Adv. Radio Sci., 4, 319–322, 2006
www.adv-radio-sci.net/4/319/2006/

Advances in Radio Science

Automatisierte VHDL-Code-Generierung eines Delta-Sigma Modulators

R. Spilka and T. Ostermann

Institut für Integrierte Schaltungen, JK Universität Linz, Altenberger Str. 69, 4040 Linz, Austria

Zusammenfassung. Im vorliegenden Beitrag wird eine automatische Generierung des VHDL-Codes eines Delta-Sigma Modulators präsentiert. Die Koeffizientenmultiplikation wird hierbei durch Bit-Serielle-Addition durchgeführt. Mit Hilfe zweier neuer Matlab Funktionen wird der Systementwurf durch die bekannte Delta-Sigma Toolbox von R. Schreier erweitert und direkt synthesefähiger VHDL Code erzeugt.

1 Einleitung

Stand der Technik im digitalen Schaltungsentwurf ist die Verwendung eines Entwurfablaufes mittels Verhaltensbeschreibssprache (VHDL oder Verilog) und Logiksynthese. Für den Entwurf eines Delta-Sigma Modulators muss zunächst eine entsprechender Systementwurf, in der Regel erfolgt dies mit Hilfe der Delta-Sigma Toolbox für Matlab von Schreier (2004) durchgeführt werden. Anschließend kann der Modulator durch eine analoge oder eine digitale Schaltung realisiert werden. Entscheidend für die automatische Generierung einer digitalen Hardwarebeschreibung, z.B. in VHDL, ist die Umsetzung des Systementwurfs in einen synthesefähigen Code für den weiteren Design-Ablauf.

2 Struktur des Delta-Sigma Modulators

In der Delta-Sigma Matlab Toolbox sind die vier verschiedene Topologien berücksichtigt:

- CIFB : Cascade-of-integrators, feedback form
- CIFF : Cascade-of-integrators, feedforward form
- CRFB : Cascade-of-resonators, feedback form
- CRFF : Cascade-of-resonators, feedforward form

Correspondence to: R. Spilka (spilka@riic.at)

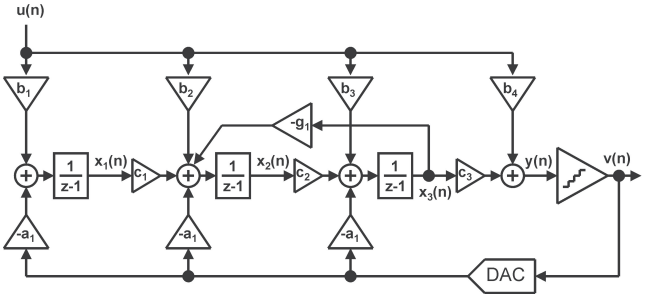

Abbildung 1. CIFB Struktur.

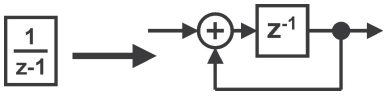

Abbildung 2. Realisierung des Integrators.

In diesem Beitrag wird die CIFB Struktur (Abb. 1) betrachtet. Jedoch ist das vorgestellte Verfahren auch für die anderen Strukturen erweiterbar und nicht auf die CIFB Struktur limitiert.

3 Realisierung als Digitalschaltung

Eine Integration der Integratorübertragungsfunktion lässt sich im digitalen Schaltungsentwurf durch einen Addierer und ein Register für das Rückkopplungssignal realisieren (Abb. 2). Das Ergebnis des Integrators wird in der CIFB-Struktur entsprechend mit den Koeffizienten C_i multipliziert. Allgemein lässt sich eine Multiplikation im digitalen Schaltungsentwurf durch eine bzw. mehrere Verschiebungen und anschliessende Addition implementieren.

Published by Copernicus GmbH on behalf of the URSI Landesausschuss in der Bundesrepublik Deutschland e.V.

Es werden daher alle Koeffizientenmultiplikationen durch eine Bit serielle Addition durchgeführt. Als Beispiel soll eine Multiplikation $15 \cdot 0,4074$ betrachtet werden.

15				
2^4	2^3	2^2	2^1	2^0
0	1	1	1	1

Eine Verschiebung des Binärwertes um eine Stelle nach rechts entspricht einer Division durch 2 bzw. eine Multiplikation mit 0, 5, bzw.:

/2	/4	/8	/16	···
0,5	0,25	0,125	0,0625	···

Wird nun die Anzahl der zur Verfügung stehenden Verschiebungsstufen auf z.B. 4 beschränkt, lässt sich $0,4074$ durch verschiedene Kombinationen darstellen. Der geringste Fehler wird dabei durch eine Division durch 4, 8 und 16 erreicht.

/2 0,5	/4 0,25	/8 0,125	/16 0,0625	Fehler
1	-	-	-	0,0926
-	1	1	1	0,0301
-	1	1	-	-0,0324
-	1	-	1	-0,0949

Viele Verschiebungsstufen erhöhen den Aufwand der nachfolgenden Addierer und können zu einem Timing Problem bei der Implementierung führen. Daher ist es sinnvoll, die Anzahl der verwendeten Verschiebungsstufen weiter einzuschränken. Wird z.B. im obigen Fall eine Beschränkung auf 2 aus möglichen 4 verschiedenen Stufen gewählt, besitzt die Kombination /4 und /8 und damit eine Verschiebung nach rechts um 2 bzw. 3 den geringsten Fehler.

Eine Verschiebung um 2 bzw. 3 Bit nach rechts liefert nach der Addition als Ergebnis für die Multiplikation 4, tatsächlich sollte jedoch $15 \cdot 0,375$ den Wert $5,625$ liefern. Die Abweichung (Fehler) kommt durch die Streichung der Nachkommastellen zustande.

2^4	2^3	2^2	2^1	2^0				
0	1	1	1	1				
0	0	0	1	1	1	1		2Bit nach re.
0	0	0	0	1	1	1	1	3Bit nach re.
0	0	1	0	0				ergibt 4

Abhilfe schafft eine Erhöhung der internen Bitbreite, um die Anzahl der maximal möglichen Verschiebungsstufen. Damit wird nach der Addition der korrekte Wert erreicht.

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
0	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	0	0
0	0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1	0

Da die Limitierung auf die maximale Anzahl von Verschiebungsstufen im Design festgelegt wird, kann im VHDL-Skript die interne Bitbreite entsprechend vergrössert werden.

4 Systementwurf

Zunächst wird mit den Funktionen aus der Matlab Toolbox von Richard Schreier in der Version 7.1. (Schreier, 2004) der Systementwurf durchgeführt.

"synthesizeNTF" liefert die Rauschübertragungsfunktion (Noise Transfer Function) NTF des Delta-Sigma Modulators und "realizeNTF" liefert die Koeffizienten a, g, b, c der gewählten Modulator Topology (hier CIFB).

Es werden anschliessend alle Koeffizienten b_i für $i \geq 2$ zu Null gewählt. Als nächster Schritt wird mit "stuffABCD" die ABCD Matrix berechnet und mit "scaleABCD" werden die c_i-Faktoren angepasst. "mapABCD" berechnet schliesslich die Koeffizienten a, g, b, und c aus der ABCD Matrix.

5 Matlab Funktion "coeff_quant"

Die neue Matlab Funktion "coeff_quant" dient zum Aufstellen der Bit seriellen Addition. Die Anzahl der maximal möglichen Verschiebungstufen wird dabei durch den Parameter "min_limit" gewählt. Bei $1/32$ sind 5 Stufen möglich. Der Wert kann intern auch noch weiter verkleinert werden. Derzeit ist die Funktion so aufgebaut, dass max. 2 aus X Stufen zur Realisierung verwendet werden. Die Funktion optimiert dabei den Fehler und sucht die Kombination mit der kleinsten Abweichung. Eine Erweiterung der Funktion so, dass auch mehr Stufen aus möglichen X Stufen wählbar sind, ist möglich und lässt sich als Eingabeparameter realisieren. Jedoch erhöht sich dabei die Komplexität des Designs und Timing-Probleme können zunehmen. Daher sollte hier eine akzeptable Begrenzung gewählt werden.

```
function [coeff_out, shift1, shift2]
      =coeff_quant(coeff_in, min_limit)
% quantizise an coefficient to sum of
% binary shift operations (max 2)
% the tiniest factor is set to 1/32
%
% function [coeff_out, shift1, shift2]
% =coeff_quant(coeff_in, min_limit)
%
```

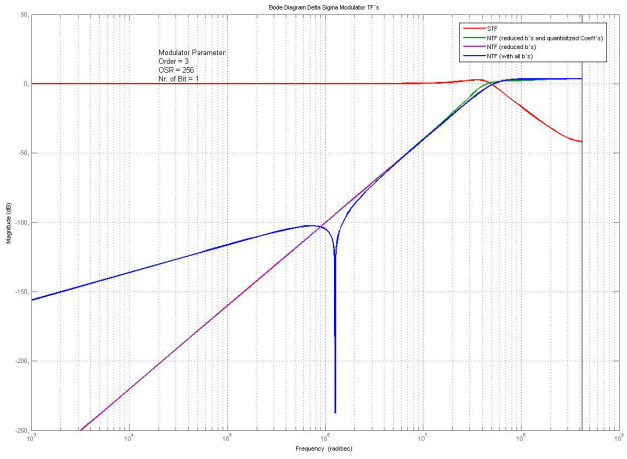

Abbildung 3. STF und NTF eines Modulatorbeispiels.

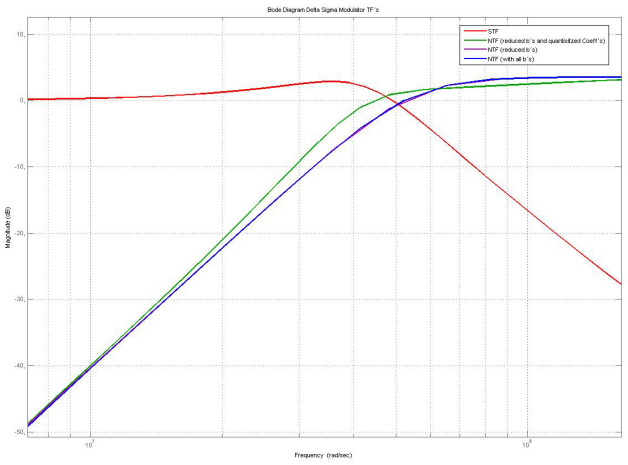

Abbildung 4. Ausschnitt: STF und NTF eines Modulatorbeispiels.

```
% Inputs:
% coeff_in = the coefficient to
%            quantisize
% min_limit = set a coefficient to zero
% if the coefficient is tinier min_limit
%
% Outputs:
% coeff_out = the quantisized
%             coefficient
% shift1 = the first shift operation
%          (binary function)
% shift2 = the second shift operation
% (binary function)
```

6 Matlab Funktion “dac_vhdl”

Mit der Funktion “dac_vhdl” erfolgt die automatische Generierung eines synthesefähigen VHDL-Codes. In der CIFB-Struktur (Abb. 1) ist der wiederholende Aufbau der Additionsstufe (Integrator) erkennbar. Die Eingänge sind c_i bzw. b_1 (alle weiteren b_i wurden zu Null gesetzt), a_i bzw. alternierend g_i, sowie die Rückkopplung des Integrators selbst (Abb. 2). Durch diese sich wiederholende Struktur ist es möglich in der Matlab-Funktion zur Aufstellung des VHDL-Codes eine FOR-Schleife zu implementieren. Hierdurch wird eine flache VHDL-Struktur erreicht. Die Matlab-Funktion “dac_vhdl” verwendet diese Variante. Als Alternative könnte der Integrator als Sub-Design in VHDL definiert und mehrmals eingebunden werden.

```
function [error_out]=dac_vhdl(order,
          bit_in, bit_out, sh1, sh2,
          vhdl_file, rst_active)
% automatic generate VHDL-Code for a
%              1bit-DAC in 'CIFB'-Form
%
% function [error_out]=dac_vhdl(order,
%     bit_in, bit_out, sh1, sh2,
%              vhdl_file, rst_active=0)
%
% Input:
%       order  = Order of the Modulator
%       bit_in  = Number of Inputbits
%       bit_out = Number of Outputbits
%        (only sigle bit is implemented)
%       sh1 = first Shift-Operation for
%         the Parameters (form dsmod_rs)
%       sh2 = second Shift-Operation for
%         the Parameters (from dsmod_rs)
%       vhdl_file = name for the
%             Outputfile excl. extension
%      rst_active = set the Reset-Active
%                   Constant (default 0)
```

Die Matlab Funktion “dac_vhdl” führt folgende Schritte aus:

- Berechnung der minimalen internen Bitbreite (vgl. Bit serielle Addition, die “externe” Bitbreite wird intern entsprechend erweitert um die Nachkommastellen)
- Erzeugung des VHDL-Codes der Einzelblöcke (Integrierer, Koeffizientenmultiplikation, . . .)
- Implementierung des Quantisieres in VHDL
- Implementierung des DDC im Rückkoppelzweig

7 Beispiel

In Abb. 3 ist als Beispiel die STF und NTF eines Delta-Sigma Modulators dritter Ordnung mit Überabtastung (OSR) von

256 und single Bit Quantisierer dargestellt. Die NTF ist hierbei für die folgenden Fälle dargestellt:

- Verwendung aller Koeffizienten b_i (blaue Kurve)
- Reduzierung mit $b_i = 0$ für alle $i \geq 2$ (lila Kurve)
- Reduzierung der b_i und Quantisierung der Koeffizienten (vgl. Bit serielle Multiplikation) (grüne Kurve)

In Abb. 4 ist nochmals ein Ausschnitt aus Abb. 3 dargestellt. Abhängig vom zulässigen Fehler (Abweichung) der Kennlinie, muss gegebenenfalls die Quantisierung erhöht werden (Erhöhung der verwendeten Stufenanzahl), falls der Fehler zu hoch ist.

8 Schlussbetrachtung

Im vorliegenden Beitrag werden zwei Matlab-Funktionen beschrieben mit deren Hilfe nach dem Systementwurf eine automatische Generierung des VHDL-Codes eines digitalen Delta-Sigma Modulators erfolgen kann. Die Koeffizientenmultiplikationen im Delta-Sigma Modulator werden dabei durch Bit serielle Additionen durchgeführt.

Literatur

Schreier, R.: The Delta-Sigma Toolbox Vers. 7.1, 2004.