Published in IEEE Signal Processing Society (DSP Deutschland), 1-7, 2000
which should be used for any reference to this work

Ein DSP-Synthesetool für schnelle Low-Power-Implementierungen von DSP-Algorithmen

Andreas Drollinger, Alexandre Heubi, Peter Balsiger, Fausto Pellandini
Institute of Microtechnology, University of Neuchâtel
Rue A.-L. Breguet 2, 2000 Neuchâtel, Switzerland
URL: www-imt.unine.ch/esplab
e-mail: andreas.drollinger@imt.unine.ch

Zusammenfassung

Dieser Artikel präsentiert ein high-level DSP-Synthesetool für Zyklus-limitierte Filter- und Datenpfadapplikationen. Das Synthesetool bildet einen einfachen und schnellen Designweg von der Gleichungsbeschreibung eines DSP-Algorithmus bis hin zu einer synthetisierbaren VHDL-Beschreibung. Es erzeugt anwendungsspezifische DSP-Architekturen, welche den hohen Randbedingungen von Lowest-Power-Applikationen Rechnung tragen. Sämtliche Operationen der Designsynthese, inklusive der Quantifizierung der einzelnen Operationen eines Algorithmus, werden vom Tool normalerweise automatisch durchgeführt, können jedoch auch vom Anwender kontrolliert und gesteuert werden.

Das Synthesetools zeichnet sich besonders in seiner Einfachheit in der Bedienung und der Schnelligkeit des Syntheseprozesses aus, aber auch in seiner Flexibilität.

1. Einführung

Filteralgorithmen werden meist mittels Datenflussgraphen oder algebraischen Gleichungen beschrieben. Viele kommerzielle und akademische Tools existieren, welche automatisch aus einer solchen Filterbeschreibung eine anwendungsspezifische DSP-Architektur erzeugen. Trotzdem muss gesagt werden, dass unter anderem bei Lowest-Power-Anwendungen diese Tools nicht alle gewünschten Ansprüche befriedigen können, sei es weil Low-Power-Kriterien und -Techniken von den Tools ignoriert werden oder aber weil der Designfluss schwierig zu handhaben ist.

Das Ziel dieser Arbeit ist, einerseits die gewünschten Anforderungen an ein komfortables und trotzdem flexibles DSP-Synthesetool für den Low-Power-Bereich zu definieren, und andererseits ein solches Tool zu realisieren.

Der Artikel ist wie folgt aufgebaut: In einem ersten Teil werden die Kriterien und Bedingungen erörtert, welches das DSP-Synthesetool erfüllen soll. Es folgen einige Erklärungen zum verwendeten Designfluss und zu der Realisation eines solchen Tools. Ein Implementierungsbeispiel und eine Diskussion der Resultate bilden den Abschluss dieses Artikels.

2. Anforderungsprofil eines effizienten, einfach zu bedienenden DSP-Synthesetool

Einfachste Bedienung und effiziente Generierung von anwendungsspezifischen Architekturen sind die beiden wichtigsten Kriterien, welche das high-level DSP-Synthesetool erfüllen soll. Um eine hohe Flexibilität des Tools zu gewährleisten, soll es jedoch auch Anwendungsinteraktionen zulassen und frei programmierbar und erweiterbar sein. Die folgenden Punkten definieren die gestellten Anforderungen in Form eines Pflichtenheftes.

- Das Synthesetool soll im Low-Power-Bereich für Zyklus-limitierte Filter- und Datenpfadanwendungen eingesetzt werden können.
- Es ist autonom und ist nicht gebunden an ein bestimmtes Betriebssystem oder existierende Entwicklungtools.
- Einfache Gleichungen sollen zur Beschreibung des Datenflussgraphes verwendet werden können. Die Syntax soll komfortabel sein, Funktionsdeklarationen zulassen und einen Verzögerungsoperator haben.
- Via einer Ressourcen-Datei können alle zur Verfügung stehenden Ressourcen spezifiziert werden. Die Menge und die Art der Ressourcen soll frei wählbar und der Zielanwendung anpassbar sein.
- Die Hauptausgabe des DSP-Synthesetools ist eine VHDL-Beschreibung, welche von den üblichsten Logik-Synthesetool verarbeitet werden kann. Das Abstraktionsniveau der VHDL-Beschreibung ist RTL.
- Der Designfluss von den Ursprungsgleichungen bis zum fertigen VHDL-Code soll so einfach wie möglich gehalten werden.

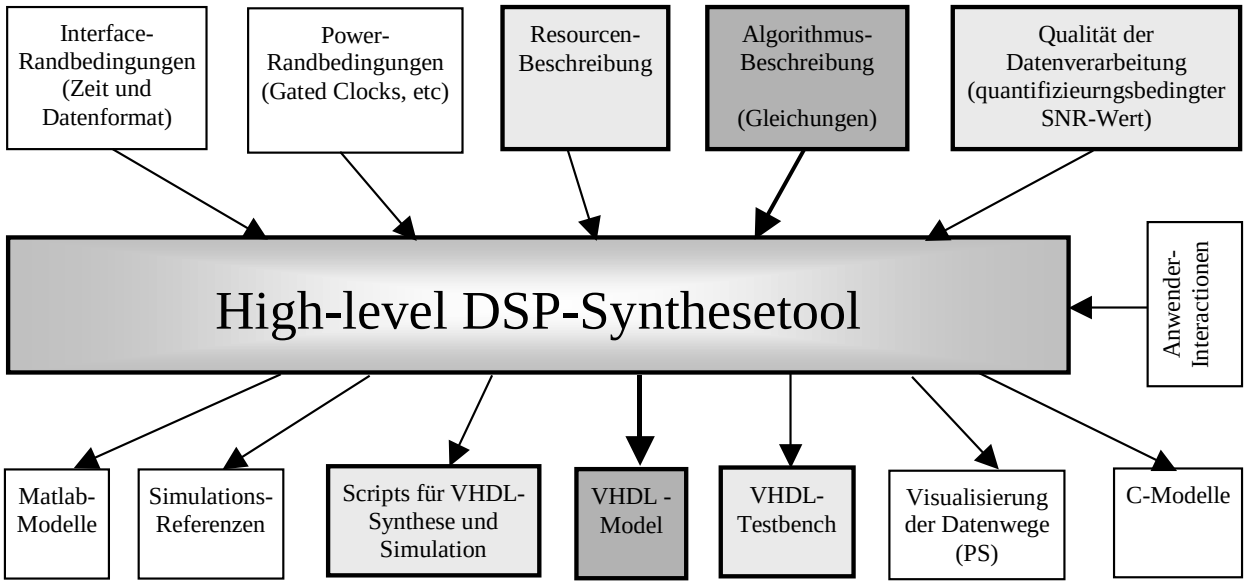

Abbildung 1: Gewünschte In- und Outputs des DSP-Synthesetool: Sind einmal alle Inputs definiert, so sollten diese genügen, um ohne Anwenderinteraktionen sämtliche Outputs zu produzieren.

- Viele Arten von Design-Randbedingungen, welche auf einfachste Weise definierbar sind, sollen berücksichtigt werden können.
- Das Tool soll mit Low-Power-Techniken wie *Gated Clocks* und Optimierung der Systemaktivität vertraut sein.
- Die Quantifizierung der einzelnen Operationen soll vollautomatisch durchgeführt werden. Der Anwender hat dafür lediglich die gewünschte Qualität der Datenverarbeitung zu definieren.
- Das Synthesetool erlaubt, Multimode-DSP-Strukturen zu erzeugen[1].
- Es hat ein komfortables, programmierbares Benutzerinterface. Eine graphische Benutzeroberfläche, Applikationen und Schnittstellen zu anderen Tools wie zu VHDL-Simulatoren, mathematischen Programme, usw. können via diesem programmierbaren Benutzerinterface realisiert werden.
- Eingriffe von Benutzerseite werden ebenfalls über dieses Benutzerinterface ermöglicht, und zwar zu jedem Zeitpunkt. Damit soll der Designfluss selbst kontrolliert, aber auch auf die Datenbasis des Synthesetools zugegriffen werden können.
- Das Synthesetool soll auch die Weiterverarbeitung der generierten DSP-Architektur vereinfachen indem er zum Beispiel für die Simulation eine Testumgebung bereitstellt, Skripts für Simulation und logischer Synthese generiert, Matlab- und C-Modelle erzeugt, usw. Mit Hilfe des programmierbaren Benutzerinterface sollen solche Erweiterungen von Anwendern geschrieben werden können.
- Es kann mittels Plug-Ins anwendungsspezifisch erweitert werden.

3. Die Realisation des Synthesetools

Der Designfluss

Zur Entwicklung des DSP-Synthesetools muss der genaue Ablauf des high-level Syntheseprozesses eruiert sein. Die Synthese kann in eine Reihe von Prozessen aufgeteilt werden, welche in einer festen Folge abgearbeitet werden. Dieses altbewährte Synthesekonzept besitzt auf globalem Niveau keine Iterationsschleifen und ist deshalb einfach und schnell [1]. Sie scheint auch für das neue DSP-Synthesetool einen guten Ansatz zu sein. Abbildung 2 zeigt den gewählten Designfluss. Es folgt eine Beschreibung der verschiedenen Syntheseoperationen zusammen mit einigen Anmerkungen bezüglich dessen Realisierung:

- Der *Parser* liest die Filtergleichungen und die Randbedingungen einer Anwendung. Die Syntax einer solchen Anwendungsbeschreibung ist in Listing 2 ersichtlich. Die Gleichungssyntax ist sehr komfortabel und hat Ähnlichkeiten mit DFL[2]. Sie erlaubt Deklarationen von Unterfunktionen und besitzt einen Verzögerungsoperator.
  Die Anwendungsbeschreibung besitzt hauptsächlich zwei Teile: In einem ersten Teil werden die verwendeten Algorithmen mittels

[1] DSP-Strukturen, welche in verschiedenen Arbeitsmoden arbeiten können

[2] **D**ata **T**ransfer **L**anguage u.a. verwendet von Frontier Design

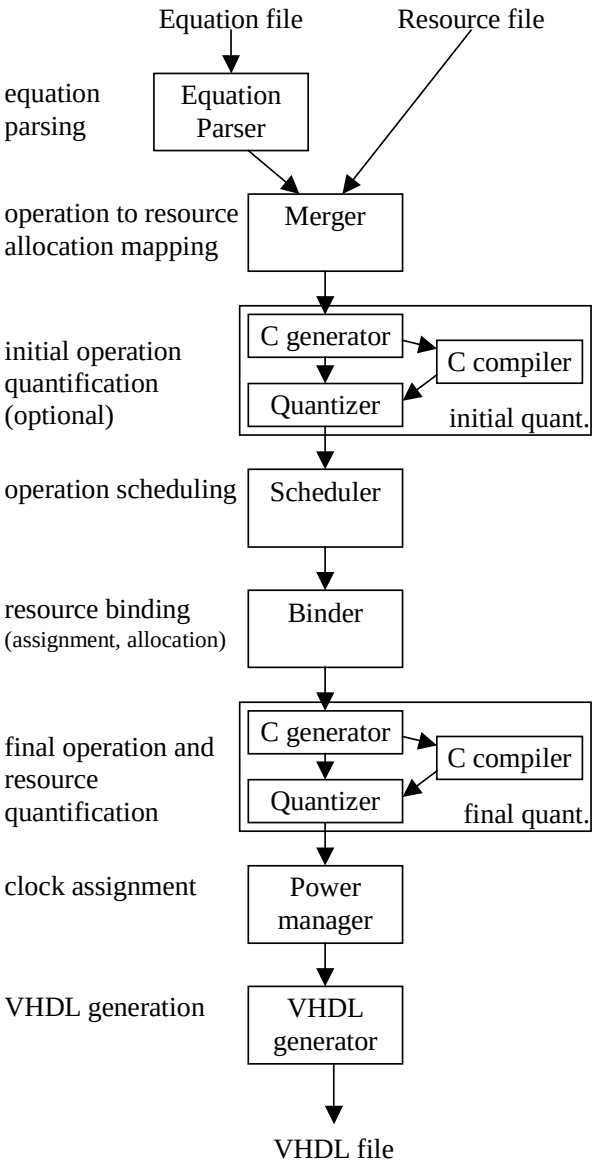

Abbildung 2: Der Designfluss

Gleichungen beschrieben, und in einem zweiten Teil folgen die Definitionen der möglichen Arbeitsmoden mit dessen Randbedingungen.

- Der *Merger* sucht für jede arithmetische Operation eine funktionale Ressourceneinheit, welche die Operation verarbeiten kann. Multiparameter-Operationen können dazu unter Berücksichtigung der kritischen Datenwegen in eine Reihe von Standardoperationen aufgeteilt werden[3]. Sämtliche zur Verfügung stehenden, funktionalen Einheiten können frei in einer Datei spezifiziert werden (siehe Listing 1).
- Es folgt eine optionale *Initialquantifizierung*, welche allen Operationen eine Initialgrösse zuzuweisen. Dies erlaubt die Berechnung von Kostenfunktionen, welche von den folgenden Optimierungsalgorithmen (*Scheduler* und *Binder*) benötigt werden.
  Die Quantifizierung geschieht in drei Schritten: Zuerst werden Gleit- und Festkomma-C-Beschreibungen der Anwendung generiert. Diese werden dann in eine Bibliothek kompiliert und dynamisch in das Synthesetool eingebunden. Die optimale Wortlänge aller Operationen wird schlussendlich mittels Simulation ermittelt. Da die Ressourcenzuweisung noch nicht stattgefunden hat, werden die Operationen als Ressourcen-unabhängig betrachtet.
- Der *Scheduler* bestimmt für jede Operation den Zyklus, an dem die Operation durchgeführt werden soll, und zwar so, dass die Totalkosten aller notwendigen Ressourcen minimiert sind. Die optimale Lösung wird mittels dem Tabu-Search-Algorithmus [4] iterativ ermittelt. Die Optimierungsproblematik selbst wurde schon vor einigen Jahren eingehend diskutiert [1][3] und soll in diesem Artikel nicht neu aufgegriffen werden. Es konnte gezeigt werden, dass mit Hilfe von Tabu-Search eine Zyklus-limitierte Time-Scheduling-Optimierung rasch konvergiert [5].
- Der *Binder* weist unter Berücksichtigung der totalen Anzahl der notwendigen Multiplexer und der resultierenden Systemaktivität jede Operation einer funktionalen Einheit zu. Dies geschieht ebenfalls iterativ unter der Verwendung von Tabu-Search als Optimierungsalgorithmus.
- Nun werden die Operationen endgültig *quantifiziert*. Bei dieser Quantifizierung wird die Ressourcenabhängigkeit der Operationen berücksichtigt.
- Der *Power Manager* teilt jeder funktionalen Einheit einen Clock zu und zwar so, dass die totale Systemaktivität minimiert ist.
- Schlussendlich generiert der *VHDL-Generator* die VHDL-Beschreibung des Filterblockes zusammen mit einem VHDL-Testbench, welcher eine rasche Verifizierung erlaubt.

Die Implementierung des DSP-Synthese-tools

Das Gesamtsystem des DSP-Synthesetools besteht aus zwei Teilen. Der Kern ist in C programmiert und beinhaltet die Datenbasis, sämtliche für den Designfluss benötigten, funktionale Module und ein Tcl/Tk-Interface. Tcl/Tk ist eine Skriptsprache und kann sehr einfach in eine Anwendung eingebunden werden [6].

[3] Beispiel: (a+b+c+d) → ((a+b) + (c+d)

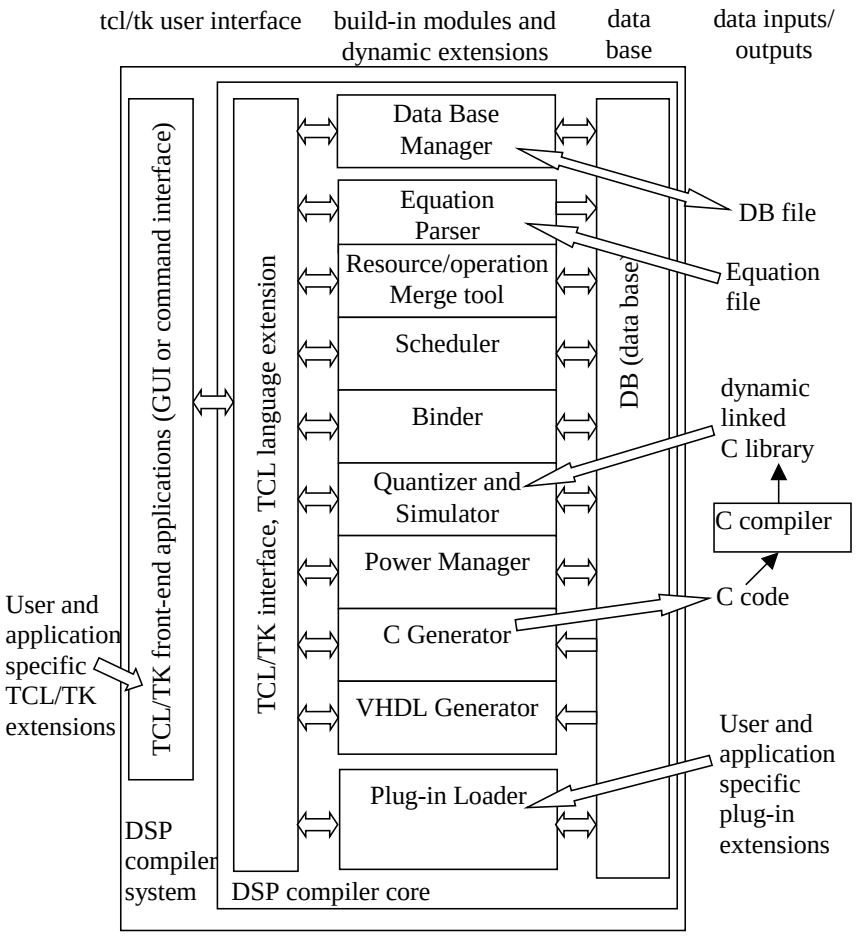

Abbildung 3: Der modulare Aufbau des DSP-Synthesetools

Das Synthesetool kann als Kommandointerpreter, im Graphikmodus und in einem kombinierten Modus gestartet werden. Als erstes liest das Tool ein Initialisierungsskript, welches verschiedenste Einstellungen und Anwenderskripts definieren kann und vorhandene Plugins lädt. Die Arbeit mit dem DSP-Synthesetool kann vereinfacht werden, wenn es im Graphikmodus gestartet wird und die graphische Bedienungsoberfläche *DspC_GUI*[4] geladen wird, welche mit Tcl/Tk realisiert wurde (Abbildung 5). *DspC_GUI* enthält alles für eine einfache Bedienung des Tools. Im Normalfall genügen so zwei, drei Mausklicke, um aus einer Gleichungsbeschreibung eines Algorithmus eine synthetisierbare VHDL-Beschreibung einer anwendungsspezifischen DSP-Architektur zu bilden. Daneben erlaubt *DspC_GUI* auch die graphischen Darstellung der internen Funktionsabläufe der generierten DSP-Architektur und besitzt verschiedenste Schnittstellen zu Simulatoren und mathematischen Programmen.
Das Tcl/Tk-Interface öffnet das Synthesetool für beliebige Erweiterungen. Zusätzliche Menüs, Funktionen, Applikationen und Interfaces zu anderen Programmen können auf einfachste Weise realisiert werden. Lediglich zwei Erweiterungen sollen noch kurz vorgestellt werden (Abbildung 4). Die erste Applikation visualisiert graphisch die Ressourcenzuordnung der Operatoren und dessen Parameterabhängigkeiten und ist sehr praktisch, wenn zum Beispiel bei einer Fehleranalyse verstanden werden muss, wie die Datenverarbeitung des generierten Designs funktioniert. Ebenfalls die zweite Applikation kann bei einer fehlerhaften VHDL-Simulation helfen, die Ursache des Problems zu finden. Sie erlaubt, Zwischenresultate einer internen Simulationen und einer VHDL-Simulation zu vergleichen und Differenzen graphisch hervorzuheben. Auf diese Weise kann die Operation, welche den Fehler erzeugt, schnell und bequem gefunden werden.

4. Anwendungsbeispiel und Resultate

Nebst den eigentlichen Filterapplikationen, wo in regelmässigen Abständen Samples in den Filterblock fliessen, und diesen nach einer gewissen Zeit verarbeitet wiederum verlassen, kann das Synthesetool auch zur Generierung von aufwendigen Datenpfaden verwendet werden. Als Anwendungsbeispiel soll der Datenpfad eines FFT-Prozessors dienen, welcher in Hörgeräten eingesetzt wird und dadurch hohe Randbedingungen bezüglich Grösse, Stromverbrauch und maximaler Taktrate hat [7]. Nebst den einfachen Grundoperationen muss der Datenpfad auch komplexere Operationen wie Radix-4-Operation verarbeiten können.

[4] DspC_GUI: *Dsp Synthesetool´s Graphical User Interface*

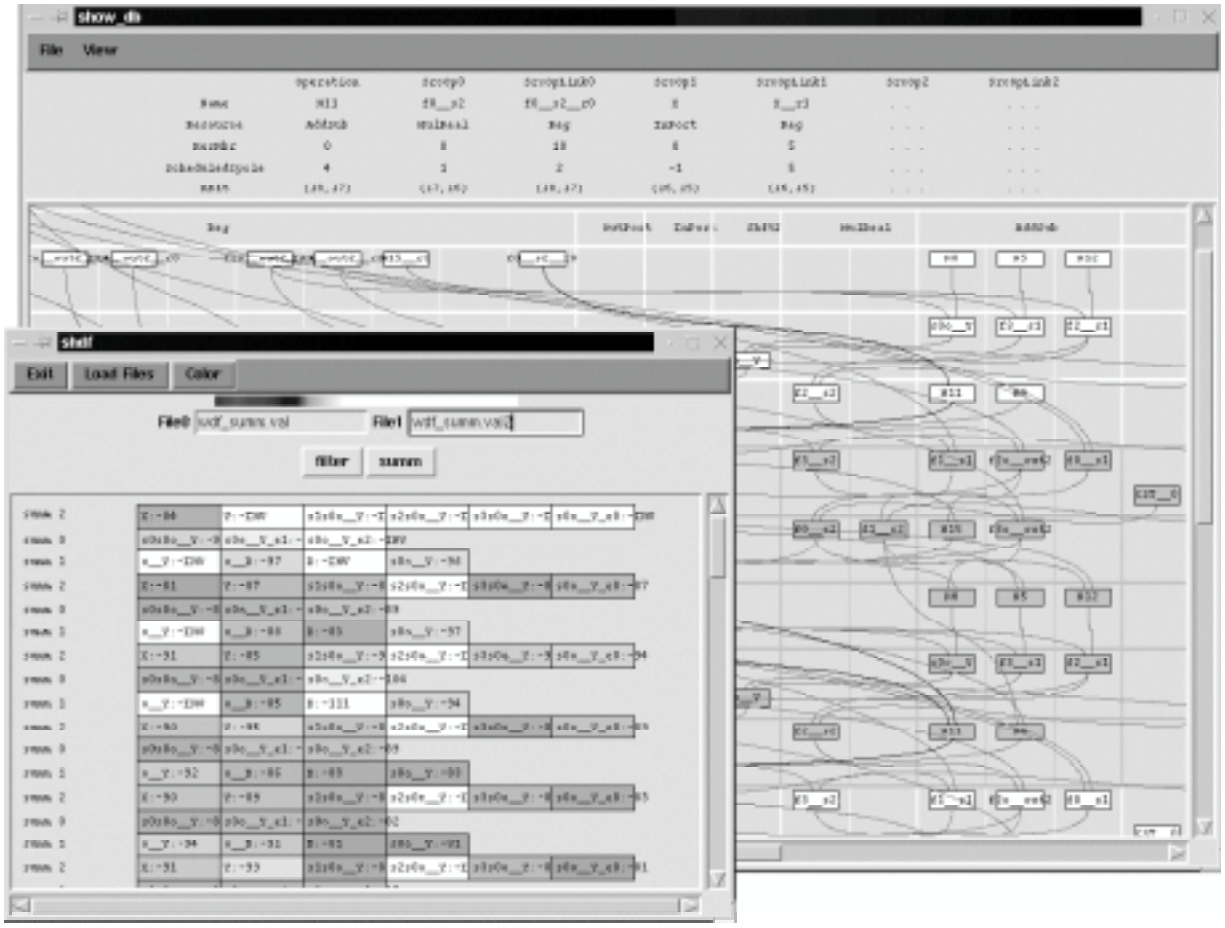

Abbildung 4: Zwei praktische Tcl/Tk-Applikationen zur graphischen visualisierung der Ressourcenzuordnung (grosses Fenster) und der Fehleranalyze mittels Vergleich zwischen einer internen Simulation und einer VHDL-Simulation.

```
#Resource
   Name MulReal
   Operation Mul
   NbrInput 2
   InputSwap 0
   Cost "4*{nBit}+10*{nBitSrc0}*{nBitSrc1}"
   Cost 3200
   Delay 1
   Period 1
   Function C_DOUBLE "{Out}={In0}*{In1};"
   Function C_LONG "{Out}=MulShR({In0},{In1},{nBitFracSrc0}+...
   Function VHDL "{Out}<=Sat(Trim(SIGNED({In0})*SIGNED({In1}),...
   Function MATLAB "{Out}={In0}*{In1};"
```

MulReal ist eine funktionale Einheit. Nebst den Standard-Attributen besitzt sie C- , Matlab- und VHDL-Beschreibungen

```
#Resource
   Name Mul
   Operation *
   NbrInput 2
   GenericExpansion 1
   IF    IsPosPower2Cst(In1) USE << In0 Log2(Abs(In1))
   ELSIF IsPosPower2Cst(In0) USE << In1 Log2(Abs(In0))
   ELSIF IsNegPower2Cst(In1) USE >> In0 Log2(Abs(In1))
   ELSIF IsNegPower2Cst(In0) USE >> In1 Log2(Abs(In0))
   ELSIF IsCst(In1) USE Mul In0 In1
   ELSIF IsCst(In0) USE Mul In1 In0
   ELSIF always USE Mul In0 In1
```

Mul ist eine logische Einheit. Je nach Bedingungserfüllungen werden eine der folgenden Einheiten verwendet: <<, >>, Mul

Listing 1: Ein Ausschnitt aus der Ressourcen-Definitionsdatei: Jede Einheit besitzt Namen, arithmetischen Operationen, Parameteranzahl, Kostenfunktionen, Timing-Spezifikationen und weitere Attribute. Eine Resourceneinheit ist entweder eine logische oder eine funktionale Einheit.

Abbildung 5: DspC_GUI, das graphische Frontend.

```
function [r,i]=RM(r0,i0,r1,i1)                    <- Funktionsdeklaration
    r=r0*r1
    i=i0*i1                                       <- Funktionsname
function [r,i]=RMAC(r0,i0,r1,i1)                  <- @==Verzögerungsoperator
    r=r0@1+r0*r1                                  <- Funtions-Rückgabewert
    i=i0@1+i0*i1
function [r,i]=CM(r0,i0,r1,i1)                    <- Funktionsparameter
    r=r0*r1-i0*i1
    i=r0*i1+r1*i0
function [r0o,i0o,r1o,i1o]=radix2(r0i,i0i,r1i,i1i,rc,ic)
    [r2i,i2i]=CM(r1i,i1i,rc,ic)
    r0o=r0i+r2i
    i0o=i0i+i2i
    r1o=r0i-r2i
    i1o=i0i-i2i
function [r0o,i0o,r1o,i1o,r2o,i2o,r3o,i3o]=radix4n(r0i,i0i,r1i,i1i,r2i,
    [r0m,i0m,r1m,i1m]=radix2n(r0i,i0i,r1i,i1i)          i2i,r3i,i3i)
    [r2m,i2m,r3m,i3m]=radix2n(r2i,i2i,r3i,i3i)
    [r0o,i0o,r2o,i2o]=radix2n(r0m,i0m,r2i,i2i)
    [r1o,i1o,r3o,i3o]=radix2n(r1m,i1m,r3i,i3i)
function [r0o,i0o,r1o,i1o,r2o,i2o,r3o,i3o]=
     radix4(r0i,i0i,r1i,i1i,r2i, i2i,r3i,i3i,r1c,i1c,r2c,i2c,r3c,i3c)
    [r1m,i1m]=CM(r1i,i1i,r1c,i1c)
    [r2m,i2m]=CM(r2i,i2i,r2c,i2c)
    [r3m,i3m]=CM(r3i,i3i,r3c,i3c)
    [r0o,i0o,r1o,i1o,r2o,i2o,r3o,i3o]=radix4n(r0i,i0i,r1m,i1m,r2m,i2m,
                                        r3m, i3m)
mode radix4                                       <- Definition des ersten Arbeitsmodus
    ModeFunction radix4                           <- Funktionsdefinition des Modus
    ModeInfo Period=6                             <- Mode-Periodizität
    OpInfo {r0i,r1i,r2i,r3i}.Resource=Input[0]
    OpInfo {i0i,i1i,i2i,i3i}.Resource=Input[1]    <- Resourcen-Randbedingungen
    OpInfo {r1c,r2c,r3c}.Resource=Input[2]
    OpInfo {i1c,i2c,i3c}.Resource=Input[3]
    OpInfo {r0o,r1o,r2o,r3o}.Resource=Output[0]
    OpInfo {i0o,i1o,i2o,i3o}.Resource=Output[1]
    OpInfo {r0i,i0i,r1i,i1i,r2i,i2i,r3i,i3i}.NbrBit=[18,15]   <- Zahlenformat-Randbedingungen
    OpInfo {r1c,i1c,r2c,i2c,r3c,i3c}.NbrBit=[16,15]
    OpInfo {r0o,i0o,r1o,i1o,r2o,i2o,r3o,i3o}.NbrBit=[18,15]
    OpInfo {r1i,i1i,r1c,i1c}.Cycle=0
    OpInfo {r2i,i2i,r2c,i2c}.Cycle=2              <- Zyklus-Randbedingungen
    OpInfo {r0i,i0i}.Cycle=3
    OpInfo {r3i,i3i,r3c,i3c}.Cycle=4
    OpInfo {r0o,i0o}.Cycle=9
    OpInfo {r1o,i1o}.Cycle=11                     <- Randbedingungen bezüglich der Datenverarbeitsungsqualität
    OpInfo {r2o,i2o}.Cycle=10
    OpInfo {r3o,i3o}.Cycle=12
    OpInfo {r0o,i0o,r1o,i1o,r2o,i2o,r3o,i3o}.SNR=83   <- es folgen die Definitionen
mode MultReal                                     <- der folgenden Arbeitsmoden: MultReal, MacReal,
    ...                                           <- MultComplex, Radix2, Radix4N
GenInfo NbrClk=4                                  <- Maximal Anzahl von (Gated-) Clocks
```

Listing 2: Funktionelle Beschreibung eines Datenpfades eines FFT-Prozessors (aus Platzgründen gekürzt)

Wenn man bedenkt, dass diese Operation bereits 34 arithmetischen Grundoperationen hat, daneben aber noch 5 weitere Operationen zu verarbeiten sind, und wenn man dann noch in Betracht zieht, dass all diese Operationen parallel, von mindestens 6 verschiedenen arithmetischen Einheiten verarbeitet werden müssen damit die benötigte Rechenleistung erzielt wird, erkennt man, dass der Zeitaufwand für den Bau der benötigten Architektur erheblich ist. Das DSP-Synthesetool kann dieses Problem in nur einigen Minuten lösen.

Listing 2 zeigt die Funktionsspezifikationen der Verarbeitungseinheit. In einem ersten Teil des Listings werden via Gleichungen die Algorithmen der Grundoperationen definiert. Die festzustellende Ähnlichkeit mit DFL ist dabei kein Zufall sondern inspirierte wegen der Klarheit und Einfachheit. Sehr praktische Elemente wie der Verzögerungsoperator und Funktionsdeklarationen vereinfachen die Algorithmusbeschreibung.

Im zweiten Teil müssen alle Funktionsmoden spezifiziert werden. Für jeden Mode müssen nebst der Funktionsweise die verschiedenen Randbedingungen definiert werden. Dies sind üblicherweise Interfacebedingungen und die zur Verfügung stehende Anzahl von Clockzyklen.

Um das Listing nicht überlang werden zu lassen, wurden nur einer von insgesamt 6 Arbeitsmoden vollständig definiert.

Je nach dem verwendeten Computer hat das DSP-Synthesetool bereits nach einer Minute (Linux auf Pentium III, 450 MHz) bis 5 Minuten (Solaris 2 auf SUN Ultra 2, 180 MHz) eine optimale Lösung gefunden. Dass sie in Sachen Qualität den handgeschriebenen VHDL-Lösungen nicht nachsteht, zeigt Tabelle 1.

Um die Tabelle korrekt zu interpretieren, müssen folgende prinzipielle Differenzen der drei Lösungen berücksichtigt werden:

- Das DSP-Synthesetool verwendet Multiplikatoren, die anderen Lösungen MAC.
- Die beiden VHDL-Lösungen können die Koeffizienten bei ihrer Verarbeitung invertieren. Dies kann ohne grossen Aufwand mit den beiden MAC gemacht werden. Die Lösung des DSP-Synthesetools hat diese Möglichkeit nicht implementiert.

- Die Lösung des DSP-Synthesetools verwendet arithmetische Blöcke von DesignWare™, einer Bibliothek mit arithmetischen Einheiten von Synopsys. Die beiden anderen Lösungen verwenden für jeden MAC eine von Hand strukturierte Lösung (ein nach Booth kodierter Wallance-Tree, gefolgt von einer Brent&Kung-Addiererstufe).
- VHDL-Lösung II besitzt am Ende der Datenverarbeitungskette Register, welche bei den beiden anderen Lösungen fehlen.

		DSP-Synthese-tool	VHDL-Lösung I	VHDL-Lösung II
Problemdefinition		1 Tag	1 Tag	1 Tag
Implementierung		5 Minuten	5 Tage	5 Tage
Architektur	# Gated Clocks	4	13	11
	# Mult.	2	-	-
	# MAC	-	2	2
	# Adder/Substr	4	4	4
	# Register[5] (16/18 Bit)	11+12	21+14	27+14
	# Mux[6] (16/18 Bit)	96 + (11+12)	49 + (21+14)	33 + (27+14)
Grösse[7]	Register	2294	3054	2712
	Komb. Logik	7786	8248	7806
	Total	10081	11303	10519

Tabelle 1: Vergleich einer DSP-Synthesetool-Lösung mit handgeschriebenen VHDL-Lösungen. Zur Ermittlung des Gatecount wurde die CMOS 0.35-um-Technologie Alcatel-Mietec verwendet.

5. Schlussfolgerungen

Das präsentierte high-level DSP-Synthesetool ermöglicht eine rasche Implementierung von anwendungsoptimierten DSP-Architekturen. Vom Moment an, wo ein DSP-Algorithmus und die Design-Randbedingungen definiert sind, vergehen nur einige Minuten, bis das Tool eine synthetisierbare VHDL-Beschreibung einer optimalen DSP-Architektur zusammen mit einem VHDL-Testbench und Simulationsreferenzen generiert hat.

Das Synthesetool minimiert das Risiko von unachtsam eingeführten Fehlern während des Syntheseprozesses da Anwenderinteraktionen laufend überprüft werden.

Designrandbedingungen können in einer einfachen Syntax definiert werden. Sie garantierten die einfache Einfügung eines generierten Blockes in eine bestehende Umgebung.

Die Quantifizierung der verschiedenen internen Operationen eines Algorithmus wird automatisch durchgeführt. Der Anwender muss dazu lediglich die gewünschte Qualität der Datenverarbeitung

Das DSP-Synthesetool kann entweder voll synchrone Architekturen generieren, oder verwendet aber *Gated Clocks* für Low-Power-Anwendungen.

Der Anwender kann den Designfluss zu jedem Zeitpunkt überwachen und steuern und hat freien Zugriff auf die interne Datenbasis des Tools. Zusammen mit dem TCL/TK-Interface und dem Plug-In-Loader ermöglicht dies die Realisierung von anwendungsspezifischen Erweiterung.

All diese Eigenschaften machen das DSP-Synthesetool zu einem effizienten Werkzeug für die Entwicklung von qualitativ hochstehenden, anwendungsspezifischen DSP-Architekturen.

[5] Der erste Wert entspricht Zwischenregister, der zweite Wert der Eingangsregister der arithmetischen Einheiten

[6] Der in der Klammer stehende Teil entspricht der zusätzlich notwendigen Multiplexer bei Generierung eines synchrones Designs.

[7] Grösse in *Gate Equivalents* (GE)

Referenzen

[1] M. C. Mc Farland, A. C. Parker, R. Camposano, “The high-level synthesis of digital systems”, Proceedings of the IEEE, vol 78. pp. 301-318, February 1990.

[2] S. Amellal, B. Kaminska, “Scheduling Algorithm in Data Path Synthesis Using the Tabu Search Technique”, Proc. of EDAC_EUROASIC93, Paris, France, February 22-25 1993.IEEE Computer Society Press, 1993.

[3] S. Amellal, B. Kaminska, “Functional Synthesis of Digital Systems with TASS”, IEEE transactions on computer-aided design of integrated circuits and systems, vol 13., 5 May 1994.

[4] F. Glover, “Tabu Search, Part I+II”, ORSA J. Computing, pp. 4-32/190-206, 1989.

[5] A. Heubi, P. Balsiger, F. Pellandini, "An Automated Design Methodology for the Mapping of DSP Algorithms into Low Power VLSI Architectures", Proc. of ISIC-97, Singapore, September 10-12, 1997.

[6] J. K. Ousterhout, “Tcl and the Tk Toolkit”, Addison-Wesley Publishing Company,Inc, ISBN 0-201-63337-X.

[7] A. Drollinger, C. Wälchli, D. Sun, L. Grisoni, C. Calame, A. Heubi, P. Balsiger, F. Pellandini, R. Brennan, T. Schneider, "An Ultra Low Power WOLA Filterbank Implementation in Deep Submicron Technology", Proc. of COST 254, Neuchâtel, CH, May, 5-7, 1999.