FH Vorarlberg
University of Applied Sciences

Assignment 04
DAC

Embedded Systems 3

Fachhochschule Vorarlberg

Master Mechatronics

Eingereicht bei

Dr. Andrè Mitterbacher

Vorgelegt von

Roman Passler

Dornbirn, 24.01.2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 DAC

1.1 Einleitung

Bei dieser Aufgabe soll ein Digital Analogue Converter (DAC) in Simulink erstellt und simuliert werden. Anschließend soll eine Verilog Codegenerierung gemacht werden. Die Verilog Codegenerierung benötigt „fixed-point“ Datentypen, somit müssen die „double“ Datentypen richtig deklariert werden. Dabei muss beachtet werden, dass bei Summen ein Überlauf stattfinden kann und somit das Ergebnis ein Bit länger sein muss wie die Eingänge.

1.2 Implementierung

Ein Delta-Sigma-Wandler wandelt einen digitalen Wert in einen Bitstrom um. Der Durchschnittswert des Bitstroms ist der digitale Eingabewert. Ein analoger Tiefpassfilter (Rekonstruktionsfilter) wird dann verwendet, um den Durchschnitt des Bitstroms zu erhalten. Dadurch wird der digitale Wert in ein analoges Signal (z. B. Spannung) umgewandelt.

In Abbildung 1.1 ist die Implementierung ersichtlich, bestehend aus:

Signalgenerierung: Vorgabe einer DC oder Sinus Spannung.

Diskretisierung mit Halteglied: Hier wird das generierte Signal auf ein „digitales“ Signal gewandelt.

„ds_dac_dbl“: Hier wird der der Sigma Delta Wandler mit „double“ Datentypen simuliert (siehe Abbildung 1.3).

„ds_dac_sl“: Hier wird der Sigma Delta Wandler mit „fixed-point“ Datentypen simuliert und ist somit für die Verilog Codegenerierung geeignet (siehe Abbildung 1.4).

„Pulse Width Modulation (PWM)“: Ist ein Vergleich, wie mit einer PWM ein „analoges“ Signal generiert werden kann.

DS_DAC_Trans_XXX: Hier wird ein Tiefpassfilter auf das generierte Pulssignal angewendet (siehe Abbildung 1.2).

Die Samplezeit wurde auf $\frac{1}{50 \cdot 10^6}$ s gestellt und ein Fixed-step Solver ausgewählt. τ wurde $0.01 \cdot 10^{-3}$ gewählt. Die Frequenz ist mit 100 Hz definiert.

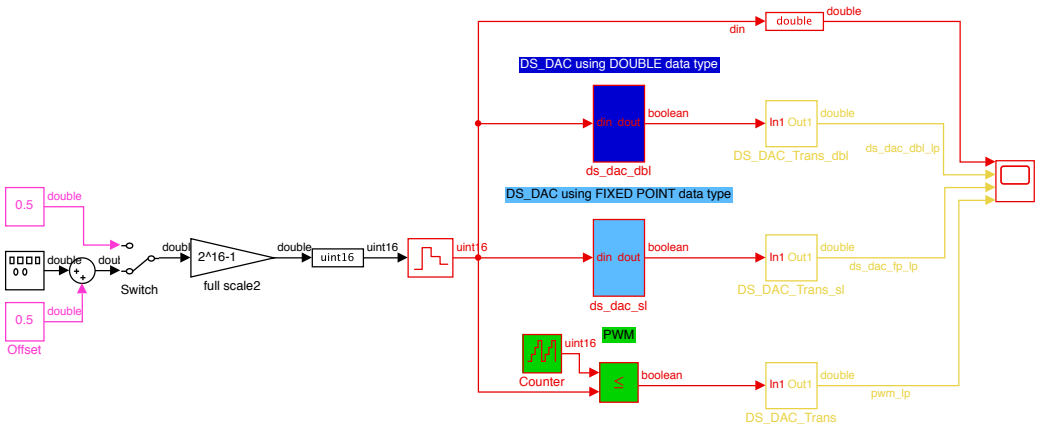

Abbildung 1.1: Simulink Implementierung
Quelle: eigene Ausarbeitung

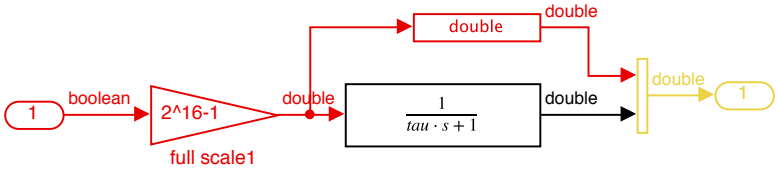

Abbildung 1.2: Simulink Implementierung: Tiefpassfilter
Quelle: eigene Ausarbeitung

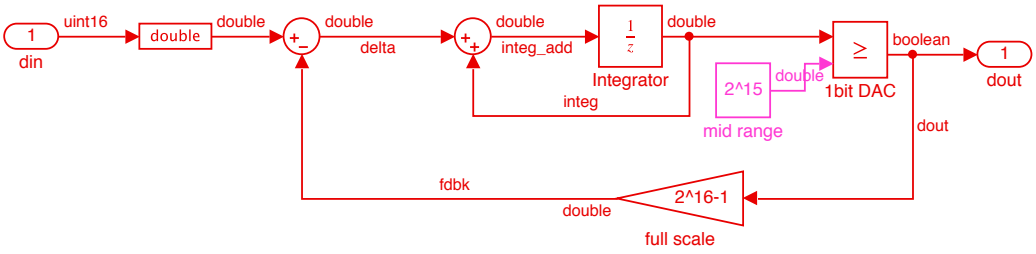

Abbildung 1.3: Simulink Implementierung: DAC mit „fixed-point“ Datentypen
Quelle: eigene Ausarbeitung

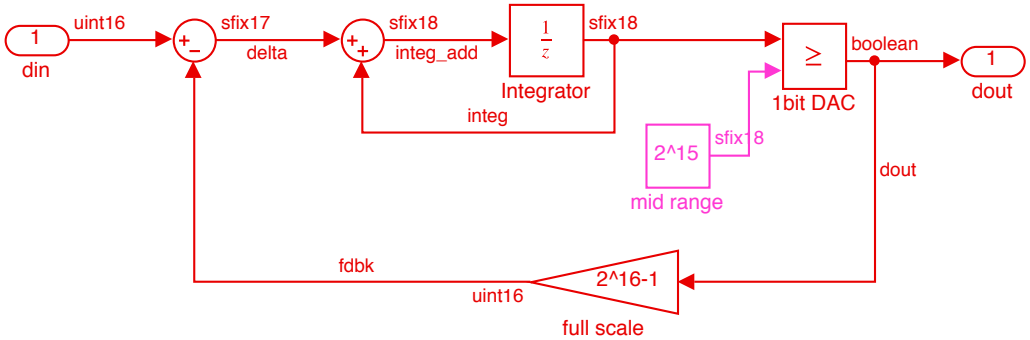

Abbildung 1.4: Simulink Implementierung: DAC mit „double“ Datentypen
Quelle: eigene Ausarbeitung

Bei der Verilog Codegenerierung ist ersichtlich, dass die Deklaration von Variablen noch genauer angeben ist. Ebenfalls muss die Dateiendung von „*.s“ auf „*.sv“ geändert werden.

```
1  // -------------------------------------------------------------
2  //
3  // File Name: hdlsrc/deltasigma_code_gen/ds_dac_sl.v
4  // Created: 2018-01-12 13:19:31
5  //
6  // Generated by MATLAB 9.3 and HDL Coder 3.11
7  //
8  //
9  // -- -------------------------------------------------------------
10 // -- Rate and Clocking Details
11 // -- -------------------------------------------------------------
12 // Model base rate: 2e-08
13 // Target subsystem base rate: 2e-08
14 //
15 //
16 // Clock Enable  Sample Time
17 // -- -------------------------------------------------------------
18 // ce_out        2e-08
19 // -- -------------------------------------------------------------
20 //
21 //
22 // Output Signal                 Clock Enable  Sample Time
23 // -- -------------------------------------------------------------
24 // dout                          ce_out        2e-08
25 // -- -------------------------------------------------------------
26 //
27 // -------------------------------------------------------------
28
29
30 // -------------------------------------------------------------
31 //
32 // Module: ds_dac_sl
33 // Source Path: deltasigma_code_gen/ds_dac_sl
34 // Hierarchy Level: 0
35 //
36 // -------------------------------------------------------------
37
38 `timescale 1 ns / 1 ns
39
40 module ds_dac_sl
41           (clk50m,
42            rst_n,
43            clk_enable,
44            din,
45            ce_out,
46            dout);
47
48
49   input    clk50m;
50   input    rst_n;
51   input    clk_enable;
52   input    [15:0] din;  // uint16
53   output   ce_out;
54   output   dout;
55
56   wire enb;
57   wire signed [17:0] mid_range_out1;  // sfix18
```

```
58  wire alpha1bit_DAC_relop1;
59  wire dout_1;
60  wire [15:0] fdbk;  // uint16
61  wire signed [31:0] Sum2_sub_temp;  // sfix32
62  wire signed [31:0] Sum2_1;  // sfix32
63  wire signed [31:0] Sum2_2;  // sfix32
64  wire signed [16:0] delta;  // sfix17
65  reg signed [17:0] integ;  // sfix18
66  wire signed [31:0] Sum1_add_temp;  // sfix32
67  wire signed [31:0] Sum1_1;  // sfix32
68  wire signed [31:0] Sum1_2;  // sfix32
69  wire signed [17:0] integ_add;  // sfix18
70
71
72  assign enb = clk_enable;
73
74  assign mid_range_out1 = 18'sb001000000000000000;
75
76
77
78  assign dout_1 = alpha1bit_DAC_relop1;
79
80  assign fdbk = (dout_1 == 1'b1 ? 16'b1111111111111111 :
81                 16'b0000000000000000);
82
83
84
85  assign Sum2_1 = {16'b0, din};
86  assign Sum2_2 = {16'b0, fdbk};
87  assign Sum2_sub_temp = Sum2_1 - Sum2_2;
88  assign delta = Sum2_sub_temp[16:0];
89
90
91
92  assign Sum1_1 = {{15{delta[16]}}, delta};
93  assign Sum1_2 = {{14{integ[17]}}, integ};
94  assign Sum1_add_temp = Sum1_1 + Sum1_2;
95  assign integ_add = Sum1_add_temp[17:0];
96
97
98
99  always @(posedge clk50m or negedge rst_n)
100   begin : Integrator_process
101     if (rst_n == 1'b0) begin
102       integ <= 18'sb000000000000000000;
103     end
104     else begin
105       if (enb) begin
106         integ <= integ_add;
107       end
108     end
109   end
110
111
112
113 assign alpha1bit_DAC_relop1 = integ >= mid_range_out1;
114
115
116
117 assign dout = alpha1bit_DAC_relop1;
118
```

```
119   assign ce_out = clk_enable;
120
121 endmodule  // ds_dac_sl
```

Listing 1.1: Resultat der Codegenerierung

1.3 Test Bench

In Listing 1.2 ist die Testbench ersichtlich. Das Herzstück der Testbench ist die Sinusgenerierung (Codezeile 68). Diese Anweisung verpackt in einer „while“ Schleife ergibt einen Sinus. Die Frequenz, Amplitude und Startpunkt der Sinusgenerierung wurde zur besseren Vergleichbarkeit an die Simulation in Simulink angepasst.

```
1  /*-----------------------------------
2  Project:     ds_dac_sl
3  Purpose:
4  Author:      rpa2306
5  Version:     00, 01.12.2018
6  -----------------------------------*/
7
8  `timescale 10ns/10ns
9
10
11 module tb_ds_dac_sl();
12
13 /*
14 *.  (1) Create wires to connect the DUT
15 *.  Like footprints for an IC on a PCB
16 */
17
18 logic    clk50m;
19 logic    rst_n;
20 logic    clk_enable;
21 logic    [15:0] din;
22 logic    ce_out;
23 logic    dout;
24
25 /*
26 *    (2) Create an instance of the DUT
27 */
28
29 ds_dac_sl                          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     automatic int cntSoftware = 0;
48     automatic int offset = 2**16/2-1;
49     automatic int sinus = 0;
50     automatic int endTime = 0;
51     $display("------------------------------");
52     $display("    ds_dac_sl started       ");
53     $display("------------------------------");
54
55     @ (negedge clk50m);
56     rst_n = 1'b0;
57     clk_enable = 1'b0;
58     din = 16'b0;
59     #50ns;
60     @ (negedge clk50m);
61     rst_n = 1'b1;
62     clk_enable = 1'b1;
63     #50ns;
64
65     endTime = $realtime + 1000000;
66     while ($realtime < endTime) begin
67         @ (negedge clk50m); // wait for negedge
68         sinus = $sin(2*3.14*$realtime*0.000001)*offset+offset; //
69         // check if sin is negativ, because din is a unsigned fixed
               point number
70         if (sinus < 0) begin
71             sinus = 0;
72         end
73             //$display("%d",$realtime);
74         din = sinus;
75     end
76
77     @ (negedge clk50m);
78     run_sim = 1'b0;
79     $display("------------------------------");
80     $display("   ds_dac_sl finished       ");
81     $display("------------------------------");
82 end
83 endmodule
```

Listing 1.2: Testbench für den DAC

1.4 Simulationsscript

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

```
1  # Create simulation environment
2  vlib work
3  vmap work work
4
5  # Compile desing files -> use file names
6  vlog ../src/ds_dac_sl.sv
7  # Compile the test bench
8  vlog tb_ds_dac_sl.sv
9  # Init simulation -> use module name
10 vsim tb_ds_dac_sl
```

```
# -r recursive
log -r *
do wave_tb_ds_dac_sl.tcl

# Run simulation
run -all
# run 100us
# Show results
view wave
```

Listing 1.3: Simulationsscript

1.5 Waveform Window und Simulink Simulation

In Abbildung 1.5 ist das Waveform Window dargestellt. Es zeigt den generierten Sinus. In Abbildung 1.6 ist der Vergleich zur Simulation ersichtlich. Es ist ersichtlich, dass bei den „Peaks“ (ca. bei 2.5 ms) der Ausgang „High“ und bei den „Lows“ (ca. bei 7.5 ms) der Ausgang „Low“ ist.

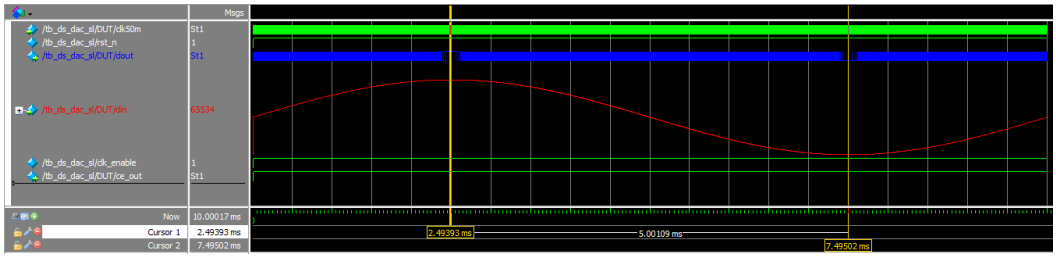

Abbildung 1.5: Waveform Window
Quelle: eigene Ausarbeitung

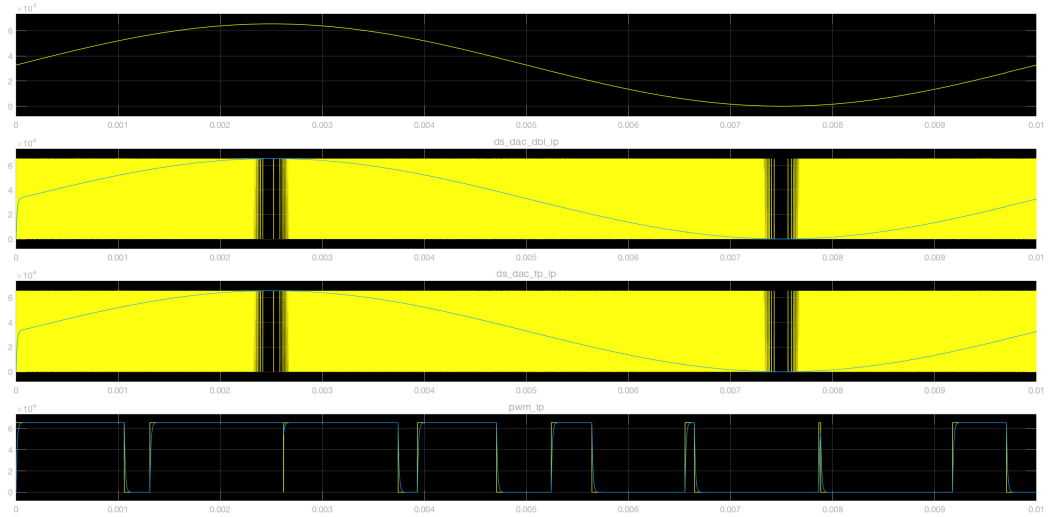

Abbildung 1.6: Simulink Simulationsergebnisse
Quelle: eigene Ausarbeitung