Architektur einer Flexiblen, Wiederverwendbaren Testbench zur Verifikation Paketverarbeitender Hardware in SystemC

Stephan Kubisch, Harald Widiger, Ronald Hecht, Dirk Timmermann, Martin Siemroth

Institut für Angewandte Mikroelektronik und Datentechnik

Universität Rostock, 18051 Rostock

{stephan.kubisch,dirk.timmermann}@uni-rostock.de

Selbst mit den neuesten Modellierungswerkzeugen und dem Einzug von Objektorientierung und Abstraktion in die Hardware-Entwicklung ist der Aufwand für die Verifikation immer noch bestimmend für den gesamten Entwurfsprozess. Wiederverwendbarkeit und Flexibilität sind deshalb aus ökonomischer Sicht unabdingbar. Es wird eine SystemC-Verifikationsumgebung vorgestellt, die sich besonders für paketverarbeitende Hardware und Systeme eignet. Der Fokus liegt dabei auf Wiederverwendbarkeit und leichte Portierbarkeit auf zukünftig zu verifizierende Designs. Weiterhin bietet dieses Framework die Möglichkeit, nicht nur textuelle Hardwarebeschreibungen während der verschiedenen Entwurfsschritte eines Designs zu verifizieren, sondern das Design auch zusätzlich auf der Zielplattform den gleichen Testfällen zu unterziehen, zum Beispiel auf einem Entwicklungsboard.

1 Einleitung

Nicht zuletzt aufgrund der wachsenden Integrationsdichte und Komplexität moderner Schaltkreise und ihrem umfangreichen Repertoire an Funktionen ist die Simulation und Verifikation integrierter Schaltungen und Systeme der zeit- und kostenintensivste Faktor im gesamten Designprozess [1]. Es ist notwendig, die Kosten für die Verifikation eines Systems zu minimieren und die Qualität der Verifikation und deren Handhabung zu verbessern. Verschiedene neue Beschreibungssprachen für Simulations- und Verifikationsumgebungen wurden entwickelt, welche in ihrem Umfang mächtiger sind als reine Hardwarebeschreibungssprachen (*H*ardware *D*escription *L*anguage, HDL) wie VHDL (*V*ery High Speed Integrated Circuit HDL) oder Verilog. Paradigmen aus dem Software-Entwurf wie Objektorientierung (OOP), Überladen von Operatoren und Vererbung vereinfachen auf der einen Seite die Umsetzung komplexer Testfälle und die Handhabbarkeit abstrakter Datentypen.

Auf der anderen Seite bieten sie im Entwurfseinstieg die Möglichkeit abstrakter Systembeschreibungen durch Formen des Transaction Level Modeling (TLM) [2] oberhalb des Register-Transfer-Levels (RTL). Zu diesen Sprachen gehören zum Beispiel SystemVerilog [3], SystemC [4] und der ***e***-Standard [5].

Graphische Simulatoren und Interfaces wie Mentor Graphics' ModelSim [6] und Aldec's Active-HDL [7] bieten darüber hinaus flexible und leistungsstarke Werkzeuge für den automatisierten Umgang mit der Verifikation und Simulation digitaler Systeme. Sie ermöglichen Visualisierung mit Hilfe von Waveform-Betrachtern, Analyse der Code-Abdeckung und Mixed-Language Simulationen. Diese ermöglichen weiterhin Mixed-Level Simulationen zum Beispiel beim schrittweisem Übergang von reinen Verhaltensbeschreibungen zu synthetisierbarem RTL-Code. Trotz dieser neuen Beschreibungssprachen und der hoch entwickelten graphischen Tools wird der Aufwand für die Entwicklung von Test- und Verifikationsumgebungen nicht wesentlich minimiert. Es wird hauptsächlich die Handhabbarkeit verbessert. Die Wiederverwendbarkeit bestehender Komponenten für zukünftig zu testende Systeme und die Flexibilität einer Verifikationsumgebung sind daher maßgebliche Faktoren zur Zeit- und Kostenreduktion. Wiederverwendbarkeit und Flexibilität bedeuten jedoch gleichermaßen eine Generalisierung dieser Komponenten. Dies wiederum erfordert eine sorgfältige Strukturierung der Verifikationsumgebung in für ein zu testendes Modul (Device-under-Test, DUT) spezifische Funktionen und in generelle Funktionen, um den Aufwand bei der Anpassung an zukünftige DUTs möglichst gering zu halten.

In diesem Beitrag stellen wir eine flexible und erweiterbare Verifikationsumgebung auf Basis von SystemC vor. Aufgrund der Ausrichtung unserer Forschungsarbeiten ist der Fokus die Simulation und Verifikation von Baugruppen und Systemen zur Paketverarbeitung (= *P*acket *P*rocessing *M*odule, PPM), wie sie in heutigen und zukünftigen Netzwerken und dem Internet eingesetzt werden. Zum Einen weisen derartige Systeme unabhängig von ihrer konkreten Funktion gewisse Gemeinsamkeiten auf, die in den erwähnten generellen Funktionen der Verifikationsumgebung enthalten sind. Der Aufwand für die Anpassung an spezifische Eigenschaften eines DUTs und für neue Testszenarien wird durch die Nutzung von SystemC-spezifischen Mechanismen und OOP-Methoden auf ein Minimum reduziert. Zum Anderen soll der Horizont herkömmlicher Testbenches und Verifikationsumgebungen bis zur Realisierung auf der jeweiligen Zielplattform ausgeweitet werden; zum Beispiel bis zur Implementierung in einem Application Specific Integrated Circuit (ASIC) oder Field Programmable Gate Array (FPGA). Bisher sind die mit den vorhandenen Beschreibungssprachen entworfenen Testbenches und Verifikationsumgebungen auf die Entwurfsschritte bis zum Post-Place-and-Route HDL-Modell begrenzt und werden nicht über den entsprechenden Simulator hinaus genutzt, zum Beispiel ModelSim. Um dies zu erreichen, wird nicht nur auf die Mechanismen von SystemC zurückgegriffen. Zusätzlich werden Schnittstellen der C++ Bibliothek WinPcap [8] genutzt.

Abschnitt 1.1 und 1.2 stellen Grundlagen zu SystemC und WinPcap dar und erläutern generelle Charakteristika paketverarbeitender Hardware (HW). Abschnitt 2 beschreibt die Struktur der vorgestellten Verifikationsumgebung, deren Eignung für paketverarbeitende HW als auch Vor- und Nachteile dieser Lösung. Abschnitt 3 fast den Beitrag zusammen.

1.1 Die C++-Bibliotheken SystemC und WinPcap

SystemC als mittlerweile von der IEEE standardisierte C++ Klassenbibliothek [4] erweitert die Programmiersprache C++ um Elemente und Methoden zum Beschreiben von HW-Strukturen. Der gesamte Umfang von SystemC ist allerdings nicht komplett synthetisierbar. Nur eine Untermenge kann für Beschreibungen auf RTL-Ebene genutzt

und synthetisiert werden [9]. Weiterhin sind Synthesetools wie von SystemCrafter [10] oder Forte [11] noch Mangelware. Momentan wird SystemC vorwiegend zur Beschreibung von Testbenches und abstrakten Verhaltensmodellen eines Designs genutzt.

Grundlegende Elemente in SystemC sind Module (`sc_module`) und Prozesse (`sc_thread`, `sc_cthread`, `sc_method`); vergleichbar mit Entitäten und Prozessen in VHDL. Zur Inter-Modul-Kommunikation dienen Schnittstellen (Interfaces, `sc_interface`), Anschlüsse (Ports, `sc_port`) und Kanäle (Channels, `sc_prim_channel`). *Interfaces* definieren eine Menge von Funktionen. Es werden jedoch nur deren Signaturen (Name, Rückgabe-Typ, Parameter) und nicht ihre interne Implementierung festgelegt. Interface-Definitionen sind das Schlüsselkonzept in SystemC für eine modulare Systemarchitektur mit hohem Grad an Wiederverwendbarkeit. *Channels* hingegen definieren genau, wie die Funktionen eines Interfaces implementiert sind. Sie können die Funktionen mehrerer Interfaces implementieren und verschiedene Channels können das gleiche Interface in verschiedenen Ausprägungen implementieren. Oft benötigte, allgemeine Channel-Typen sind bereits vordefiniert (`sc_signal`, `sc_fifo`, `sc_clock` etc.). Channels und Module werden mittels *Ports* miteinander verbunden. Diese Struktur ist in Abbildung 1 dargestellt.

Zusätzliche Erweiterungen wie die SystemC Verfication Library (SCV) [12], die TLM Library [2] oder die Master-Slave Library [13] bieten spezielle, zusätzliche Funktionen und vereinfachen die Handhabbarkeit vom Systementwurf bis zur Verifikation.

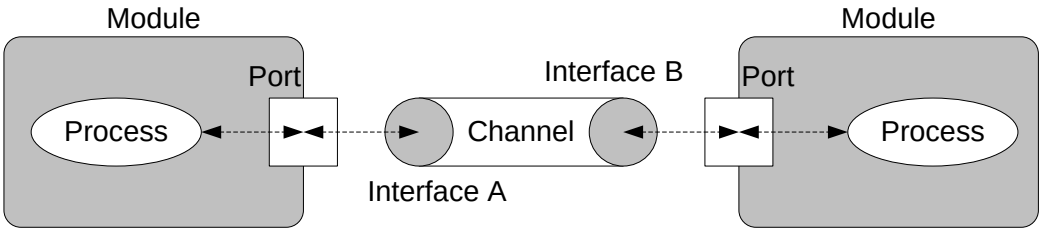

Abbildung 1: Grundkomponenten innerhalb SystemC

WinPcap [8] ist ebenfalls eine frei verfügbare C++ Bibliothek und stellt Funktionen und Schnittstellen zum Versand und Empfang von Nachrichten über die physikalische Netzwerkschnittstelle eines Rechners bereit. Über einen eigenen Treiber wird der Protokollstack des Betriebssystems umgangen und auf Ebene der Sicherungsschicht auf das Netzwerk zugegriffen. WinPcap ist die Windows-Anpassung der UNIX-Schnittstelle libpcap. Aufgrund der leichten Handhabbarkeit und den umfangreichen Funktionen ist WinPcap Kernelement vieler Netzwerktools wie Snort [14] oder Wireshark [15].

1.2 Typische Eigenschaften Paketverarbeitender Systeme

Paketverarbeitung in Software stößt durch wachsende Bandbreiten und Verkehrsvolumen mittlerweile an die Grenzen ihrer Leistungsfähigkeit. Deshalb werden Funktionen der Paketverarbeitung in Netzwerkgeräten zunehmend in HW realisiert und als PPMs zum Beispiel in Breitband-Zugangsnetzwerken eingesetzt, wie sie in [16, 17, 18] vorgestellt sind. HW-Lösungen bieten auch zukünftig genug Potential hinsichtlich der Performance.

Unabhängig von ihrer konkreten Funktion weisen PPMs von außen betrachtet oft ähnliche Strukturen auf. Es existieren Schnittstellen für einen oder mehrere parallele Hauptdatenpfade; zudem oft auf Basis des selben Protokolls. Orthogonal dazu sind Konfigurations- und Steuerinterfaces notwendig. Bezüglich des Hintergunds dieses Beitrags bedeutet „von

außen betrachtet“ aus Sicht der Testbench. Abbildung 2 zeigt die typische Anordnung der Interfaces eines PPMs. Weiterhin lehnen sich alle Arten von Netzwerken an die Spezifikationen des Open Systems Interconnection (OSI) Referenzmodells [19] an. Generell bedeutet dies, dass die Funktionen und Protokolle auf verschiedenen Ebenen gekapselt und hierarchisch angeordnet sind. Die Repräsentation von Informationen erfolgt dabei ebenfalls in gekapselter Form in strukturierten Einheiten wie Frames (Sicherungsschicht), Paketen (Vermittlungsschicht) und Segmenten (Transportschicht). Diese enthalten Protokoll-spezifische Steuerinformationen und zum größten Teil Nutzdaten, welche aus Sicht konkreter PPMs sehr unterschiedlich interpretiert werden.

Diese generellen Eigenschaften (Paketaufbau, Protokoll-Hierarchie, Interfaces) fließen direkt in die Struktur des im Abschnitt 2 beschriebenen Frameworks ein.

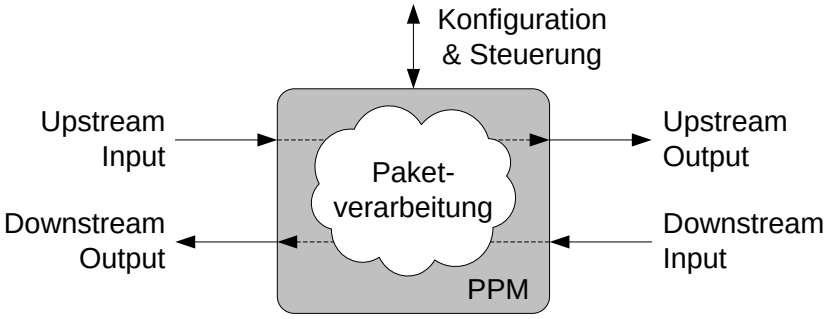

Abbildung 2: Typische Schnittstellen eines PPMs

2 Die SystemC-Verifikationsumgebung

Abbildung 3 zeigt den Aufbau der Verifikationsumgebung. Es wurde die typische Zweiteilung in Testbench und Harness beibehalten [1], wodurch gleichzeitig Daten- und Signalebene separiert sind. Die Testbench ist die Datenebene. Hier werden Stimuli generiert und Antworten des DUT analysiert. In der Harness ist das DUT eingebunden. Der Zugriff auf das DUT wird realisiert und Stimuli aus der Datenebene werden in die Signalebene umgesetzt.

Der Ablauf soll anhand eines Beispiels erläutert werden. Für detaillierte Informationen zur Message-Klasse, zur Umsetzung der Schichten-Hierarchie und zu Bus Functional Models (BFMs) sei auf die folgenden Unterabschnitte verwiesen. Im Generatorprozess sind die Testfälle für das DUT spezifiziert. Dazu soll ein IP-Frame (Internet Protokoll) erzeugt und versendet werden. Der Generator erzeugt ein neues Message-Objekt mit zufälliger Payload und sendet dies über seine Port-Interface-Anbindung an die Vermittlungsschicht, in welcher das Internet Protokoll implementiert ist. Die IP-Funktionalität fügt dem Message-Objekt einen IP-Header hinzu. Die Vermittlungsschicht sendet das IP-Paket weiter an die Sicherungsschicht, wo zusätzlich ein Ethernet-Header hinzugefügt wird. Das IP-Paket ist nun in einem IP-Frame gekapselt. Die Sicherungsschicht leitet dieses Message-Objekt nun aus der Testbench zur Bitübertragungsschicht in die Harness weiter. Die Bitübertragungsschicht wird durch ein BFM realisiert, welches das Message-Objekt als IP-Frame an das DUT sendet. Das DUT selbst verarbeitet den IP-Frame. Ein weiteres BFM erfasst die Antwort des DUTs, wandelt diese in ein neues Message-Objekt um und übergibt es der Sicherungsschicht in der Testbench. Dort wird das Message-Objekt wieder

durch die entsprechenden Schichten weitergeleitet bis zum Monitor-Prozess, welcher die Antwort des DUT entsprechend seiner beabsichtigten Funktion und der Testfälle analysiert. Generator und Monitor können dabei auf eine gemeinsame Datenbasis zugreifen.

Die Harness liegt dabei in zwei Versionen vor. Eine Version implementiert über das BFM den Zugriff auf das HDL-Modell des DUT. Eine zweite Version nutzt die WinPcap Bibliothek und veranlasst das Senden des Message-Objekts über die physikalische Netzwerkschnittstelle des PCs, zum Beispiel zu einem FPGA-Entwicklungsboard. Da die Nutzung der physikalischen Netzwerkschnittstelle innerhalb eines Verifikations-Tools wie ModelSim nicht möglich ist, wurde an dieser Stelle mit Hilfe der Compiler-Direktive `#ifdef MTI_SYSTEMC ... #else ... #endif` eine Compiler-spezifische Teilung in der Verifikationsumgebung vorgenommen. Unter ModelSim (`MTI_SYSTEMC`) wird die erstgenannte Version kompiliert. Unter VisualC++ wird die WinPcap-Version kompiliert. Die Erzeugung der Stimuli und die Analyse der Antworten in der Testbench sind identisch.

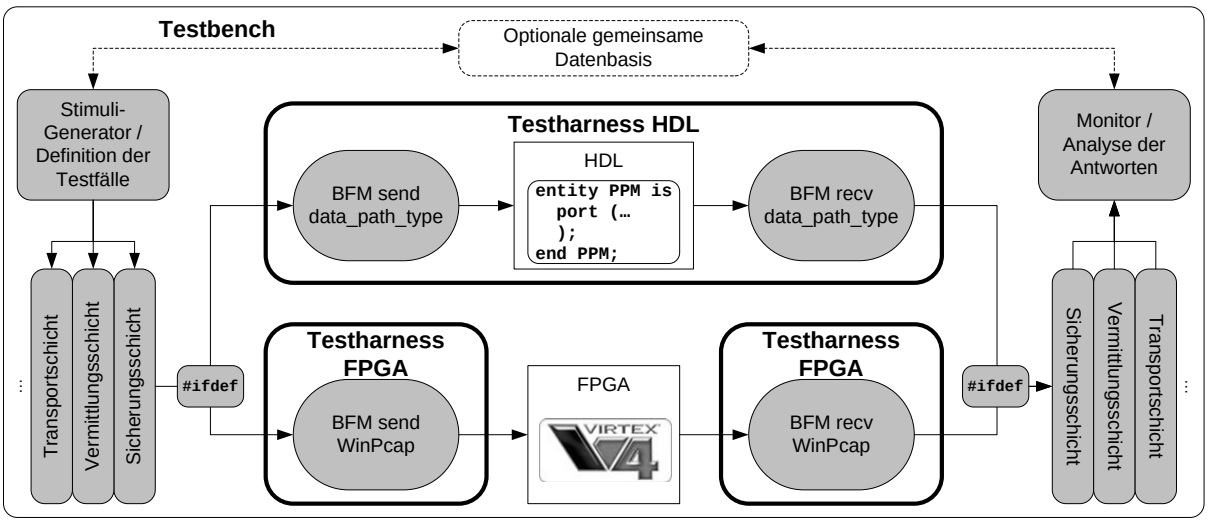

Abbildung 3: Struktur der Testbench

2.1 Die Message-Klasse

Die vorgestellte Verifikationsumgebung bezieht sich auf Systeme und Module, welche innerhalb von Netzwerkszenarien Pakete analysieren, modifizieren oder abhängig von Quelle und Ziel weiterleiten. Die Informationseinheiten (Frames, Pakete, Segmente ...) weisen dabei eine allgemeine Struktur auf, welche durch eine abstrakte Message-Klasse beschrieben ist. Diese Klasse ist weder Protokoll- und Schichten-spezifisch noch ist sie an ein bestimmtes DUT gebunden. Sie kann zur Simulation und Verifikation verschiedener PPMs und ähnlicher Systeme wiederverwendet werden, denn der grundlegende Aufbau einer Informationseinheit in Netzwerken ist weitestgehend gleich: ein vorangestellter, Protokoll-spezifischer Kopf (Header), der Teil der eigentlichen Nutzdaten (Payload) und gegebenfalls noch ein Anhang (Trailer). Die Payload kann wiederum strukturiert sein. In dieser Kapselung spiegelt sich die durch das OSI Modell spezifizierte Schichten-Hierarchie auf Protokollebene in den Informationseinheiten wieder. Ein Objekt der Message-Klasse implementiert dieses Prinzip der Kapselung ohne konkreten Protokollbezug. Ein Message-Objekt stellt einen Container für verschiedenste Informationen zur Verfügung. Weiterhin besitzt es Methoden zur Modifikation dieser Informationen.

Abbildung 4 skizziert die Struktur eines Message-Objektes. Durch Modifikation der Informationen im Message-Objekt können drei verschiedene Bereiche vorhanden sein: für das Objekt allokierter Speicher (schwarz), für Nutzdaten reservierter Speicher (grau) und der Bereich für momentan gültige Daten (weiß). Aufgrund von Lösch- und Einfüge-Operationen kann sich der Bereich gültiger Daten verschieben, liegt aber immer innerhalb des reservierten Bereiches. Die Initialisierung einer neuen Message erfolgt mit Angabe der Größe des zu allokierenden Speichers (`m_size`) und des vorerst zu reservierenden Bereiches (`m_room`). Die Differenz kann später als Puffer dienen. Notwendige Zeiger werden auf Start & Ende des allokierten Speichers (`m_buff`, `m_buffend`) sowie auf Start & Ende der gültigen Informationen (`m_mstart`, `m_mend`) gesetzt. Zu Beginn sind `m_mstart` und `m_mend` identisch, da noch keine gültigen Daten vorhanden sind. Die Message-Klasse bietet nun eine Vielzahl von Methoden (u.a.: `push(int)`, `push(str)`, `pull(int)`, `trim(int)`), um Informationen hinzuzufügen, zu entnehmen oder den Inhalt zu verändern. Weiterhin bietet es Hilfsfunktionen zum Erzeugen zufälliger Nutzdaten oder zur Abfrage privater Objekt-Parameter (u.a.: `int length()`, `random_mssg()`). Für ein konkretes Szenario werden Nachrichten beliebigen Protokolls schichtweise erzeugt und innerhalb der Testfälle nutzbar gemacht. Aufbau und Funktion der Protokolle sind in der in Abschnitt 2.2 vorgestellten Ebenen-Hierarchie implementiert. Trotz der bereits vorhandenen Flexibilität kann die Message-Klasse als Basisklasse vererbt und in angepassten Formen weitergenutzt werden.

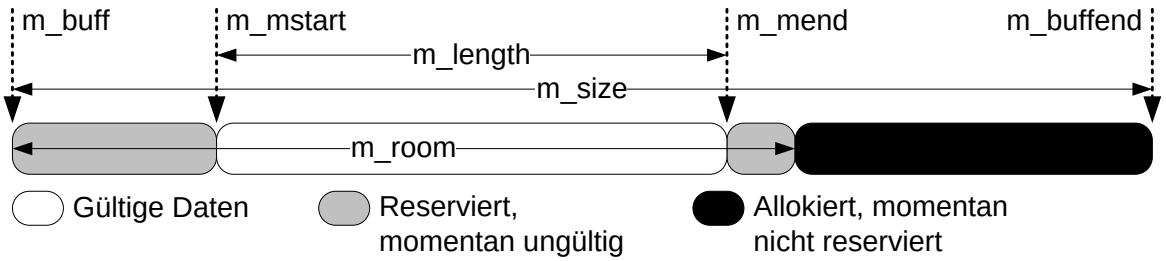

Abbildung 4: Struktur eines Message-Objekts

2.2 Umsetzung der Protokoll-Hierarchie

Zentraler Bestandteil der Verifikationsumgebung ist die Realisierung der Protokoll-Hierarchie und -Kapselung des OSI Referenzmodells. Dabei wurde vor allem auf die Abgrenzung verschiedener Netzwerkaufgaben durch definierte Schichten Wert gelegt. Dies bedeutet *nicht*, dass der komplette TCP/IP-Protokoll-Stack [20] umgesetzt wurde. Vielmehr wurde ein flexibles Gerüst geschaffen, in welchem die typischen Netzwerkschichten als Container für Protokolle dienen. Die Flexibilität besteht in der einfachen Erweiterung dieser Container um benötigte Protokolle.

Abbildung 5 skizziert die hierarchische Anordnung der Protokollschichten. Jede Schicht ist in SystemC als `sc_module` definiert und von den zu dieser Schicht gehörenden Interface-Klassen abgeleitet. Eine Schicht implementiert das konkrete Verhalten der von diesen Interface-Klassen geerbten Funktions-Signaturen. Weiterhin besitzt jede Schicht Ports zu der ihr über- und untergeordneten Schicht. Zum Beispiel kann das Port `dl` eines Objekts der Klasse `network_layer` an das Interface `datalink_up_if` eines Objekts der Klasse `datalink_layer` gebunden werden. Message-Objekte werden über die Interface-Port-Verbindungen ausgetauscht. Enden Protokolle auf einer bestimmten Schicht, so bietet diese Schicht Zugriffspunkte für dieses spezifische Protokoll. Jede Schicht besitzt

zudem Methoden für Schicht-spezifische Attribute. Zum Beispiel kann für ein Objekt der Klasse `network_layer` die IP-Adresse gesetzt und verändert werden. Generator- und Monitor-Objekte können an jede dieser Schichten angeschlossen werden, wenn sie über die entsprechenden Ports und Interfaces verfügen. Die konkrete Realisierung eines Generators oder Monitors ist dabei allerdings abhängig von den jeweiligen Testfällen und vom DUT. Nicht immer werden alle Schichten benötigt. Jede kann als unabhängiger Protokoll-Prozessor genutzt werden.

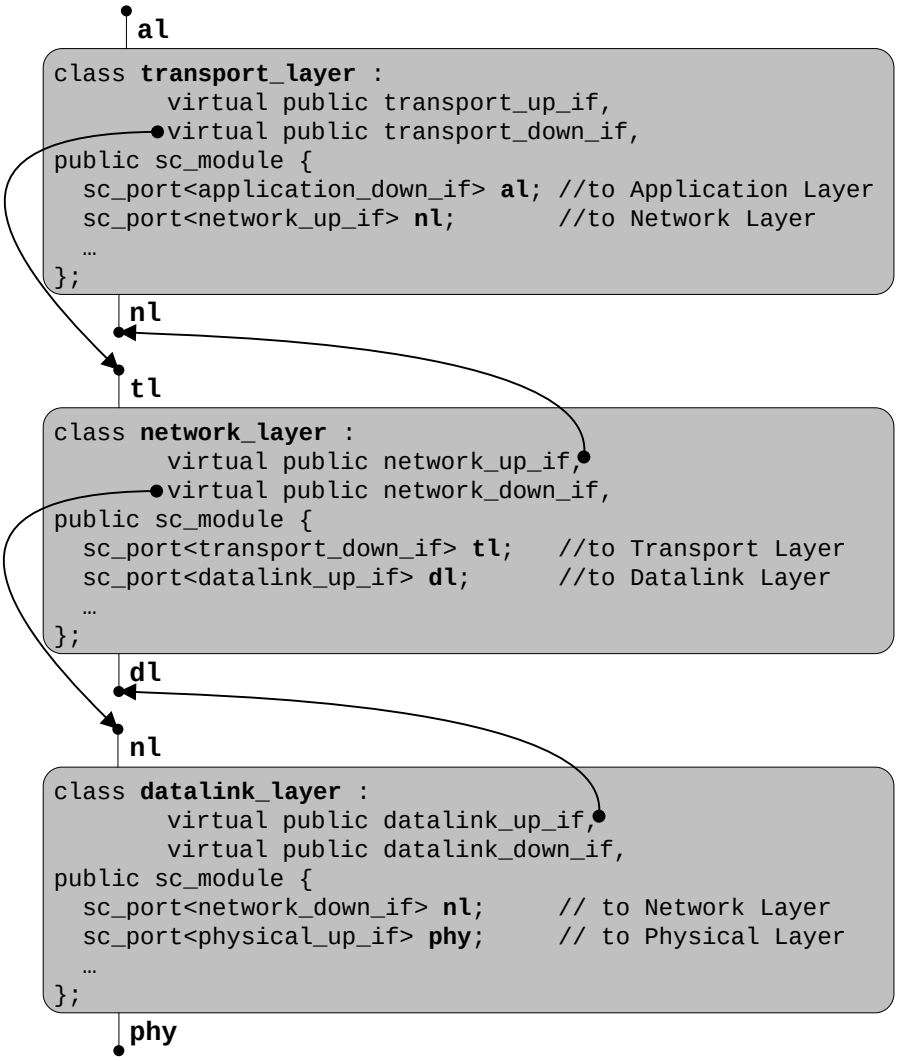

Abbildung 5: Ebenen-Hierarchie mittels Modul-, Port- und Interface-Primitive

2.3 Die Bus Functional Models

BFMs dienen zwei Zwecken. Zum Einen erfolgt in ihnen der Übergang zwischen der Datenebene in die Signalebene. Zum Anderen dienen sie dem Zugriff auf das DUT. Sie implementieren das Kommunikationsprotokoll der Schnittstellen des DUT. In den BFMs werden jedoch keine Stimuli für das DUT, wie zum Beispiel Message-Objekte, erzeugt und auch keine Reaktionen und Antworten des DUT analysiert. In der Testbench erzeugte Stimuli werden einem BFM übergeben. Das BFM legt diese Stimuli an die Schnittstelle des DUT an. Antworten des DUT werden durch ein BFM empfangen und wieder an die

Testbench übergeben. Dafür benötigt ein BFM zwei unterschiedliche Schnittstellen. Zum DUT besitzt es eine von einem Takt abhängige Schnittstelle. Dies ist die Signalebene. Zur übergeordneten Test- und Verifikationsumgebung besitzt es die notwendigen Port- und Interface-Bindungen. Dies ist die Datenebene. BFMs sind spezifisch für ein DUT und für einen hohen Grad an Wiederverwendbarkeit meist modular ausgelegt. Bezogen auf das OSI Referenzmodell stellen sie die Bitübertragungsschicht dar.

Innerhalb der vorgestellten Verifikationsumgebung wurden zwei Klassen von BFMs in SystemC implementiert. Einerseits wurde ein BFM implementiert, welches direkt an das in VHDL beschriebene PPM gebunden wird. Dieses implementiert zum Beispiel den in Abbildung 6a gezeigten einfachen Datenpfad-Typ zum DUT. Der Inhalt eines dem BFM übergebenen Message-Objekts wird byteseriell mit Hilfe der Steuerleitungen Start-of-Frame (SoF) und End-of-Frame (EoF) an das PPM übertragen. Ein ähnliches BFM nimmt die Antworten des PPM entgegen und wandelt diese in neue Message-Objekte um, die an die Verifikationsumgebung übergeben werden.

Die Zweite Klasse von BFMs besitzt zur Datenebene hin die selben Schnittstellen. Es ist jedoch nicht direkt an ein DUT angeschlossen, sondern nutzt Funktionen der WinPcap-Bibliothek, wie Abbildung 6b skizziert. Message-Objekte aus der Verifikationsumgebung werden in entsprechendem Format an die WinPcap-Schnittstelle übergeben. WinPcap ist verantwortlich für die Übertragung dieser Informationen als Ethernet-Frame über die Netzwerkkarte des PCs an ein angeschlossenes FPGA-Entwicklungsboard beziehungsweise für das Erfassen von Antworten des DUTs.

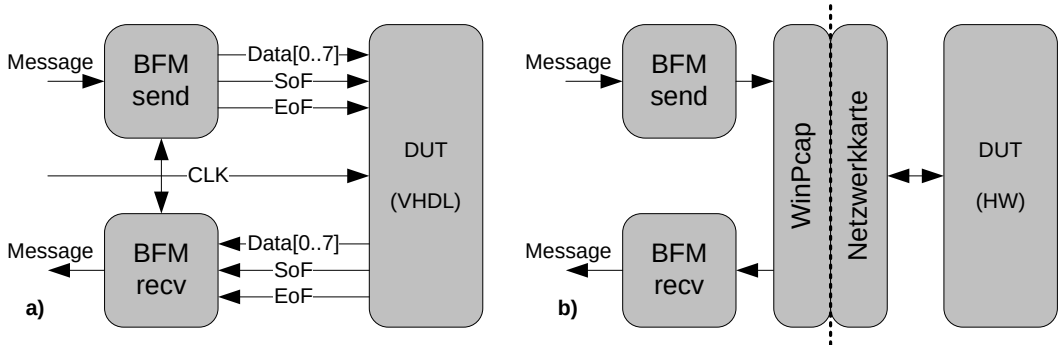

Abbildung 6: Bus Functional Models für ein PPM a) in VHDL b) in Hardware

2.4 Vorteile und Nachteile der Verifikationsumgebung

Während des Entwurfs der Verifikationsumgebung wurde Wert auf *Flexibilität* und *Modularität* gelegt. Durch Container-Klassen für einzelne Protokollschichten und durch die Message-Klasse wurde ein allgemeines Gerüst geschaffen, welches für beliebige Protokolle genutzt und erweitert werden kann. Die Anlehnung an das OSI Referenzmodell erfolgte dabei in der Adaption der Hierarchie und Kapselung und nicht in der Implementierung konkreter Protokolle. Weiterhin sind Daten- und Signal-Ebene durch BFMs voneinander entkoppelt. BFMs sind DUT-abhängig modular austauschbar. Durch die Trennung von allgemeinen und spezifischen Funktionen in der Verifikationsumgebung ist der Aufwand zur Anpassung an ein anderes DUT relativ gering. Zu den notwendigen Modifikationen zählen die Anpassung der Generator- und Monitor-Funktionen zur Modellierung der Testfälle, die Erzeugung spezifischer BFMs für das DUT und gegebenfalls die Erweiterung der Protokoll-

Container um die benötigten Protokolle. In unseren Testszenarien [16, 17, 18] ist vor allem ersteres notwendig, da die Schnittstellen am DUT und die Protokolle weitestgehend identisch sind. Dadurch wird von einem hohen Grad an *Wiederverwendbarkeit* profitiert.

C++ inklusive der SystemC-Erweiterung ist eine bewährte und mächtige Programmiersprache. Methoden der Softwaretechnik lassen sich zur Modellierung von Hardware nutzen und bieten große Vorteile gegenüber den relativ starren HDLs. Gerade komplexe Testfälle lassen sich in C++ einfacher modellieren als mit VHDL (z.B. Umgang mit Strings, Zeigern und Listen). Zudem ist SystemC nicht an HDL-Simulatoren wie ModelSim gebunden. Als eigenständige Konsolenanwendung lassen sich Bibliotheken wie WinPcap nutzen oder Waveforms mittels Trace-Funktionen erzeugen.

Des Weiteren bietet das vorgestellte Framework durch Nutzung der WinPcap-Bibliothek eine zusätzliche Ebene für die Verifikation eines Designs. Bisher war es mit typischen Software-Testbenches ausschließlich möglich, ein System in textueller Form als Verhaltensmodell auf TLM-Ebene, als RTL-Modell, als post-synthesis Modell oder als post-place-and-route Modell zu verifizieren. Die „Reichweite“ der Testbench wurde um die Möglichkeit erweitert, ein System auch auf einer HW-Plattform mit den gleichen Stimuli zu testen, zum Beispiel auf einem FPGA-Entwicklungsboard. Damit ist nun die Verifikation eines DUT *während aller Stadien* des Entwurfs möglich – von der ersten Realisierung der Spezifikation bis zur Implementierung im integrierten Schaltkreis.

Demgegenüber stehen einige Nachteile. Erstens gibt es nach wie vor keinen Compiler, der den gesamten C++-Standard korrekt implementiert. Verschiedene Compiler sind nicht zwingend kompatibel, so dass im C++-Code an einigen Stellen Kompromisse eingegangen werden müssen. Dies ist vor allem problematisch in einem heterogenen Umfeld mit verschiedenen HW und SW Plattformen. Zudem ist es für den Entwickler nach wie vor notwendig, neben C++ auch eine HDL zu beherrschen, da SystemC momentan nur begrenzt synthetisierbar ist. Weiterhin besteht ein gewisser Aufwand beim Entwurf beziehungsweise bei der Anpassung einer Testbench an ein neues DUT, dessen Ausmaß nach oben offen ist. Vorgestellt wurde ein anpassungsfähiges Grundgerüst für eine Verifikationsumgebung.

3 Zusammenfassung

In diesem Beitrag haben wir ein auf SystemC basierendes Gerüst zur Simulation und Verifikation paketverarbeitender Systeme vorgestellt. Durch die Trennung von spezifischen und generellen Funktionen in der Verifikationsumgebung konnte ein hohes Maß an Flexibilität und Wiederverwendbarkeit erreicht werden. Sowohl das Schichten-Prinzip des OSI Referenzmodells als auch die allgemeine Nachrichten-basierte Repräsentation von Informationen in Netzwerken wurde in SystemC umgesetzt. Der Aufwand zur Anpassung an künftige zu testende PPMs wurde auf das Notwendigste reduziert. Zudem können Systeme mit dem vorgestellten Framework auch auf ihrer Zielplattform den gleichen Tests unterzogen werden, wodurch im gesamten Entwurfsprozess mit der gleichen Verifikationsumgebung gearbeitet werden kann. Die Anwendungsszenarien sind jedoch nicht nur auf die reine Paketverarbeitung beschränkt. Systeme auf Grundlage Nachrichten-basierter Kommunikation (Networks-on-Chip & Systems-on-Chip) oder auch Datenstrom-basierte Systeme können nach einigen Anpassungen simuliert und verifiziert werden.

Diese Arbeiten entstanden durch Unterstützung von und in Zusammenarbeit mit Siemens Networks GmbH & Co. KG, Greifswald, Deutschland.

Literatur

[1] Janick Bergeron. *Writing Testbenches: Functional Verification of HDL Models, Second Edition.* Kluwer Academic Publishers, ISBN: 1-4020-7401-8, 2003.

[2] Stuart Swan. A Tutorial Introduction to the SystemC TLM Standard, Cadence Design Systems, Inc., März 2006.

[3] IEEE Computer Society. 1800 – IEEE Standard for System Verilog, ISBN: 0-7381-4811-3.

[4] IEEE Computer Society. 1666 – IEEE Standard SystemC Language Reference Manual, ISBN: 0-7381-4871-7.

[5] IEEE Computer Society. 1647 – IEEE Standard for the Functional Verification Language '***e***', ISBN: 0-7381-4940-3.

[6] Mentor Graphics, ModelSim. http://www.model.com.

[7] Aldec Active-HDL. http://www.aldec.com.

[8] WinPcap: The Windows Packet Capture Library. http://www.winpcap.org.

[9] Synthesis Working Group of Open SystemC Initiative. SystemC Synthesizable Subset, Draft 1.1.18, Dezember 2004.

[10] SystemCrafter SC. http://www.systemcrafter.com.

[11] Forte Design Systems. http://www.forteds.com.

[12] SystemC Verification Working Group. SystemC Verification Standard Specification, Mai 2003.

[13] OSCI. Master-Slave Communication Library – A SystemC Standard Library, http://www.systemc.org, 2002.

[14] Snort – The de facto Standard for Intrusion Detection/Prevention. http://www.snort.org.

[15] Wireshark: The World's most Popular Network Protocol Analyzer. http://www.wireshark.org.

[16] S. Kubisch, H. Widiger, D. Duchow, D. Timmermann, and T. Bahls. Wirespeed MAC Address Translation and Traffic Management in Access Networks. In *Proc. of the World Telecommunications Congress 2006 (WTC06) on CD-Rom*, Budapest, Ungarn, April 30 - Mai 3 2006.

[17] H. Widiger, S. Kubisch, D. Duchow, D. Timmermann, and T. Bahls. A Simplified, Cost-Effective MPLS Labeling Architecture for Access Networks. In *Proc. of the World Telecommunications Congress 2006 (WTC06) on CD-Rom*, Budapest, Ungarn, April 30 - Mai 3 2006.

[18] H. Widiger, S. Kubisch, D. Timmermann, and T. Bahls. An integrated Hardware Solution for MAT, MPLS-UNI, and TM in Access Networks. In *Proc. of the 31st Annual IEEE Conf. on Local Computer Networks (LCN) (in press)*, Tampa, Florida, USA, November 14 - 16 2006.

[19] ISO/IEC. International Standard 7498 – OSI - Basic Reference Model: The Basic Model, 1994.

[20] Douglas E. Comer. *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture (4th Edition).* Prentice Hall, ISBN: 0-13-018380-6, 2000.