Methodischer Entwurf und Simulation eines Δ-Σ A/D Wandlers mit VHDL-AMS

Steffen Klupsch, Sorin A. Huss

{klupsch|huss}@iss.tu-darmstadt.de
Integrierte Schaltungen und Systeme
TU Darmstadt, FB Informatik
Alexanderstr. 10, 64283 Darmstadt

Zusammenfassung

Der folgende Beitrag stellt exemplarisch die Verwendung von VHDL-AMS für den Entwurf und die simulative Validierung von integrierten Mixed-Signal Schaltungen vor. Die zu implementierende Komponente ist ein Delta-Sigma Analog-/Digital-Wandler. VHDL-AMS ermöglicht eine Gesamtsystemsimulation des Wandlers, sowie einen hierarchischen Modellierungsansatz, der die methodische Weiterentwicklung der enthaltenen Teilkomponenten ermöglicht. Es wird gezeigt, daß VHDL-AMS sowohl eine effiziente Simulation von konzeptionellen 'High-Level' Modellen, als auch eine genaue Simulation von hardwarenah beschriebenen Entwurfsobjekten ermöglicht.

1 Einleitung

VHDL-AMS [4] ist eine Weiterentwicklung des IEEE Standards 1076 von 1993. Diese neue Simulationssprache beinhaltet das aus dem Entwurf digitaler Schaltungen bekannte VHDL und ergänzt es mit Beschreibungsmöglichkeiten für zeitkontinuierliche Modelle. Dadurch wird es möglich 3 Modellklassen mit verschiedenen Ausführungsparadigmen gemeinsam zu simulieren: Sequentielle zeitfreie Algorithmen mit ereignisdiskreter Kommunikationsschale, nebenläufige ereignisdiskrete Verhaltensbeschreibungen und zeitkontinuierliche Modelle mit bidirektionalem Signalfluß. Die durch implizite differential-algebraische Gleichungssysteme beschriebenen Modelle in der zuletzt genannten Klasse eigenen sich z. B. zur Beschreibung von analogen Schaltungen. Zur Beschreibung von synthetisierbaren digitalen Schaltungen sind die über explizite Zuweisungen (bzw. Bus-Resolution Functions) beschriebenen Modelle über den ereignisdiskreten Zustandsgrößen sinnvoll und mit Hilfe von ‚zeitlosen', lediglich sequentiell geordneten kausalen Algorithmen können in *Subprograms* komplexe Berechnungen effizient durchgeführt werden [5]. Zur Unterscheidung der Modellklassen werden im folgenden die Begriffe α-Klasse, β-Klasse und γ-Klasse verwendet. Ein über *zeitfreie* Variablen definiertes Verhaltensmodell ist ein Element der α-Klasse, die mit Hilfe von Signalen modellierten ereignisdiskreten Modelle sind Elemente der β-Klasse und Modelle mit ungerichteten zeit- und wert-kontinuierlichen Zustandsgrößen sind aus der γ-Klasse. Mischmodelle, in denen Elemente verschiedener Klassen kombiniert werden, sind in VHDL-AMS möglich und oft sinnvoll, da sie eine Zusammenfas-

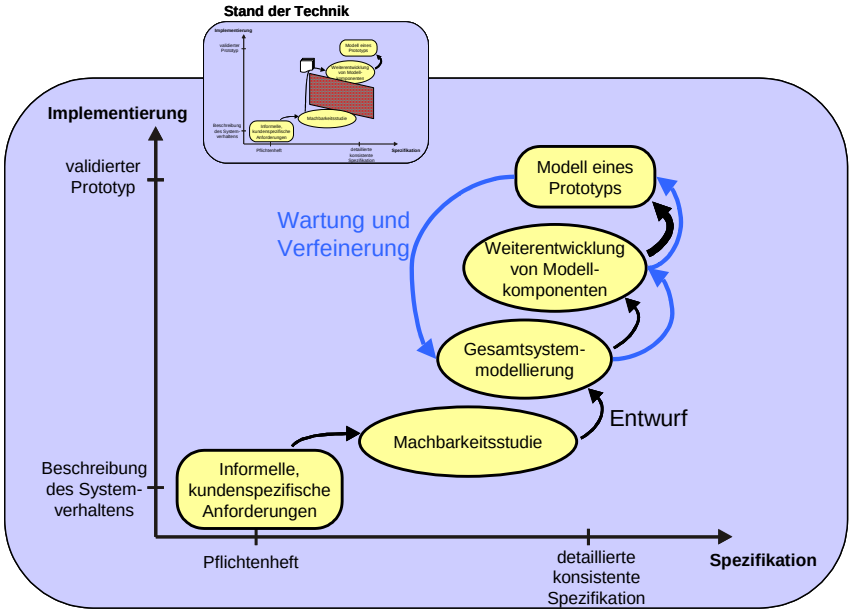

Abbildung 1: Entwurfsablauf, Designschritte und deren Nutzen im Hinblick auf Produktspezifikation und Implementierung

sung des Modellverhaltens in Funktionsgruppen ermöglichen und somit die Modellpflege erleichtern.[1]

Diese Ausdrucksmächtigkeit ermöglicht einen Entwicklungsablauf, der von der Konzeptionsphase bis zur Implementierungsphase auf VHDL-AMS Modellen basiert. Abb. 1 zeigt den im folgenden verwendeten Entwurfsablauf [3]. Aus den Ergebnissen einer Machbarkeitsstudie werden Gesamtsystemmodelle erzeugt, die für eine experimentelle Validierung verwendet werden. Dementsprechend sind die Anforderungen an ein Gesamtsystemmodell: Es muß effizient simulierbar sein, die enthaltenen Komponentenmodelle müssen leicht verständlich sein und das durch die Gesamtsystemsimulation nachgebildete Systemverhalten muß eine aussagekräftige Validierung ermöglichen.

In den folgenden Implementierungsschritten werden einzelne Komponenten des Gesamtsystemmodells weiterentwickelt und genauer beschrieben. Dadurch entstehen Modelle, deren Verhalten besser beschrieben ist, deren Simulation jedoch mehr Rechenzeit in Anspruch nimmt. Dementsprechend ist es in der Regel ineffizient eine Simulation des Gesamtsystems mit vielen detailliert beschriebenen Komponenten durchzuführen. Stattdessen werden *kalibrierte High-Level Modelle* benötigt, die das Verhalten einer Komponenten simulationseffizient nachbilden. Diese Pflege der Gesamtsystemmodelle ist ein zusätzlicher und unverzichtbarer Bestandteil des verwendeten Entwurfsablaufs.

[1] Für die Kommunikation zwischen Modellen der α-Klasse werden ereignisdiskrete Signale benötigt, so daß zwangsläufig Elemente der β-Klasse enthalten sind. Wenn in einem Modell Signale ausschließlich zu Kommunikationszwecken eingesetzt werden, so wird es im folgenden als Modell der α-Klasse bezeichnet.

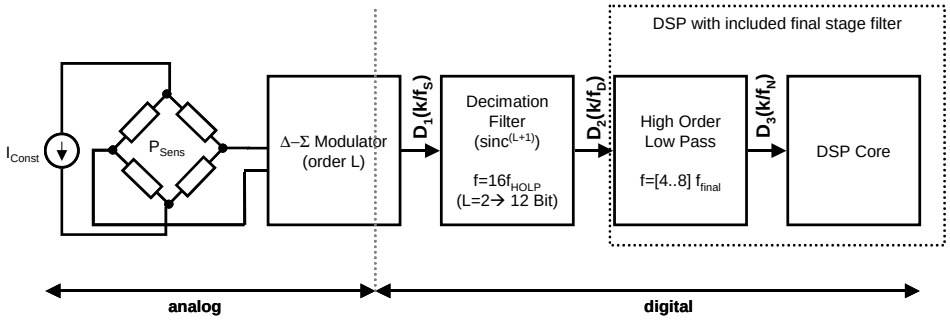

Abb. 2: Generische Architektur eines Δ-Σ A/D Wandlers

In größeren Projekten, die von mehreren Arbeitsgruppen bearbeitet werden, ist die Integration der Teilkomponenten zu einem Gesamtsystem eine zusätzliche Fehlerquelle, deren Signifikanz nur durch die frühzeitige Gesamtsystemmodellierung an Bedeutung verlieren kann. Trotzdem ist es sinnvoll ausführliche Testbenches zu erarbeiten, die eine automatisierte Validierung ermöglichen. Zu diesem Zweck können Komponentenmodelle aus der Gesamtsystemodellierung als *Referenz* verwendet werden, so daß sich der Aufwand für die Gesamtsystemmodellierung relativiert.

Als Anwendungsbeispiel dient der Entwurf und die Validierung eines Delta-Sigma A/D Wandlers. Δ-Σ A/D Wandler sind in den letzten 20 Jahren sehr beliebt geworden, da bei diesen Architekturen durch digitale Nachbearbeitung Ungenauigkeiten im analogen Teil ausgeglichen werden können. Das ermöglicht eine industrielle Produktion mit guter Ausbeute in günstigen IC-Prozessen. So werden heutzutage Δ-Σ A/D Wandler kostengünstig mit DSP Cores kombiniert - in IC-Prozessen, die für die meisten A/D Wandler Architekturen ungeeignet sind.

Ein Δ-Σ A/D Wandler besteht im allgemeinen aus drei Blöcken: dem Delta-Sigma Modulator, einem schnellen Decimation-Filter und einem nachgeschalteten weiteren Filter hoher Ordnung (s. Abb. 2). Im allgemeinen ändert sich in jeder Stufe des Wandlers die Datenrate und die Breite der Datenworte. Im DSP Core werden die Daten entsprechend der enthaltenen Nutzsignalfrequenzen und des Nyquist-Theorems mit einer Datenrate von f_N verarbeitet. Der vorgeschaltete Tiefpaß erzeugt diese Daten i. d. R. aus einem 4 bis 8 mal schnelleren Signal. Dieser Tiefpaß kann auch aus kaskadierten Halbbandfiltern bestehen, die jeweils ein Downsampling um Faktor 2 zur Folge haben. Um den Platzbedarf klein zu halten, kann der Tiefpaß mit einer bitseriellen Architektur implementiert werden, wodurch sich die interne Taktrate aus der Eingangsdatenrate multipliziert mit der Wortbreite ergibt.

Vor dem Tiefpaßfilter ist ein schneller und platzsparender Decimation Filter, der einen hohen Downsampling-Faktor bei geringer Rauschsignal-Einkoppelung ermöglichen muß. Im Decimation-Filter wird das Ausgangssignal des Modulators i. d. R. zwischen 8- und 64-fach reduziert. Daraus ergibt sich ein Oversampling-Faktor für den Delta-Sigma Modulator,

```
function Convert_AnalogToDigital
  ( U_min : real; U_max : real;
    BitRes : integer; U_in : real)
  return std_logic_vector is

  constant i_max : integer := (2**BitRes)-1;
  variable r : real;
  variable i : integer;
  variable u : unsigned(BitRes-1 downto 0);

begin
  r := (U_in - U_min) / (U_max - U_min);
  i := integer(r * (2.0**BitRes));
  if (i > i_max) then
    i := i_max;
  end if;
  u := conv_unsigned(i, u'length);
  return std_logic_vector(u);
end Convert_AnalogToDigital;
```

Listing 1: Algorithmisches Modell eines idealen A/D Wandlers

der zwischen 32 und 512 liegt. Die analogen Komponenten des Modulators müssen passend zur benötigten Samplingrate dimensioniert werden. So wird beispielsweise für einen Audio-A/D Wandler, der ein Signalspektrum bis 48 kHz erfassen soll, bei 128-fach Oversampling eine Samplingrate (f_S) von etwa 6 MHz benötigt.

2 Modellierung des idealen A/D Wandlers

Um den A/D Wandler zu validieren, benötigt man eine Referenzimplementierung, die beispielsweise aus einem idealen A/D Wandler bestehen kann. Der ideale A/D Wandler hat folgende Eigenschaften:

- Lineares Verhalten, d. h. es treten keine Konvertierungsfehler im digitalen Ergebnis auf.
- Keine Modellierung des Zeitverhaltens, d. h. das Modell kann nur eingesetzt werden, wenn die Konvertierungszeit vernachlässigbar ist.
- Keine interne Zustände, somit werden Korrelationen zwischen zeitlich benachbarten Konvertierungsergebnissen vernachlässigt.
- Stets betriebsbereit, d. h. das Modell hat keine Initialisierungsphasen, es liefert auf jede Anfrage ein gültiges Ergebnis.

Zur Modellierung eines solchen A/D Wandlers bietet sich die α-Klasse an. Listing 1 zeigt eine Implementierung als VHDL-Funktion. Ist dieses Modell gut? Es modelliert keine Einschwingvorgänge, es ist nicht geeignet, um einen schnellen Flash-A/D Wandler von

einem langsamen Δ-Σ A/D Wandler zu unterscheiden, es gibt keine Möglichkeit Nichtlinearitäten eines A/D Wandlers zu berücksichtigen. Darüber hinaus ist es im Vergleich zu einem physikalischen Testaufbau zu ungenau - sowohl im zeitlichen Verhalten, als auch im quasistationären Betriebsfall. Phänomene wie Oszillation zwischen zwei digitalen Werten können durch Rauschen auf dem Eingangssignal erzeugt werden, es ist jedoch nicht möglich eine Hysterese einzustellen.

Das Modell ist trotzdem sinnvoll: Es ist klein, überschaubar und verständlich. Es ist simulationseffizient, da es rückkopplungsfrei ist und nur wenige Zuweisungen evaluiert werden müssen. Es ist trivial und beinhaltet wenig Fehlerquellen - es eignet sich somit zur automatischen Validierung, da es mit wenig Aufwand verbunden ist, im Fehlerfall den Defekt auf das Device-Under-Test (DUT) zurückzuführen. Das Modell ist universell einsetzbar, es ist für beliebige Bitbreiten konfigurierbar und modelliert das quasistationäre Verhalten beliebiger A/D Wandler. Es kann zur Stimulierung digitaler Schaltungen eingesetzt werden, ohne die Digitalsimulation durch einen Analogsolver zu bremsen.

3 Modellierung eines Δ-Σ A/D Wandlers

Ein Δ-Σ A/D Wandler hat aufgrund von Noise-Shaping und überlagertem Pattern-Noise keine weißes Rauschspektrum. Die integrierten digitalen Tiefpäße erzeugen eine frequenzabhängige Rauschdichte und der analoge Modulator hat zusätzliche mit dem Eingangssignal korrelierte *Töne* [6]. Zur Validierung der digitalen Filter wird ein Modell des Modulators benötigt, das bei der Simulation wenig Rechenzeit benötigt und trotzdem die zeit- und frequenzabhängigen Eigenschaften des Modulators nachbildet. Da die Lösung von differentialalgebraischen Gleichungssystemen wesentlich aufwendiger ist als die Simulation eines ereignisdiskreten Modells, stellt sich die Frage, ob aus den Ergebnissen der Machbarkeitsstudie ein brauchbares β-Klassen-Modell des Modulators extrahiert werden kann. Für einen A/D Wandler mit etwa 75 dB SNR und einer Auflösung von 12-Bit bietet sich ein Delta-Sigma Modulator 2. Ordnung an. Die Architektur des gewählten Delta-Sigma Modulator ist in Abb. 3 gezeigt. Sie ist aus dem in [1] vorgestellten Modulator abgeleitet.

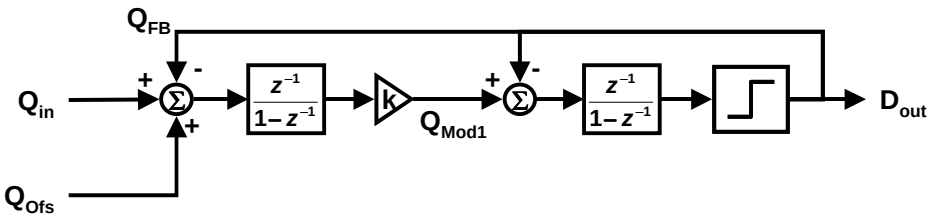

Abb. 3: Architektur des Delta-Sigma Modulators 2. Ordnung

In den meisten Literaturquellen werden zur Berechnung der Modulator-Übertragungsfunktion Spannungspegel verwendet. Da die Definition des Feedbacksignals und der Eingangssignale mit Hilfe von Ladungen jedoch eine flexiblere Spezifikation ermöglicht, wurde in Abb. 3 Ladungen als Zustandsgrößen verwendet.

Ein 1-Bit Δ-Σ Modulator 2. Ordnung kann übersteuert werden, d. h. an einer Stelle im Modulator kann keine zusätzliche Ladung auf einem Kondensator gespeichert werden, obwohl dies für die korrekte Funktion des Modulators notwendig wäre (Integrator Overload Noise). Der Fehler entsteht, weil die enthaltenen Operationsverstärker einen begrenzten Ausgangsspannungsbereich haben. Mit steigender Frequenz des Eingangssignals werden diese Fehler wahrscheinlicher. Um Fehler, bzw. Rauschen durch Überladung des Modulators zu verringern, muß man den Eingangsspannungsbereich reduzieren. (Bei einer Erhöhung des Oversampling-Ratio wird die Auswirkung dieses Fehlers ebenfalls gedämpft.)

Für die Modulator Architektur aus Abb. 3 wurde für $k = \frac{1}{2}$ über Wahrscheinlichkeitsrechnungen die in Gl. (2) verwendete Definition begründet. Sie ist aus [1] hergeleitet, indem in den ebenda angegebenen Spannungsgleichungen die Kapazitätsverhältnisse eingesetzt wurden.

Man berechnet die Ladung $\max(Q_{int})$ in den Feedbackkondensatoren aus

$$\max(Q_{int}) = C_{int}\left(\max(U_{OP,out}) - \min(U_{OP,out})\right) \tag{1}$$

$$\Delta := \frac{\max(Q_{int})}{3.4} \tag{2}$$

Als Feedbacksignal wird $\pm\Delta/2$ verwendet, für die abgetasteten Eingangssignale gilt:

$$-0.3\Delta \leq Q_{in} \leq 0.3\Delta \tag{3}$$

Für ein differentielles Eingangssignal aus dem Bereich $[0; 360\,mV]$ und $\pm 1\,V$ Konstantspannungsquellen für die Bildung der Offset- und Feedbackladungen, können die benötigten Sample-Kondensatoren berechnet werden:

$$C_{in} = \frac{25}{17}C_{Int} \qquad C_{Ofs} = \frac{9}{34}C_{Int} \tag{4}$$

$$C_{FB} = \frac{15}{34}C_{Int} \qquad C_{Mod1} = \frac{1}{2}C_{Int} \tag{5}$$

Mit diesen Daten kann ein einfaches α-Klassen Modell in VHDL-AMS notiert werden (siehe Listing 2), welches das Noise-Shaping des Modulators inklusive der *Töne* nachbildet. Ein *Overload*-Fehler kann durch Überwachung der Variablen Sigma1 und Sigma2 detektiert werden und durch Variation der Modellparameter kann das Modell für unterschiedliche Worst-Case Simulationen verwendet werden.

Modelle der α-Klasse sind in vielen Designflows sogenannte *Metamodelle* - ein Modell eines Modells. Bei der Entwicklung des Δ-Σ A/D Wandlers ist das ebenso und die anderen Entwurfsebenen spielen eine signifikante Rolle. Für die Weiterentwicklung des analogen

```
library IEEE;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
library disciplines;
  use disciplines.electromagnetic_system.all;

entity delta_sigma_modulator is
  generic (
    VRefP   : emf; -- 3.5V
    VRefN   : emf; -- 1.5V
    C_FB    : real := 15.0/34.0; -- C_FB / C_Integrator
    C_Sample : real := 25.0/17.0; -- C_sample / C_Integrator
    C_Offset : real := 9.0/34.0; -- C_Offset / C_Integrator
    C_Mod1  : real := 0.5);    -- C_Mod1 / C_Integrator
    port (terminal In_P : Electrical;
          terminal In_M : Electrical;
          signal Clock, Reset : in std_logic;
          signal DsmRes : out std_logic);
end delta_sigma_modulator;

architecture second_order_sample_and_hold of delta_sigma_modulator is
  quantity u_in across In_P to In_M;
begin
  DSM: process (reset, clock)
    constant Q_Feedback : charge := (VRefP-VRefN) * C_FB;
    constant Q_Offset : charge := (VRefP-VRefN) * C_Offset;
    variable delta1, delta2, sigma1, sigma2, feedback : real := 0.0;
    variable res : std_logic;
  begin
    if Reset = '1' then
      sigma1 := 0.0;
      sigma2 := 0.0;
      feedback := Q_Feedback;
      DsmRes <= '0';
    end if;
    if clock'event and clock = '1' then
      delta1 := u_in * (2.0 * C_sample) - Q_Offset + Feedback;
      delta2 := sigma1 * C_Mod1 + Feedback;
      sigma1 := sigma1 + delta1;
      sigma2 := sigma2 + delta2;
      if (sigma2 > 0.0) then
        res <= '1';
        feedback := -Q_Feedback;
      else
        res <= '0';
        feedback := Q_Feedback;
      end if;
      DsmRes <= res;
    end if;
  end process;
end second_order_sample_and_hold;
```

Listing 2: α-Klassen Modell eines Δ-Σ Modulators

```
component Eldo_ro3 is
  port (
    terminal VDDA, AGND : electrical;
    terminal nRst : electrical;
    terminal VRefP, VRefN, VcmOP, VcmOut : electrical;
    terminal IRefP : electrical;
    terminal SCinP, SCinN, SCoutP, SCoutN, FB, nFB : electrical;
    terminal Clk1, nClk1, Clk1D, nClk1D, Clk2, nClk2, Clk2D, nClk2D : electrical);
end component Eldo_ro3;
attribute eldo_device of Eldo_ro3: component is Eldo_subckt;
attribute eldo_subckt_name of Eldo_ro3: component is "ro3__1";
attribute eldo_file_name of Eldo_ro3: component is "ckt/ro3.cir";
```

Listing 3: VHDL-AMS Interface zu einer SPICE-Netzliste

Modulators ist eine Modellierung auf γ-Klasse notwendig, die letztendlich in einer Strukturbeschreibung von analogen Grundelementen (Transistoren, Kapazitäten, etc.) endet. Ob für diese Grundelemente Verhaltensmodelle verwendet werden, für die eine Notation in VHDL-AMS vorliegt, oder statt dessen Verhaltensmodelle aus einer Bibliothek des Simulatorherstellers verwendet werden, ist letztendlich nebensächlich. Eine Möglichkeit zur Einbindung von externen Bibliotheken, wie sie Advance MS von Mentor Graphics bietet, erleichtert jedoch den VHDL-AMS basierten Validierungsprozeß: Für das Layout wurde eine SPICE Netzliste der analogen Schaltungen benötigt, deren Verhalten mit den VHDL-AMS Modellen validiert wurde, indem die Netzlisten als *Components* in eine VHDL-AMS Testbench integriert wurden (siehe Listing 3).

3.1 Validierungsmodelle für digitale Filter

An den digitalen Ausgang des Modulators schließt sich der Decimation Filter an. Die Aufgabe dieses Filters ist es aus dem n-fach überabgetasteten, grob quantisierten Datenstrom durch eine geeignete mathematische Funktion (oft eine sinc^k Funktion) eine lediglich 4-8 fach überabgetastete, feiner quantisierte Datenfolge zu generieren. Da der maximale Arbeitstakt dieser Baugruppe die mögliche Oversamplingrate begrenzt, werden Hardware effiziente Implementierungen wie die in Abb. 4 gezeigte Modulo-n Decimation bevorzugt [2]. Für diesen Filter wurde ein simulationseffizientes Verhaltensmodell geschrieben, in dem der Decimation-Ratio und die Anzahl der Filterstufen über Parameter bestimmt werden können (siehe Listing 4).

Der nachgeschaltete Filter ist ein FIR-Filter, dessen VHDL-AMS Modell vollautomatisch generiert wird (siehe Listing 5). Dazu werden lediglich der Sperr- und der Durchlaßbereich, sowie die gewünschte Rauschsignalunterdrückung in einem GnuPlot-Script angegeben. Das generierte Verhaltensmodell realisiert einen Kaiser-Filter [7]. Abb. 5 zeigt den für diese Modellen berechneten Frequenzgang des Δ-Σ A/D Wandlers.

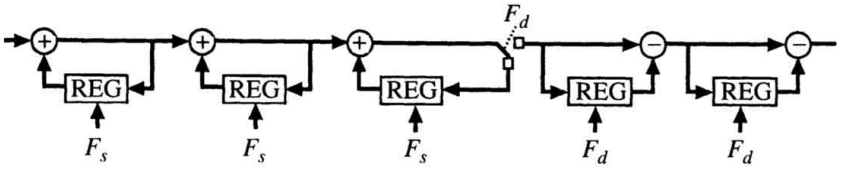

Abb. 4: Funktionsprinzip der Modulo Decimation Stufe

4 Zusammenfassung

VHDL-AMS wurde beim Entwurf eines Δ-Σ A/D Wandlers zur Modellierung eines Gesamtsystems eingesetzt. Bei der Entwicklung der Teilkomponenten des A/D Wandlers können diese Modelle zur Validierung eingesetzt werden. Dazu ist es notwendig den Entwurfsprozeß so auszulegen, daß eine gemeinsame Simulation der Gesamtsystemmodelle mit den konkretisierten Komponentenmodellen möglich wird. Für den Entwurf digitaler Teilschaltungen sind durch VHDL-basierte Synthesewerkzeuge dementsprechende Entwicklungsumgebungen verfügbar, bei der Implementierung analoger Schaltungsblöcke ist zur Zeit ein Transfer der VHDL-AMS Verhaltensmodelle in eine SPICE kompatible Netzliste ein oft unumgänglicher Arbeitsschritt. Dementsprechend wichtig ist eine gemeinsame Simulation von VHDL-AMS Modellen mit Strukturmodellen in SPICE. Der signifikante Vorteil der Modellierung mit VHDL-AMS liegt in den abstrakten Beschreibungsmöglichkeiten, die weit über die Möglichkeiten von SPICE hinausgeht. Bei der Implementierung analoger Blöcke wandelt sich das Modell von einem Verhaltensmodell zu einem Strukturmodell, dessen Grundkomponenten im analogen Chipdesign zur Verfügung stehen müssen. Ein solches Strukturmodell kann oft in einer SPICE Netzliste effektiver abgebildet werden, als mit VHDL-AMS Komponenteninstanzierungen.

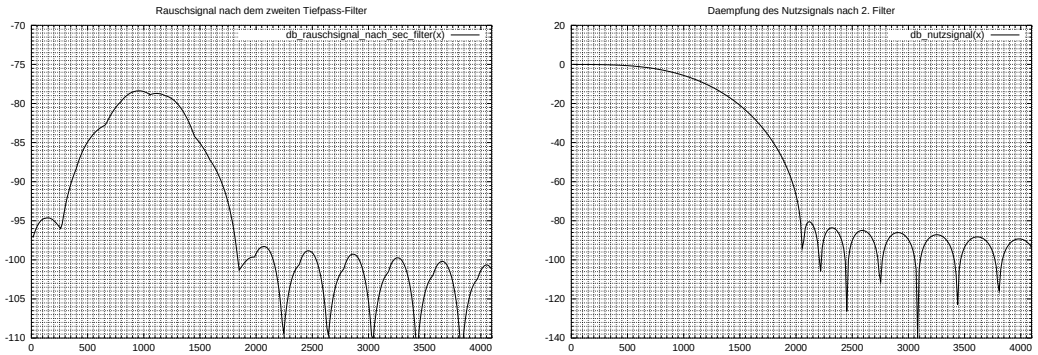

Abbildung 5: Frequenzgang des Δ-Σ A/D Wandlers

```
library IEEE;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_arith.all;

entity sin_k_filter is
 generic (
  sin_k_stages : integer := 3; -- BitsPerSample must be equal to
  DecimationRatio : integer := 16; -- sin_k_stages * ld(DecimationRatio)
  BitsPerSample : integer := 12);
 port (
  signal Clock, Reset, Din : in std_logic;
  signal Dout : out std_logic_vector(BitsPerSample-1 downto 0);
  signal DoutValid : out std_logic);
end entity sin_k_filter;

architecture functional of sin_k_filter is
 type int_vector is array (natural range <>) of integer;
begin
 Main: process (Clock, reset) is
  constant Imax : integer := 2**(BitsPerSample+1);
  variable integ : int_vector(sin_k_stages-1 downto 0);
  variable diff : int_vector(sin_k_stages-1 downto 0);
  variable i, temp1, temp2, IterationsPerSample : integer;
  variable Dout_unsigned : unsigned(BitsPerSample downto 0);
 begin
  if reset = '1' then
   IterationsPerSample := 0;
   integ := (others => 0);
   diff := (others => 0);
  elsif Clock'event and Clock = '1' then
   if (Din='1') then integ(0) := (integ(0) + 1) mod Imax; end if;
   for i in 1 to integ'high loop
    integ(i) := (integ(i) + integ(i-1)) mod Imax;
   end loop; -- i
   IterationsPerSample := IterationsPerSample + 1;
   if IterationsPerSample = DecimationRatio then
    IterationsPerSample := 0;
    temp1 := diff(0);
    diff(0) := integ(integ'high);
    integ(integ'high) := 0;
    for i in 1 to diff'high loop
     temp2 := temp1; temp1 := diff(i);
     diff(i) := (diff(i-1) - temp2);
     if diff(i) < 0 then diff(i) := diff(i) + Imax; end if;
    end loop; -- i
    Dout_unsigned := conv_unsigned(diff(diff'high),BitsPerSample+1);
    if Dout_unsigned(Dout_unsigned'high) = '1' then
     Dout <= (others => '1');
    else
     Dout <= std_logic_vector(Dout_unsigned(Dout_unsigned'high-1 downto 0));
    end if;
    DoutValid <= '1';
   else DoutValid <= '0';
   end if;
  end if;
 end process Main;
end architecture functional;
```

Listing 4: VHDL Modell eines parametrisierbaren $\sin^k$-Filter

```
library IEEE;
 use ieee.math_real.all;

package kaiser_filter_coeff is
 constant kaiser_filter_order : integer := 81;
 constant kaiser_coef : real_vector(0 to 81) := (
                    2.09043252204112e-05,
                    4.0870757316356e-05,
                    6.63873273264839e-05,
                    9.51350785222413e-05,
                    [...]
                    2.09043252204112e-05);
end kaiser_filter_coeff;

package body kaiser_filter_coeff is
end kaiser_filter_coeff;
--------------------------------------------------------------------------------
library IEEE;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_arith.all;
 use work.kaiser_filter_coeff.all;

entity kaiser_filter is
 generic(BitsPerSample : integer := 12);
 port(
  signal Clock, Reset : in std_logic;
  signal Din : in std_logic_vector;
  signal Dout : out std_logic_vector(BitsPerSample-1 downto 0));
end entity kaiser_filter;

architecture functional of kaiser_filter is
begin -- architecture functional
 Main: process (Clock, reset)
  variable z_reg : real_vector(kaiser_filter_order downto 0);
  variable r1,r_in,r_out : real;
  variable i : integer;
 begin -- process Main
  if reset = '1' then        -- asynchronous reset (active high)
   Dout <= (others => '0');
   for i in z_reg'range loop
    z_reg(i) := 0.0;
   end loop;
  elsif Clock'event and Clock = '1' then -- rising clock edge
   r_in := real(conv_integer(unsigned(Din))) / (2.0**(Din'length-1));
   i := 1;
   r1 := kaiser_coef(0) * r_in;
   while (i <= kaiser_filter_order) loop
    r1 := r1 + kaiser_coef(i) * z_reg(i);
    z_reg(i) := z_reg(i-1);
    i := i+1;
   end loop;
   r_out := r1*(2.0**(BitsPerSample));
   Dout <= std_logic_vector(conv_unsigned(integer(r_out),BitsPerSample));
  end if;
 end process Main;
end architecture functional;
```

Listing 5: Automatisch erzeugter Kaiser-Filter

Literatur

[1] BOSER, Bernhard E. ; WOOLEY, Bruce A.: The Design of Sigma-Delta Modulation Analog-to-Digital Converters. In: *IEEE Journal of Solid-State Circuits*, IEEE, Dezember 1988, S. 1298–1308. – Vol. 23, Issue 6

[2] DIJKSTRA, E. ; O.NYS ; PIGUET, C. ; DEGRAUWE, M.: On the Use of Modulo Arithmetic Comb Filters in Sigma Delta Modulators. In: *ICASSP-88: International Conference on Acoustics, Speech, and Signal Processing*, IEEE, 1988, S. 2001–2004. – Vol. 4

[3] HUSS, Sorin A. ; KLUPSCH, Steffen ; ROSENBERGER, Ralf: Modellierung gemischt analog/-digitaler Schaltungen mit VHDL-AMS. In: *8. GMM-Workshop, Methoden und Werkzeuge zum Entwurf von Mikrosystemen*. Berlin, Dezember 1999. – ISSN 0947-1413

[4] IEEE-SA STANDARDS BOARD (Hrsg.): *IEEE Standard VHDL Analog and Mixed-Signal Extensions*. IEEE Std. 1076.1-1999. New York : IEEE, März 1999. – ISBN 0-7381-1641-6

[5] KLUPSCH, Steffen ; HUSS, Sorin A.: Abstraktionsebenen und ihre Anwendung bei der Modellierung und Simulation von heterogenen Systemen. In: *3. GMM-ITG-GI Workshop Multi-Nature Systems: Optoelektronische, mechatronische und sonstige gemischte Systeme*. Hamburg : Verlag TUHH Technologie GmbH, Februar 2001. – ISBN 3-930400-31-6

[6] NORSWORTHY, Steven R. (Hrsg.) ; SCHREIER, Richard (Hrsg.) ; TEMES, Gabor C. (Hrsg.): *Delta-Sigma Data Converters: Theory, Design and Simulation*. New York : IEEE, 1997. – ISBN 0-7803-1045-4

[7] OPPENHEIM, Alan V. ; SCHAFER, Ronald W.: *Zeitdiskrete Signalverarbeitung*. München : R. Oldenbourg Verlag, 1992. – ISBN 3-486-21554-2