Schnittstellenbasierter Entwurf mit wiederverwendbaren Modulen

Interface Based Design with Reusable Modules

Gordon Cichon, Technische Universität Dresden

Dieser Beitrag beschreibt den Einsatz von On-Chip-Bussen und die Entwicklung von Protokolladaptern für verschiedene zum Einsatz kommende Busprotokolle. In diesem Artikel wird vorgestellt, wie man ausgehend vom Konzept des VCI-Standards *On-Chip-Bus* durch eine Beschränkung auf Lese- und Schreibtransaktionen in der Busmodellierung die Funktionalität eines Interfaces auf einen Speicherbereich abbilden kann. Es wird gezeigt, wie aus solch einer Beschreibung Teile der Kommunikationsinfrastruktur automatisch synthetisiert werden können.

This article describes the application of on-chip buses and the development of protocol adapters for different protocols. In the discourse, it is explained, how the functionality of a module's interface can be mapped into a memory interface based on the VCI On-Chip-Bus standard if transactions on this interface can be constrained to solely read- and write-transactions. It is shown, how parts of the internal communication infrastructure can be synthesized from such a constrained specification.

1 Einleitung

Der stetig wachsende Preisdruck in der Konsumgüterindustrie zwingt Elektronikproduzenten, immer mehr Funktionalität auf immer kleinerem Raum zu integrieren. Wurde früher die Elektronik noch aus vielen einzelnen Chips auf einer Platine (PCB, Printed Circuit Board) zu einem System zusammengebaut, so bemüht man sich heute, ein Gesamtsystem möglichst auf einem einzigen Chip zu integrieren, da die Materialkosten für die Platine selbst und die Verpackung der einzelnen Bauteile die größten Kostenfaktoren bei der Herstellung von elektronischen Geräten darstellen.

Die Entwicklung solcher Gesamtsysteme auf einem Chip (SoC, System on Chip) unterscheidet sich stark von der traditionellen Chipentwicklung. Die traditionelle Chipentwicklung verfolgt die Lösung eines abgegrenzten technischen Problems, die möglichst allgemein und wiederverwertbar sein soll. Die moderne SoC-Entwicklungsmethodik ähnelt eher derjenigen, die traditionell für Platinen angewendet wird (siehe Bild 1).

Als Beispiel soll in diesem Artikel ein Modul zur Verschlüsselung von Daten betrachtet werden, wie es in Mobilfunktelefone der neuesten Generation eingebaut wird. Dazu soll das verbreitete und gut verständliche System der sogenannten DES-Verschlüsselung [4] genauer beleuchtet werden.

Einen Überblick über den Algorithmus vermittelt Bild 2. Die zu verschlüsselnden Daten werden in 64 Bit große Blöcke aufgeteilt. Jeder dieser Blöcke wird einer Vorverarbeitung unterzogen, danach werden 16 gleichartige Verschlüsselungsrunden hintereinander durchgeführt, die den Datenblock mit jeweils

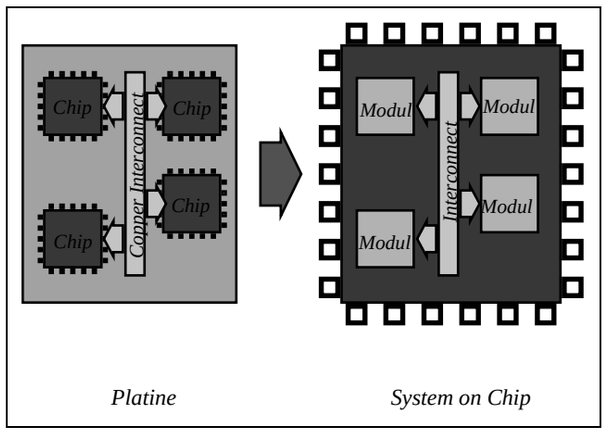

Bild 1: Von der Platine zum SoC (System on Chip).

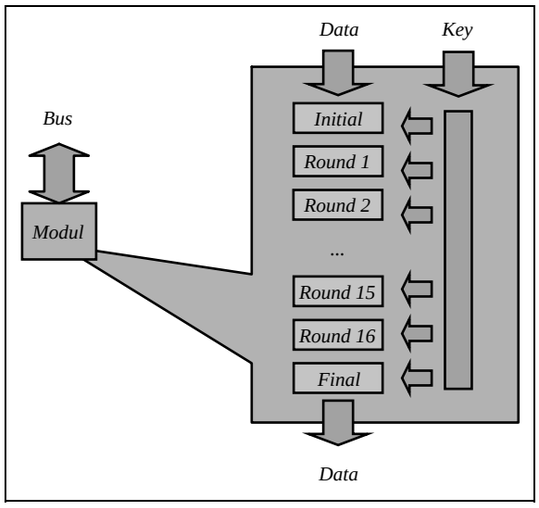

Bild 2: Algorithmus der DES-Verschlüsselung.

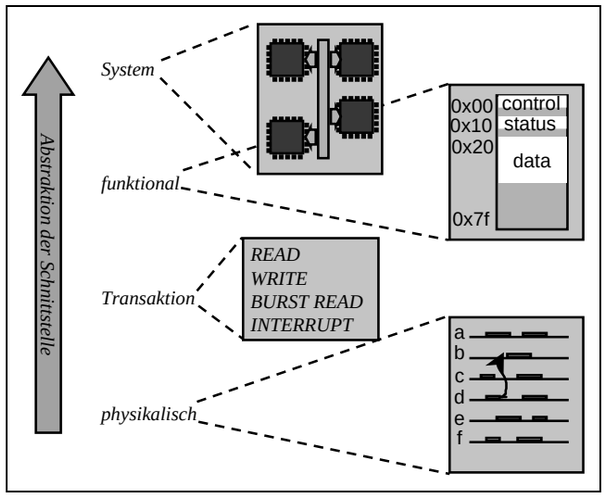

Bild 3: Abstraktionsebenen der Schnittstellen.

einer anderen Untermenge des 56 Bit-Schlüssels unkenntlich machen. Jeder Runde dient als Eingabe das 64 Bit-Datenergebnis der vorausgehenden Runde.

2 Schnittstellen von Modulen

Um die Zusammenarbeit verschiedener Entwickler in einem Team zu ermöglichen und die Wiederverwendbarkeit einzelner Bestandteile in verschiedenen Projekten zu erleichtern, wird ein Projekt üblicherweise in verschiedene Module oder Komponenten unterteilt, zwischen denen Schnittstellen definiert werden. Eine solche Schnittstelle wird häufig auch Bus genannt.

Beim traditionellen Platinendesign sind Chips die Bausteine und bilden so eine natürliche Einheit für Module. Ein solches Modul – jeweils auf einem eigenen Chip – implementiert eine bestimmte Funktion und besitzt durch die Anschlussleitungen (Pins) eine definierte Schnittstelle. Diese Schnittstelle wird im Datenbuch des Chips definiert, beispielsweise in Form von Spannungen und Strömen der Anschlusssignale, Anforderungen an die zeitliche Signalabfolge (Timing), usw.

Im Gegensatz hierzu werden SoCs nun nicht mehr aus fertig hergestellten und physikalisch verpackten Einzelteilen hergestellt. Die Module werden statt dessen durch Programme beschrieben, die die Funktionsweise der späteren Hardware simulieren. Diese Darstellungsform wird *Intellectual Property* (IP), zu Deutsch *geistiges Eigentum*, genannt. Nach dem Zusammenbau dieser Module zu einem System wird die Darstellung von einem weiteren Programm, der sogenannten *Synthese*, in einen Schaltplan und später in ein Layout, also letztendlich in die eigentliche Hardware, umgewandelt.

Bei dieser Vorgehensweise müssen die Schnittstellen zwischen den Modulen nun also nicht mehr vor dem Zusammenbau bis ins kleinste Detail festgelegt werden. Parameter – wie z. B. Spannungspegel – sind teilweise in der Beschreibungsform von IP noch gar nicht definiert. Bild 3 gibt einen Überblick über den unterschiedlichen Detaillierungsgrad der Schnittstellenbeschreibungen auf verschiedenen Abstraktionsebenen, die in den folgenden Absätzen genauer behandelt werden.

Im Beispiel soll unser Verschlüsselungsmodul über einen PCI-Bus [5] mit dem System kommunizieren. Dieser Bus-Standard ist für SoC-Systeme weit verbreitet und findet sogar als interner Bus auf einem Chip Anwendung. PCI definiert neben physikalischen (wie den Abmessungen einer Steckverbindung) und elektrischen Parametern (wie Spannungen und Ströme) eine Reihe verschiedener Bus-Zyklen, um über diesen Bus Daten zu übertragen:

- Address-Phase: Da bei einem PCI-Bus Address- und Daten-Leitungen gemultiplext sind, muss vor jedem Datenzugriff eine Adresse in einem abgegrenzten Zyklus übermittelt werden
- Daten-Schreiben
- Daten-Lesen (einzeln)
- Daten-Lesen (Cache-Line): Das Lesen und Schreiben von Daten basiert auf der gemeinsamen Vorstellung beider beteiligter Module von einer aktuellen Adresse; Konsistenzanforderungen beim Caching von Daten erfordern eine spezielle Behandlung, wenn Teile eines gecachten Datenblocks gelesen oder überschrieben werden sollen
- Konfiguration-Lesen
- Konfiguration-Schreiben: Mit Hilfe dieser Zyklen kann auf die Konfiguration eines PCI-Gerätes bzw. dessen Bus-Anschluss zugegriffen werden
- Interrupt-Anfrage: Der PCI-Bus unterstützt ebenfalls die Abwicklung von Unterbrechungen, die bei asynchronen Ereignissen auftreten können

Ein anderes Beispiel für eine Schnittstelle wäre ein I2C-Bus, bei dem die gesamte Kommunikation über

eine einzige serielle Leitung abgewickelt wird. Diese Schnittstelle erlaubt eine besonders kostengünstige Verständigung zwischen mehreren Bausteinen mit geringer Bandbreite, wie dies beispielsweise für manche Consumer-Geräte erforderlich ist.

3 Transaktionsbasierte Schnittstellen (VCI)

Der erste Schritt, der von der physikalischen Schnittstelle von Modulen abstrahiert, besteht darin, die zeitliche Abfolge der elektrischen Signale in sogenannte Transaktionen einzuteilen. Dabei verschiebt sich das Augenmerk von der Abfolge der elektrischen Signale, deren Eigenschaften in Volt oder Nanosekunden definiert sind, auf den Zweck, der mit dem jeweiligen Signal verfolgt werden soll, wie beispielsweise das Lesen oder Schreiben von Daten oder das Auslösen eines Interrupts.

Dies soll bei einem PCI-Bus mittels einer Lese-Transaktion in Bild 4 stark vereinfacht veranschaulicht werden. In Zyklus 1 legt der Initiator der Lese-Transaktion den Befehl „Lesen“ an die Command-Leitungen an, nachdem er über eine nicht dargestellte Arbitrierung den Bus erhalten hat. Anschließend folgt ein Ruhezyklus, bei dem der sogenannte Initiator seinen Bustreiber abschaltet, und das Ziel (Target) in genügendem zeitlichen Abstand seinen Bustreiber einschaltet, sodass kein Kurzschluss entsteht. Danach werden vom Target die angeforderten Daten übertragen. Die Übertragung kann jederzeit angehalten werden, sobald einer der beiden Partner dies mit seinem „Ready“-Signal anzeigt.

Durch das explizite Beschreiben der Transaktionen entsteht innerhalb des Moduls eine weitere Schnittstelle, die es erlaubt, das Modul durch Austausch eines Protokolladapters an beliebigen physikalischen Busprotokollen zu betreiben (siehe Bild 5, linke Seite). Dies stellt eine große Vereinfachung für Anbieter von IP-Komponenten dar.

Zur einheitlichen Beschreibung von Schnittstellen auf der Transaktions-Ebene hat das VSIA-Konsortium den „Virtual Component Interface“-Standard (VCI)

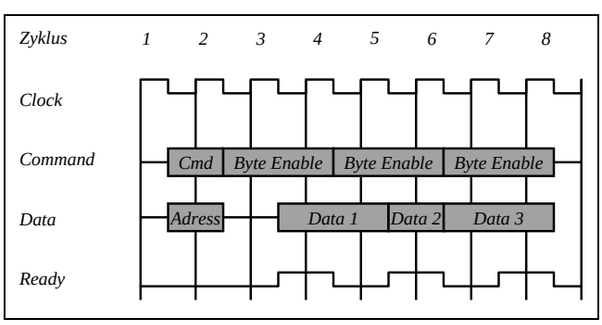

Bild 4: PCI-Lese-Transaktion.

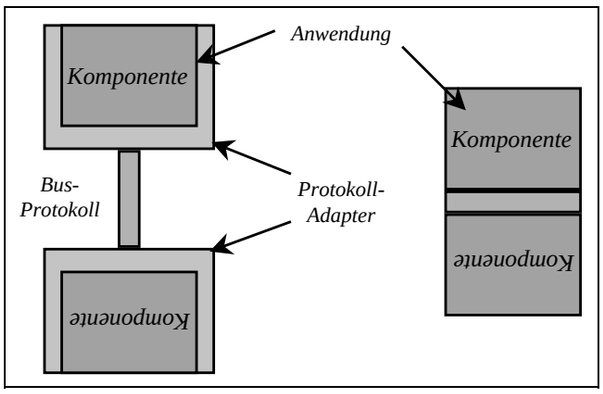

Bild 5: Der Protokolladapter.

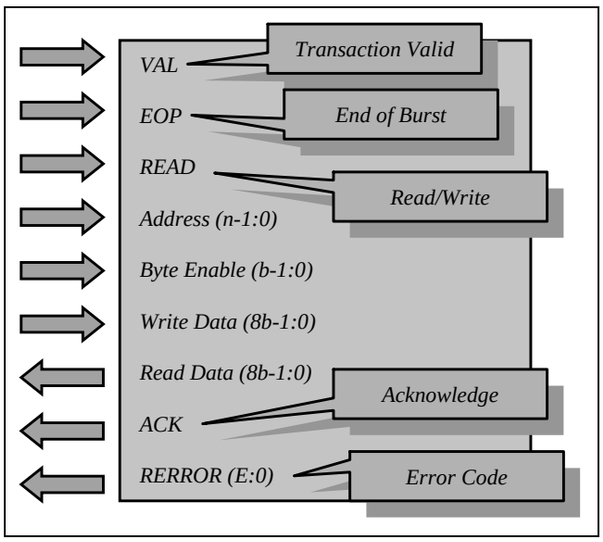

Bild 6: Das On-Chip-Bus-Protokoll.

definiert [1]. Die Abbildung der Transaktionen einer Schnittstelle auf ihre physikalische Implementierung nennt man das Protokoll. Der VCI-Standard definiert außerdem ein neues Protokoll, das On-Chip-Bus- oder OCB-Protokoll [6].

Das OCB-Protokoll legt ein Bus-System, bestehend aus jeweils 32 voneinander getrennten Leitungen zur Übertragung von Adressen, Daten zum Lesen und Daten zum Schreiben fest (siehe Bild 6). Das Bus-Protkoll ist als striktes Punkt-zu-Punkt-Protokoll definiert, d. h. es gibt genau zwei Verbindungspartner, von denen jeweils einer als Initiator und einer als Target designiert ist, d. h. der eine kann nur Transaktionen initiieren, der andere nur darauf reagieren.

Falls allerdings in einem System Protokolladapter und die restliche Komponente zusammen synthetisiert werden, erscheint der physikalische Teil dieses Protokolls nicht notwendigerweise in der Hardware, weil die Synthese diese Schnittstelle durch Optimierung entfallen lassen kann. Eine eingehende Untersuchung zu diesem Optimierungseffekt wurde bislang noch nicht veröffentlicht.

Die Transaktionen ihrerseits werden jeweils durch mindestens eine atomare Aktion auf physikalischer Ebene durchgeführt. Beispielsweise wird eine Schreibtransaktion im VCI-Protokoll durch das Anlegen der Adresse an die Adressleitungen und der Daten an die Datenleitungen in einer einzigen atomaren Transaktion durchgeführt. Beim Schreiben auf einem PCI-Bus hingegen werden Adressen und Daten in getrennten Adress- und Datenzyklen übertragen. Lesezyklen nach der sogenannten *Split-Transaction-Methode* bestehen aus zwei von einander unabhängigen und möglicherweise sogar zeitlich weit auseinander liegenden Aktionen, zwischen denen auch weitere Transaktionen ablaufen können.

Auf einem SoC genügt häufig das einfache Protokoll des VCI-Standards, sodass der Aufwand für ein komplizierteres Protokoll nicht gerechtfertigt ist, wenn zwei Komponenten direkt in einer Punkt-zu-Punkt-Verbindung stehen. In diesem Fall kann ein expliziter Protokolladapter auch weggelassen werden (siehe Bild 5, rechte Seite).

Als Standardwerkzeug zur Erstellung von Protokolladaptoren hat sich das Werkzeug *Protocol Compiler* der Firma Synopsys etabliert. Bei diesem Werkzeug werden Signalverläufe beschrieben und Aktionen angegeben, die durchgeführt werden sollen, wenn ein bestimmter Signalverlauf festgestellt wird. Die Art der Beschreibung ist der von sogenannten *regulären Ausdrücken* [7] entlehnt, allerdings ist die Eingabe nur in grafischer Form möglich.

Bild 7 zeigt beispielhaft die Beschreibung einer Lese-Transaktion über einen PCI-Bus. Diese Transaktion besteht aus der Abfolge einer Adressphase und einer oder mehrerer Datenphasen. Die Adressphase wird durch die Übermittlung eines Adresskommandos über die CMD-Leitungen eingeleitet. Die dabei übermittelten Daten können mittels einer Assign“-Zuweisung als Adresse interpretiert werden. Die nachfolgende Datenübertragung kann stets durch ein geeignetes Signal einer „Ready“-Leitung unterbrochen werden. Falls die Übertragung gelingt, werden die Daten über die Datenleitungen übertragen und zusätzlich auf den CMD-Leitungen sogenannte „Byte-Enables“ beigefügt.

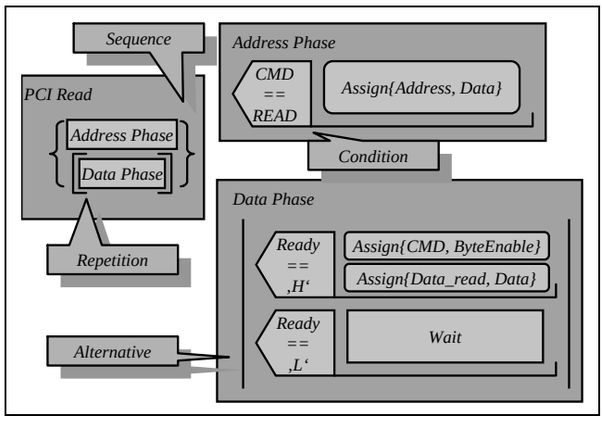

Bild 7: PCI-Lese-Transaktion im Protocol-Compiler.

4 Datenstruktur-Schnittstellen

Da die Komponenten in einem SoC-System meistens an einen eingebetteten Prozessor angeschlossen sind, stellen sie ihre Schnittstelle oft eingeblendet in einen Speicherbereich dar (memory mapped). Die wichtigsten Operationen, die dann von außen her durchgeführt werden, sind Lese- und Schreib-Operationen. Der genaue Transaktions-Typ solcher Operationen (z. B. „Split Transaction“) spielt auf dieser Abstraktionsebene keine Rolle mehr.

Die Herausforderung beim Design von SoC-Systemen ist die Umsetzung einer anwendungsspezifischen Funktionalität auf einen Speicherbereich, der mit Bus-Transaktionen angesprochen werden kann. Zu diesem Zweck lässt sich ein sogenannter funktionaler Adapter [2] einsetzen (siehe Bild 8). Dieser Adapter stellt die anwendungsspezifische Funktionalität einer Komponente als Speicherbild (memory map) dar, auf das mit Bustransaktionen zugegriffen werden kann.

Dieses Speicherbild wird als Datenstruktur beschrieben, die derjenigen einer Programmiersprache ähnelt. Diese Datenstruktur wird mit Operationen annotiert, die wiederum bestimmte Transaktionen auslösen können. Dieses Gesamtgebilde bezeichnet man als *Annotierter Datentyp* (ADT).

Das Grundgerüst eines ADTs bildet ein Datentyp, wie er aus strukturierten Programmiersprachen ge-

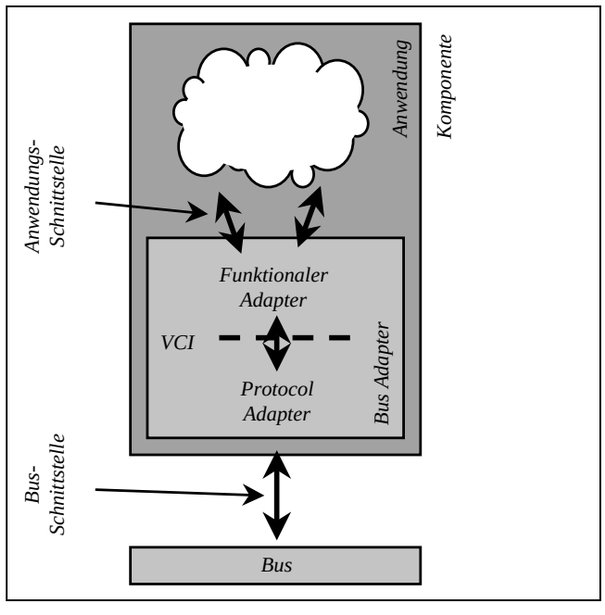

Bild 8: Der funktionale Adapter.

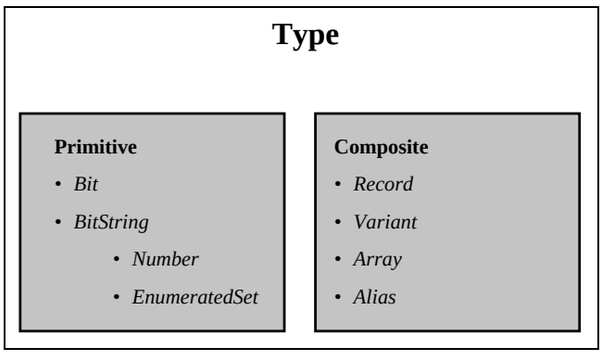

Bild 9: Typen-System der Annotierten Datentypen.

läufig ist [8]. Ein dazugehöriges Typen-System ist in Bild 9 dargestellt.

Eine Datenstruktur besteht aus folgenden Grundbausteinen [2]:

Einfache Datentypen

Dazu gehören nicht weiter unterteilbare Konstrukte wie einzelne Bits oder Bit-Ketten, die als Zahlen oder Elemente einer Aufzählungsmenge interpretiert werden.

Zusammengesetzte Datentypen

Die am weitesten verbreitete Art von zusammengesetztem Datentyp, die dem Begriff der Datenstruktur auch seinen Namen verleiht, ist der *Record*, die Struktur im engeren Sinne. Eine solche Struktur enthält eine Menge von Objekten anderer Datentypen, die jeweils über einen individuellen Namen angesprochen werden können. Während in einem Record alle Elemente gleichzeitig vorhanden sind, wird hingegen in einer sogenannten Variante der gleiche Speicherbereich nur unterschiedlich interpretiert. Eine Reihung enthält jeweils mehrere Elemente eines einzigen Datentyps, die durch eine Indexzahl angesprochen werden. Ferner liefert ein *Alias* (Künstlername) die Möglichkeit, einen bestimmen Datentyp an einer anderen Stelle weiter zu benutzen.

Die Datenstruktur wird nun um Annotierungen erweitert, die die aktiven Eigenschaften des Moduls beschreiben. Im Gegensatz zu einer Software, bei der eine Datenstruktur selbst passiv im Speicher liegt, entwickelt ein Modul-Objekt, das seine Funktionalität in einen Speicherbereich einblendet, ein aktives Verhalten.

Bild 10 zeigt, wie eine solche ADT-Beschreibung für unser Beispiel des DES-Verschlüsselungsmoduls aussehen könnte. In das Grundgerüst der Datenstruktur sind zusätzliche Informationen zur Feineinstellung des Speicher-Layouts eingefügt sowie

```
INTERFACE des_engine IS length=0x100 BEGIN
  Control : RECORD access=w BEGIN
    mode : ENUMSET (encrypt, decrypt);
    reset : BIT pos=16
                action={reset <= 1};
  END.
  Key : RECORD access=rw
              addr=0x10 length=56;
  Status : RECORD access=r BEGIN
   full : BIT action={engine.full};
   empty : BIT action={engine.empty};
  END.
  Input : ARRAY ( 1 TO 0x40 ) OF WORD
    addr=0x80 length=0x40 access=w
    action=Write_Data;
  Output : ARRAY ( 1 TO 0x40 ) OF WORD
    addr=0xC0 length=0x40 access=r
    action=Read_Data;
END.
```

Bild 10: ADT für die DES-Verschlüsselung.

sogenannte Aktionen oder Operationen, die durchgeführt werden, falls ein bestimmter Speicherbereich angesprochen wird. Diese Operationen sind in Bild 10 in geschweifte Klammern gefasst und werden auf RT-Ebene – im Beispiel in VHDL – spezifiziert.

Aus der obigen Beschreibung des ADT einer Modul-Schnittstelle lässt sich ein funktionales Adapter-Modul synthetisieren (siehe Bild 11). Den Kern dieses Moduls bildet ein statischer oder dynamischer Adressdekoder, der die Zuordnung der Adressen zu den ihnen jeweils zugehörigen Operationen vornimmt. Von diesem Adressdekoder aus wird ein Controller angesteuert, der die anwendungsspezifischen Operationen durchführt, die in den Annotierungen spezifiziert sind.

In bestimmten Situationen soll auch das Verhalten eines konventionellen Datentyps teilweise nachgebildet werden. Zu diesem Zweck kann in das Modul auch ein sogenannter Zurücklese-Speicher (*Read-Back-Memory*) synthetisiert werden, aus dem der Zustand des Moduls zurückgelesen werden kann.

Durch den Einsatz eines funktionalen Adapters wird die Erweiterbarkeit und Testbarkeit von Kom-

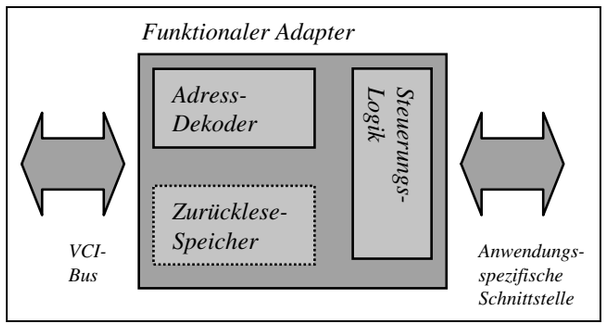

Bild 11: Blockschaltbild eines synthetisierten funktionalen Adapters.

ponenten wesentlich verbessert, wie in [2] gezeigt wurde. Zu den Werkzeugen, die diesen Arbeitsablauf unterstützen, gehören *Innoveda Regent* und *Cadence VCC.*

5 Verifikation

Die abstrakte Darstellung von Modulschnittstellen mittels ADTs erlaubt eine übersichtliche Beschreibung von Testpatterns und deren Wiederverwendung bei Spezifikationsänderungen. Bild 12 zeigt solche zusätzlichen Module, die sich zu Testzwecken synthetisieren lassen.

Eine symbolische Testbench erlaubt die Interpretation von sogenannten symbolischen Testpatterns. Symbolische Testpatterns für das Beispiel sind in Bild 13 links angegeben. Diese Darstellung ist wesentlich einfacher lesbar als binären Pattern (Bild 13, rechts), wie sie physikalisch am Bus erscheinen. Die symbolische Spezifikation der Testpatterns erlaubt in gewissen Grenzen auch eine Weiterverwendung der Testpatterns bei Spezifikationsänderungen.

Ein zweites Modul, das den umgekehrten Weg geht, d. h. das numerische Bitmuster wieder in symbolische Patterns zurückverwandeln kann, ist das sogenannte Dateiausgabe-Modul (File-Dump). Dieses Modul kann den Bitmustern wieder eine Bedeutung in der Memory-Map zuordnen [2].

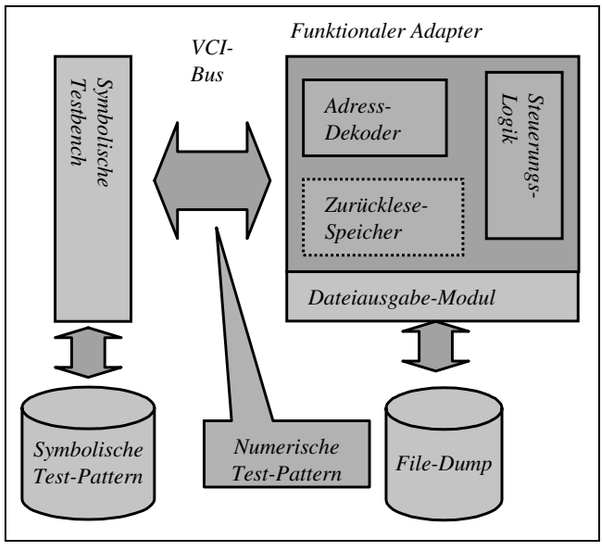

Bild 12: Synthese von symbolischer Testbench und File-Dump-Modul.

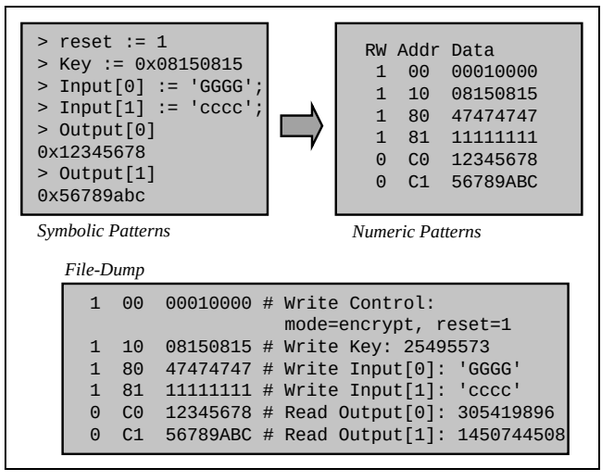

Bild 13: Symbolische vs. numerische Testpatterns.

6 Synthese von Kommunikations-Strukturen

Neben dem Anschluss an einen eingebetteten Prozessor spielen bei komplexer werdenden Systemen auch die Interaktionen zwischen eigenständigen Komponenten eine immer größere Rolle. Auch in solchen Fällen kann man von einer abstrakten Darstellung der Schnittstellen zwischen den einzelnen Komponenten profitieren.

Als besonders typische Kommunikationsmuster im Zusammenspiel von Komponenten haben sich die Pipeline-Struktur sowie die Parallelisierung erwiesen (siehe Bild 14). Bei komplexen SoC-Designs hat sich die Umsetzung und Fehlersuche in solchen Kommunikationsstrukturen häufig als wesentlicher Zeit- und Kostenfaktor herausgestellt, häufig sogar in noch größerem Maße als diejenige der eigentlichen Anwendungsfunktionalität.

Auf Systemebene lassen sich durch Anreicherung der Schnittstellenbeschreibungen um zusätzliche Informationen (siehe [3]) solche Kommunikationsstrukturen synthetisieren. Dadurch lassen sich erhebliche Einsparungen im Entwicklungsaufwand realisieren.

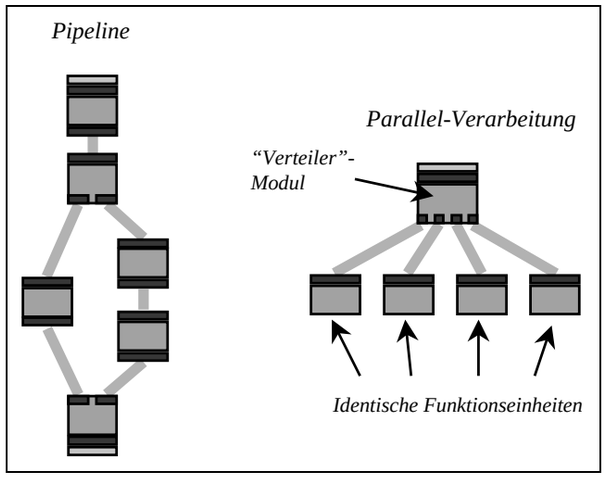

Bild 14: Kommunikationsmuster.

7 Zusammenfassung

In diesem Artikel wurde ein Überblick über Busschnittstellen in SoC-Umgebungen auf verschiedenen Abstraktionsebenen gegeben. Die immer wichtiger werdende Wiederverwendbarkeit von Bausteinen sowie die steigende Komplexität der Systeme, die aus immer mehr Komponenten bestehen, verlangen einen sorgfältigen Umgang mit diesen Schnittstellen. Die Schnittstellen selbst werden immer wichtiger und rücken gegenüber den austauschbaren Anwendungen der jeweiligen Bausteine immer weiter in den Vordergrund.

Danksagung

Vielen Dank an Dr. Caroline Cichon.

Literatur

[1] VSI Alliance, editor: Virtual Component Interface Standard. VSI Alliance 2000.

[2] *Cichon, Gordon:* Annotated Data Type Declarations for Bus Interface Synthesis. In Forum on Design Languages, 2000.

[3] *Cichon, Gordon* and *Brunnbauer, Winthir:* Annotated Data Types for Addressed Token Passing Networks. Design Automation and Test Conference Europe, 2001.[4] *Schneider, Bruce:* Applied Cryptography. John Wiley & Sons, 1995.

[5] *Shanley, T.* and *Anderson, D.:* PCI System Architecture. Addison Wesley, Reading, 1995.

[6] VSI-Alliance: Virtual Component Interface Standard. OCB, On Chip-Bus Standard.

[7] *Hopcroft, E.* and *Ullman, J.:* Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. Oldenbourg Verlag, München, 2000.

[8] *Lensen, Kathleen; Wirth, Nicolaus* and *Mickel, A.:* Pascal User Manual and Report: ISO Pascal Standard. Springer Verlag, 1991.

Dipl.-Inf. Gordon Cichon ist wissenschaftlicher Mitarbeiter an der TU Dresden. Zuvor war er bei Philips und Infineon tätig. Arbeitsfelder umfassen den Entwurf und die Spezifikation von SoC-Systemen für Anwendungen im Bereich Mobilfunk.

Adresse: Tassiloplatz 2, D-81541 München
Tel.: 0174–5441291, E-Mail: gordon@cichon.com