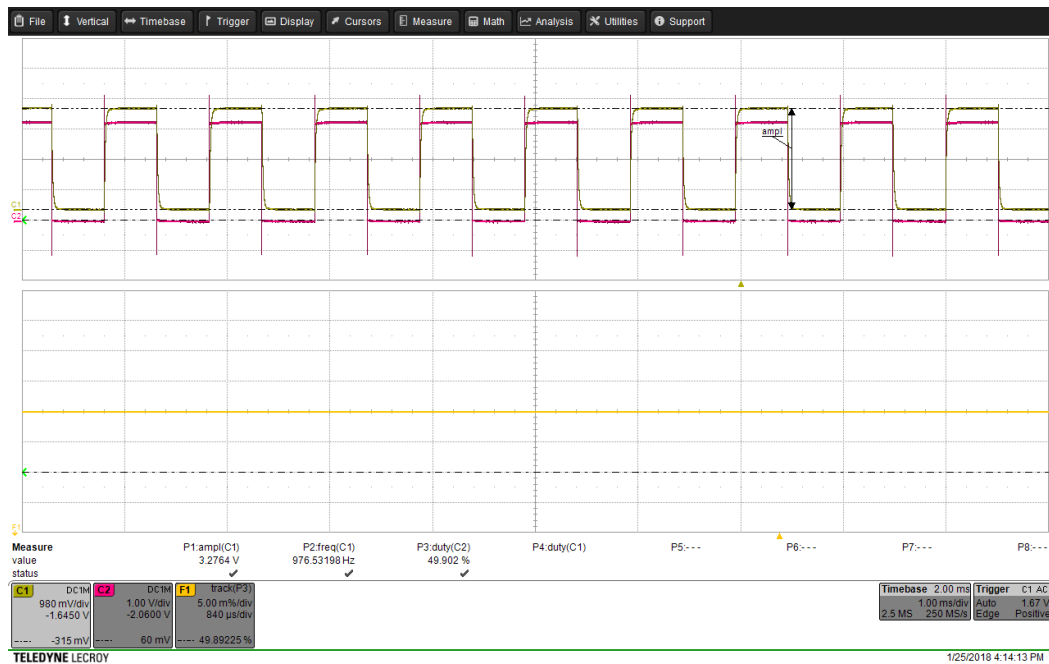


Abbildung 1.9: Rechteck Verifikation  
Quelle: eigene Ausarbeitung