Entwurf von Controller-Schaltungen für Kommunikationsprotokolle mit dem Protocol-Compiler von Synopsys

Peter Blinzer, Ulrich Golze
TU-Braunschweig, Abteilung E.I.S.
Postfach 3329, Gaußstraße 11, 38023 Braunschweig
Telefon: 0531-391-2380, Fax: 0531-391-5840
E-Mail: (blinzer|golze)@eis.cs.tu-bs.de

Ulrich Holtmann
Synopsys Inc.
700 East Middlefield Road
Mountain View, CA 94043
E-Mail: ulrich@synopsys.com

1 Zusammenfassung

Im Bereich der digitalen Kommunikations-Elektronik werden zur Bearbeitung von Datenprotokollen aus Geschwindigkeitsgründen oft Controller-Schaltungen eingesetzt. Der Protocol-Compiler von Synopsys ist ein neues Werkzeug, das auf diesen Aufgabenbereich spezialisiert ist. Durch eine weitgehend graphische, protokollorientierte Modellierungstechnik und die problemlose Einbettung in etablierte Design-Flows ermöglicht er eine Vereinfachung solcher Entwürfe. In diesem Beitrag werden neben einigen grundlegenden Modellierungstechniken auch Aspekte der Ergebnisqualität anhand realer Entwürfe näher betrachtet.

2 Einführung

Die digitale Datenkommunikation hat in den letzten Jahren unübersehbar an Bedeutung gewonnen, wie die mittlerweile im allgemeinen Sprachgebrauch häufig anzutreffenden Schlagwörter „Internet“, „Mobilkommunikation“ und „Multimedia“ deutlich zeigen.

Aus Gründen der Verarbeitungsgeschwindigkeit, des Energieverbrauchs und der Baugröße werden in Kommunikationsanwendungen oft hochspezialisierte Digitalschaltungen von sehr großer Komplexität eingesetzt. Ihr fehlerfreier und zeitgerechter Entwurf ist eine Herausforderung, die sich mit dem verbreiteten Verfahren der Synthese von manuell generierten Beschreibungen auf Register-Transfer-Ebene in Verilog oder VHDL kaum noch bewältigen läßt.

Eine Lösung dieses Problems durch weitere Abstraktion der Entwurfsebene ohne gleichzeitigen, deutlichen Effizienzverlust ist ohne Berücksichtigung der besonderen Erfordernisse dieses Anwendungsgebietes jedoch nicht möglich. Im Bereich der Datenkommunikation ist ein Großteil der Entwurfskomplexität durch die Strukturierung und den zeitlichen Ablauf der Daten bestimmt, d.h. durch die Kommunikationsprotokolle. Daher erweisen sich einige modernere Methoden der höheren Entwurfsabstraktion, wie die Verhaltenssynthese aus algorithmischen Beschreibungen (z.B. Synopsys Behavioral Compiler) oder Statechart-basierte Ansätze (z.B. Statemate oder Speedcharts), hierbei als weniger gut geeignet.

Die Verhaltenssynthese algorithmischer Beschreibungen ist wegen ihres Operator-Scheduling und ihrer Resource-Allocation gut für Bereiche geeignet, bei denen Datenwerte in imperativer Weise mit einem Kompromiß zwischen Verarbeitungsgeschwindigkeit und Hardwareaufwand transformiert bzw. miteinander verknüpft werden sollen. Da sich aber Kommunikationsprotokolle weder besonders einfach als imperative Algorithmen implementieren lassen noch nichttriviale Transformationen bzw. Verknüpfungen der Datenwerte erfordern, ist die Verhaltensynthese in diesem Fall weniger gut zur Lösung von Entwurfsproblemen geeignet.

Statechart-basierte Entwurfsmethoden sind im allgemeinen gut geeignet, um Systeme mit explizit definierten Zuständen bezüglich ihrer Zustandsübergänge und ihren sonstigen Aktionen für beliebige Eingangsstimuli zu modellieren bzw. näher zu untersuchen. Die Verarbeitung eines Kommunikationsprotokolls läßt sich mit derartigen Systemen gut beschreiben, erfordert aber die Umsetzung der Protokollspezifikation in eine Menge von diskreten Zuständen. Dabei sind auch für die über weite Strecken linearen Protokollabläufe die einzelnen Zustandsübergänge explizit zu modellieren. Es sind also noch etliche zeitraubende Entwurfsschritte vom Entwickler selbst durchzuführen, die bei einer weiteren Spezialisierung des Werkzeugs auch automatisiert werden könnten.

Die Designmethode des Protocol-Compilers [1] berücksichtigt die besonderen Eigenschaften von Kommunikationsprotokollen besser als die beiden vorher genannten Verfahren. Die Entwurfseingabe erfolgt durch Aneinanderreihung bzw. Verschachtelung graphischer Elemente, vergleichbar der Aufstellung von Grammatiken für formale Sprachen. Ihr Aussehen ist zudem der in gedruckten Protokollspezifikationen benutzten Darstellungsweise recht ähnlich. Es müssen dabei weder Zustände noch Zustandsübergänge durch den Entwickler selbst festgelegt werden, da diese Informationen bereits implizit in der Entwurfsstruktur enthalten sind und aus dieser vom Protocol-Compiler synthetisiert werden.

Der Protocol-Compiler baut im wesentlichen auf der Methode der *production based specification* (PBS) auf. Hierbei werden synchrone Automaten auf der Grundlage von grammatischen Produktionen formaler Sprachen beschrieben. Dieses Verfahren wurde unter besonderer Berücksichtigung der High-Level-Spezifikation von Schaltungen und der automatisierten Schaltungssynthese von A. Seawright und F. Brewer eingehend untersucht [2, 3].

Neben den offensichtlichen Vorteilen im Bereich der Entwurfseingabe ist aber auch die Einbindung des Protocol-Compilers in bestehende und bewährte Entwurfsabläufe von elementarer Wichtigkeit im Schaltungsentwurf.

3 Einbindung des Protocol-Compilers in den Entwurfsablauf

Durch den Protocol-Compiler wird der weit verbreitete Entwurfsablauf der HDL-Synthese auf einfache Weise erweitert. Statt eine Beschreibung in Verilog oder VHDL auf Register-Transfer-Ebene von Hand zu schreiben, wird diese nun aus dem Protocol-Compiler-Entwurf automatisch generiert. Der übrige Ablauf aus Synthese, Plazierung, Verdrahtung und Post-Layout-Verifikation wird in keiner Weise verändert.

Die Simulation der Protocol-Compiler-Entwürfe erfolgt innerhalb dessen graphischer Benutzeroberfläche unter Verwendung eines Verilog- oder VHDL-Simulators. Hierdurch wird die Einbindung von eigenen Testrahmen, zusätzlich benötigten HDL-Codes oder bereits vorhandenen Entwürfen möglich.

Ein Entwurfsablauf mit dem Protocol-Compiler unter Anwendung des Design-Compilers von Synopsys zur Schaltungssynthese stellt sich somit vereinfacht wie in Bild 1 dar.

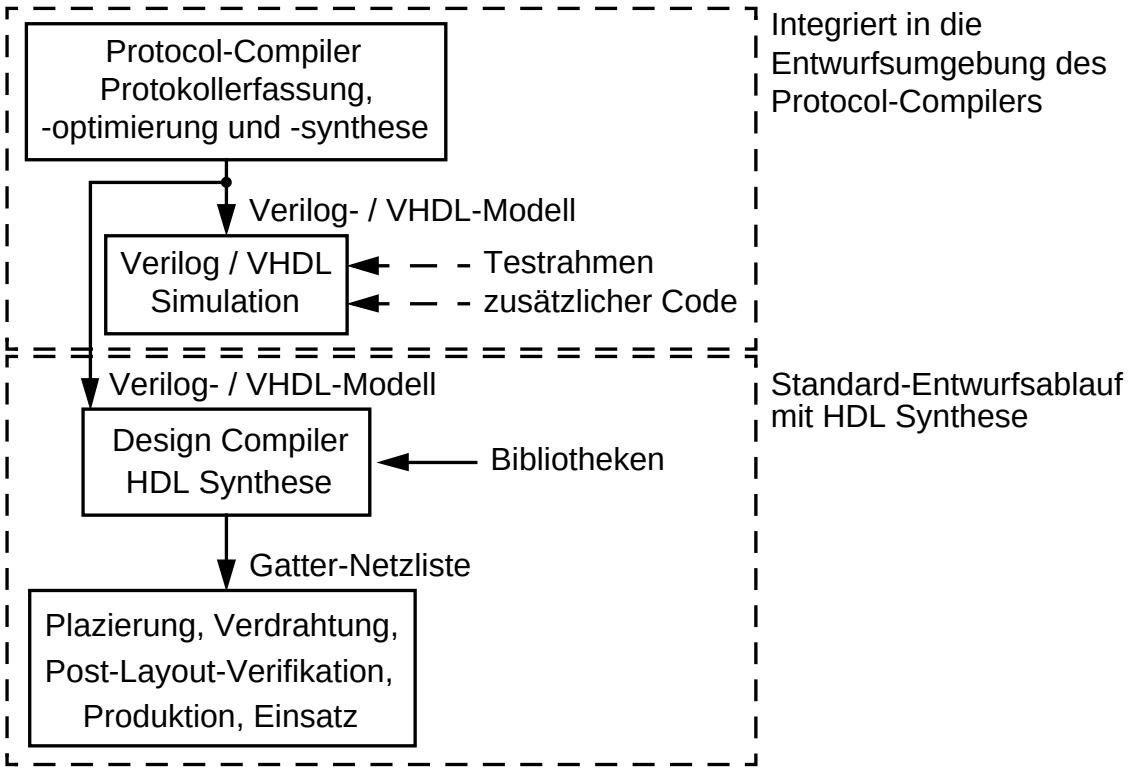

Bild 1: Vereinfachte Darstellung des Entwurfsablaufs

Aufgrund der Semantik der Entwurfselemente des Protocol-Compilers ist die Synthesetauglichkeit des erzeugten HDL-Codes in jedem Fall sichergestellt. Die Entscheidung, welche der beiden HDLs Verilog bzw. VHDL letztendlich besser zu verwenden ist, bleibt dabei vollständig dem Benutzer und dem Projektumfeld überlassen, da beide Sprachen gleich gut unterstützt werden.

4 Die Entwurfselemente des Protocol-Compilers

Die Entwürfe bauen auf einer sehr überschaubaren Menge von Basiselementen auf, die teils stark an die aus formalen Sprachen bekannten Elemente erinnern. Damit die Entwürfe nicht nur auf Stimulidaten reagieren können sondern auch Daten manipulieren können, gibt es zusätzlich Operationen auf Variablen und Ports.

Die Entwurfselemente lassen sich in drei Klassen unterteilen:

- Frames
- Actions
- Operatoren

Die sogenannten Frames sind sehr gut mit den Symbolen in formalen Sprachen vergleichbar. Es gibt sowohl primitive Frames, welche die Funktion terminaler Symbole haben, als auch Referenz-Frames, die den nichtterminalen Symbolen entsprechen. Die Frames können außerdem durch Operatoren zu Sequenzen oder Alternativen zusammengestellt werden, als Option deklariert werden, an Bedingungen geknüpft werden oder aber auch wiederholt werden.

Die Frames werden durch Rechtecke dargestellt, in denen die Bedingung für die Bearbeitung eingetragen ist (primitive Frames) oder eine Referenz auf eine Frame-Definition (Referenz-Frames). Wird ein primitives Frame bearbeitet, so wird hierdurch ein Takt des zugrundeliegenden Automatenmodells verbraucht. Außerdem kann die Bearbeitung eines Frames mit Aktionen verbunden werden, wie der Berechnung und Zuweisung eines arithmetischen Ausdrucks. Bild 2 zeigt einige Beispiele für einfache Frame-Definitionen.

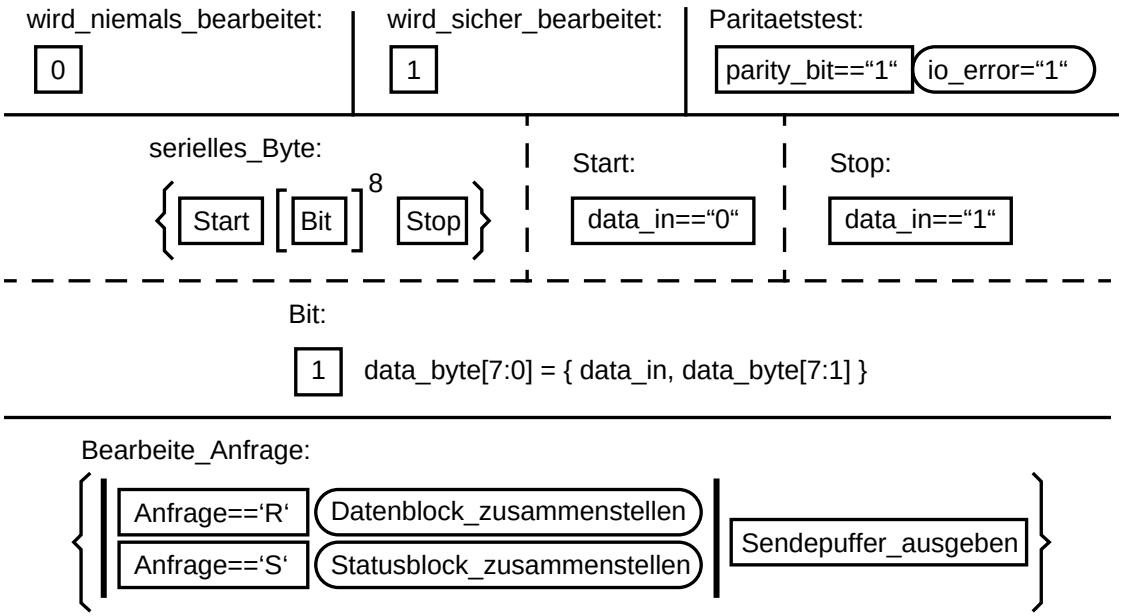

Bild 2: Beispiele für einfache Frame-Definitionen

Mit den drei Operatoren „Sequenz“ (geschweiftes Klammerpaar), „Alternative“ (vertikale Balken) und „Wiederholung“ (eckige Klammern mit Exponent) lassen sich in Kombination mit der hierarchischen Struktur relativ leicht komplexere Modelle erstellen. Durch Abwandlung des Wiederholungs-Exponenten kann die Ausführung optional (kein Exponent) oder beliebig oft erfolgen (∗). Als Aktionen können außerdem auch auf HDL-Ebene beschriebene Funktionen ausgeführt werden.

Wird die Bedingung eines Frames jedoch nicht erfüllt, so wird dieses Frame nicht bearbeitet und eine es einbettende Frame-Sequenz abgebrochen. Dies ist analog zu der Verwerfung von grammatischen Produktionen bei der Bildung von Ausdrücken in formalen Sprachen.

Neben diesen grundlegenden Entwurfselementen stehen noch zwei weitere leistungsfähige Operatoren zur Verfügung, die bei der Datensynchronisation und Fehlerbehandlung von Kommunikationsprotokollen die Modellierung stark vereinfachen.

Hierbei handelt es sich zum einen um den Bedingungs-Operator (Qualifier), der die Ausführung der von ihm umgebenen Frames nur zuläßt, wenn eine bestimmte Bedingung erfüllt ist. Ansonsten wird die Ausführung abgebrochen. Zum anderen kann mit dem Warte-Operator (RunIdle) die Abarbeitung von Frame-Sequenzen solange angehalten werden, bis eine bestimmte Bedingung erfüllt wird. Diese Operatoren werden zusammen mit einigen grundsätzlichen Modellierungstechniken im Rahmen des folgenden Beispiels näher erläutert.

5 Ein Entwurfsbeispiel

In diesem Abschnitt werden anhand von Ausschnitten aus einem Entwurf, der mit Hilfe des Protocol-Compilers bereits real implementiert wurde, allgemein verwendbare Modellierungstechniken vorgestellt.

Bei dem Entwurf handelt es sich um eine Schaltung zur Verarbeitung eines digitalen Audiosignals nach SPDIF-Standard [4], wie es z.B. von Audio-CD-Playern und DAT-Recordern zum Datenaustausch verwendet wird. Dieses Signal besitzt drei Protokollebenen (Bits, Blöcke und Datenblockrahmen) und eine Bitrate von 5,6MBit/s. Aufgrund der Datenrate und der Strukturierung der Daten ist die Verarbeitung des SPDIF-Signals bereits ein nichttriviales Beispiel für eine reale Anwendung, dessen Modellierungstechniken problemlos auf andere Anwendungsgebiete übertragbar sind.

Ziel des Entwurfes war, das SPDIF-Signal zu empfangen, zu dekodieren und die darin enthaltenen Audio-Daten für eine akustische Wiedergabe auszugeben. In der obersten Ebene der Modellhierarchie ergaben sich damit vier Referenz-Frames, je ein Frame für jede Protokollebene und eines für die Datenausgabe an einen Digital-Analog-Converter (Bild 3). Alle vier Frames arbeiten parallel, da keine der Ausführungsalternativen modellierungsbedingt verworfen werden kann, und kommunizieren über Variablen miteinander.

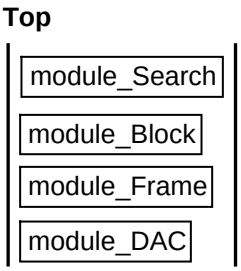

Bild 3: Oberste Ebene der Modellhierarchie des SPDIF-Receivers

Empfang und Dekodierung des Bitstroms erfolgen im Frame module_Search (Bild 4), welches die Daten und deren Synchronisation an die übrigen Frames weitergibt.

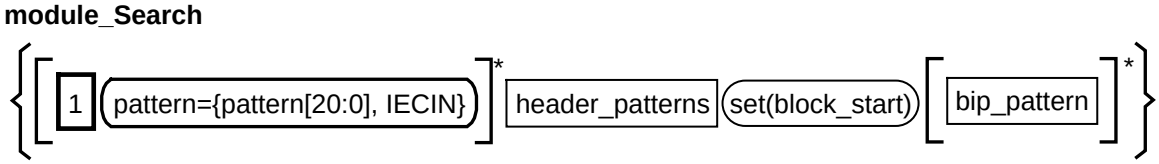

Bild 4: Protokoll-Synchronisierung und Bit-Dekodierung

Das Frame module_Search sucht in dem Schieberegister pattern ständig nach den Synchronisations-Bitmustern (header_patterns) des Datenstroms und führt nach der Auffindung die Dekodierung der biphasencodierten Bits durch. Der erste Wiederholungs-Operator hat dabei zwei Auswirkungen. Zum einen wird mit jedem Takt des modellierten Controllers ein Bit in pattern eingeschoben, also ein Schieberegister modelliert. Zum anderen wird aber auch eine Ausführungs-Pipeline für module_Search generiert, damit trotz der ständig ausgeführten Wiederholung der Schiebeoperation die Bearbeitung des nächsten Frames (header_patterns) möglich ist. Der zweite Wiederholungs-Operator bleibt nach seiner Aktivierung parallel zum ersten solange aktiv, bis der Biphasencode durch ein neues Synchronisations-Muster verletzt wird, so daß bip_pattern abbricht.

Die Erkennung der Synchronisations-Bitmuster wurde über zwei Taktzyklen verteilt (Bild 5). Hierdurch konnte im Vergleich zu einer innerhalb eines Taktes arbeitenden Lösung die Laufzeit der kombinatorischen Logikpfade erheblich verkürzt werden. Das Ziel-FPGA (ein Xilinx 3042A-7) konnte daher problemlos mit dem aufgrund der Signalabtastung benötigten Takt von 16MHz betrieben werden.

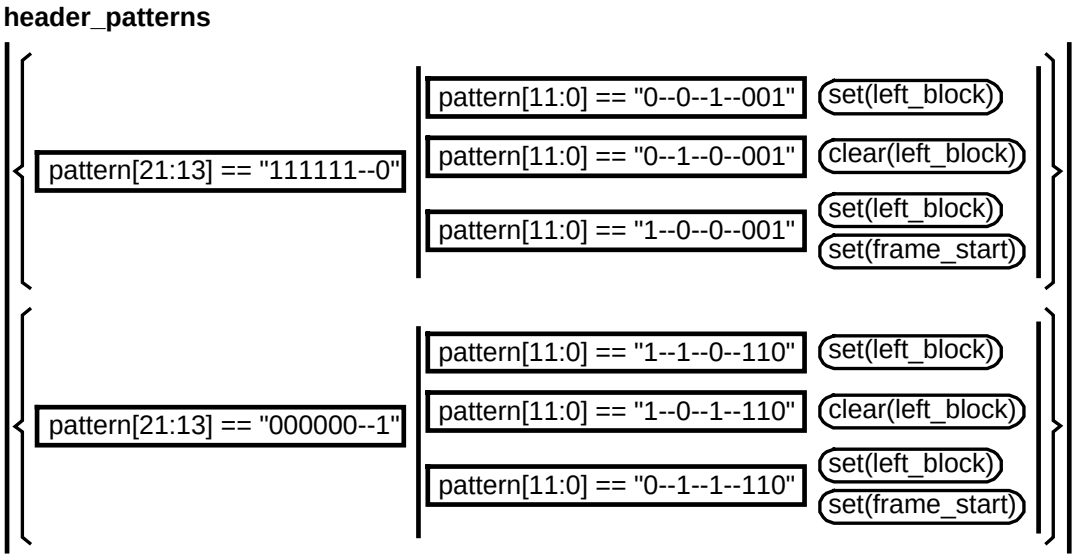

Bild 5: Die Suche nach den Synchronisations-Bitmustern

Die mit Strichen ausgesparten Stellen in den Vergleichs-Bitmustern nehmen die Positionen vom Vergleich aus (dont‘t care) , die aufgrund ihrer Nähe zu den Flanken der Synchronisation-Muster als instabil angenommen werden müssen. Da header_patterns durch das übergeordnete Frame in jedem Takt gestartet wird, wird auch hier automatisch eine Ausführungs-Pipeline angelegt.

Nach dem erfolgreichen Auffinden der Synchronisations-Bitmuster wird die Ausführung von module_Search in dem Frame bip_pattern fortgesetzt, welches im wesentlichen eine digitale PLL zur Regeneration des Datentaktes des SPDIF-Datenstroms enthält (Bild 6). Je nach der Lage der Signalflanken im Abtastraster wird hier nach 5 oder 6 Takten ein neues Datenbit dekodiert, um die bestmögliche Näherung des Datentaktes an den Signaltakt zu erreichen.

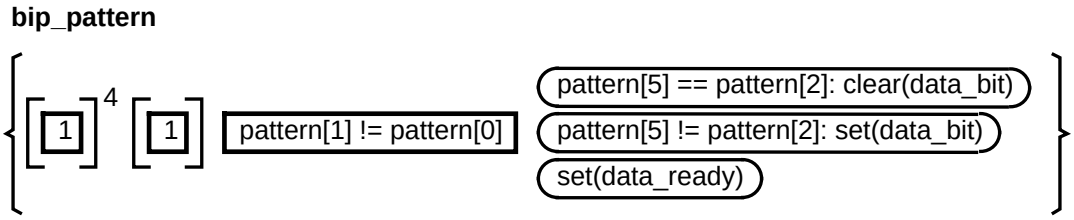

Bild 6: Die digitale PLL und die Bit-Dekodierung

Tritt jedoch weder nach fünf noch nach sechs Takten eine Flanke im Eingangssignal auf, wie z.B. im Fall von Synchronisations-Bitmustern, so beendet das Frame seine Ausführung bis zu seinem erneuten Start durch module_Search.

Die dekodierten Datenbits werden von der parallel zur Bitverarbeitung aktiven Blockverarbeitung (module_Block) weiterverarbeitet (Bild 7).

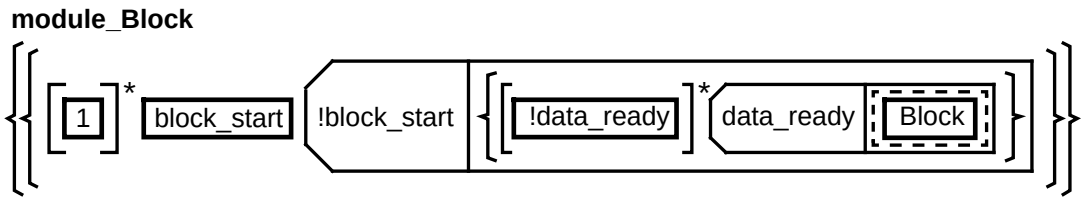

Bild 7: Verarbeitung von Datenblöcken

Hier wird zunächst auf den Beginn eines neuen Datenblockes gewartet, welcher durch die Variable block_start angezeigt wird. Der übrige Ablauf des Frames ist in einen Bedingungs-Operator (Qualifier) gekapselt, wodurch im Falle einer fehlerhaften Blocksynchronisation (block_start wird innerhalb eines Blockes erneut aktiv) die laufende Blockbearbeitung abgebrochen wird. Dies vermeidet eine unnötig große Ausführungs-Pipeline, da ein neuer Eintritt eine Beendigung der laufenden Ausführung bedeutet.

Die schrittweise Verarbeitung der einzelnen Datenbits eines Blockes wird über einen Warte-Operator (RunIdle) gesteuert. Dieser schaltet die Ausführung von Block mit jedem Bit (angezeigt durch data_ready) um einen Verarbeitungsschritt weiter. Das Frame Block selbst ist entsprechend des gegebenen Aufbaus eines SPDIF-Datenblocks organisiert (Bild 8).

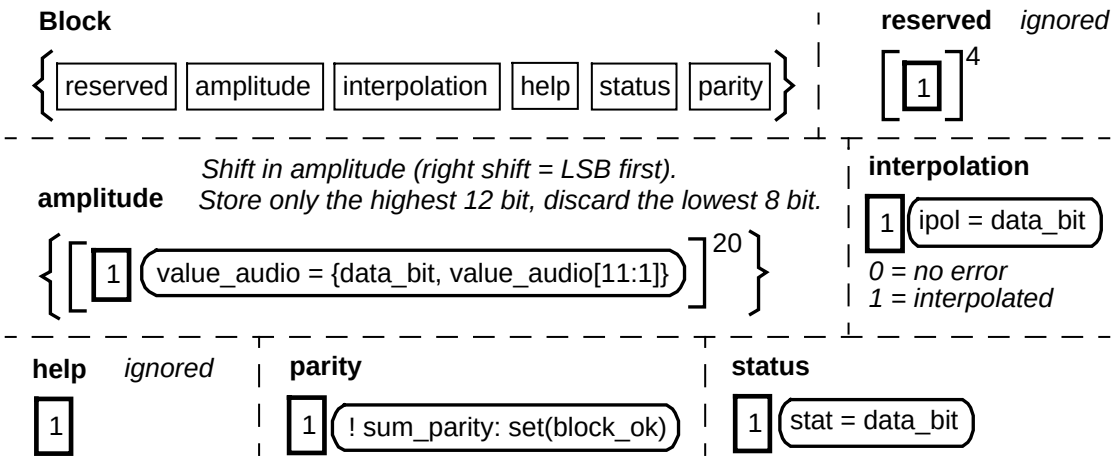

Bild 8: Unterframes der Datenblock-Verarbeitung

Die Verarbeitung der Datenblockrahmen sammelt im wesentlichen den über mehrere Datenblöcke in die Block-Statusbits eingebetteten Status-Bitstrom. Von der Modellierung ist dieser Vorgang der Blockverarbeitung sehr ähnlich, so daß hier nicht näher auf diesen Teil des Modells eingegangen wird.

Die Übertragung der aus dem SPDIF-Signal gewonnenen Audio-Abtastwerte an den Digital-Analog-Converter (DAC), der zur Audio-Wiedergabe in Pulse-Code-Modulation (PCM) verwendet wird, erfolgt in einem einfachen seriellen Protokoll. Da hierbei lediglich eine triviale Sequenz aus primitiven Frames zur Anwendung kommt, wird auch in diesem Fall auf eine Darstellung verzichtet.

Anhand des vorangegangenen Beispiels wurden grundsätzliche Methoden demonstriert, die zur Erstellung einer real funktionsfähigen Schaltung mit dem Protocol-Compiler verwendet werden können. Die in diesem konkreten Fall erzielten Entwurfsergebnisse sind Inhalt des nächsten Abschnitts.

6 Ergebnisse

Der Protocol-Compiler-Entwurf des SPDIF-Dekoders wurde unter Verwendung des Synopsys Design-Compilers und der XACT-Tools von Xilinx erfolgreich in ein Xilinx XC3042A-7 FPGA umgesetzt. Da entsprechende Schaltungen bereits mittels Schematic-Entry und Synthese handgeschriebener Verilog-Modelle entwickelt worden waren, ergab sich eine gute Möglichkeit zum Vergleich von Entwurfsdaten zwischen den verschiedenen Entwurfsstilen. Tabelle 1 gibt einen Überblick über charakteristische Daten der verschiedenen Entwürfe.

Entwurfsstil	Logikblöcke	maximale Laufzeit	Entwurfsdauer
Protocol Compiler	103 CLBs	53.4ns	2-3 Tage
Verilog-RTL + Synthese	90 CLBs	53.2ns	1 Woche
Verilog-RTL + Schematics	114 CLBs	33.6ns	4-5 Wochen

Tabelle 1: Ergebnisdaten verschiedener Entwurfsstile

Durch die Verwendung des Protocol-Compilers für den SPDIF-Dekoder wurde im Vergleich zum Entwurf in Verilog auf Register-Transfer-Ebene mit Synthese eine Halbierung der Entwurfszeit erreicht. Es wurden in diesem Fall etwa 10% mehr Chip-Ressourcen bei vergleichbarem Timing benötigt.

Es muß allerdings hinzugefügt werden, daß bei dem Protocol-Compiler-Entwurf im Bezug auf die FSM-Optimierung, HDL-Umsetzung und Anpassung an die Zielarchitektur (insbesondere Poweron-RESET des FPGAs) eine Vielzahl an Möglichkeiten zur Verbesserung des Ergebnisses angewendet wurden. Diese Möglichkeiten erforderten jedoch eine genaue Kenntnis der tiefer liegenden Ebenen (RTL-Ebene, Gatterebene, FPGA-Architektur) und waren aufgrund der hohen Entwurfsabstraktion nicht sehr einfach nutzbar. Ein erster lauffähiger Entwurf benötigte ohne diese Optimierungen 139 von 144 verfügbaren CLBs und lag mit 61.4ns knapp unter der zulässigen Obergrenze von 62.5ns.

7 Zusammenfassung

Der Protocol-Compiler ist ein Werkzeug für die Modellierung von Controller-Schaltungen im Bereich von Kommunikationsprotokollen. Die Modellierung von solchen Entwürfen ist gut an das Aufgabengebiet angepaßt und erlaubt eine schnellere und besser verständliche Problemlösung als der klassische HDL-Entwurf, wobei Ergebnisse von konkurrenzfähiger Qualität erzielbar sind.

Die Einbindung in bestehende Entwurfsabläufe erweist sich ebenso wie die Wiederverwendung bestehender HDL-Entwürfe als unproblematisch. Es gibt des weiteren vielfältige Möglichkeiten zur Auslotung des Lösungsraumes, die aber nicht immer leicht erkennbar und zugänglich sind. Hier bieten sich durchaus Ansätze für Verbesserungen und weitere Untersuchungen.

8 Literaturangaben

[1] A. Seawright, U. Holtmann, W. Meyer, B. Pangrle, R. Verbrugghe and J. Buck; *A System for Compiling and Debugging Structured Data Processing Controllers*; European Design Automation Conference 1996, Geneva, Switzerland

[2] A. Seawright, F. Brewer; *Clairvoyant: A Synthesis System For Production-Based Specification*; IEEE Transactions on VLSI Systems, June 1994

[3] A. Seawright, F. Brewer; Synthesis from production-based specification; 29th Design Automation Conference, 1992, Anaheim, California

[4] International standard IEC958, *digital audio interface*, First edition 1989-03