Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Recherchebericht AW1

Framework für den SoC Entwurf

Armin Jeyrani Mamegani

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Framework für den SoC Entwurf

Armin Jeyrani Mamegani

Recherchebericht AW1
im Studiengang Informatik Master
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer: Prof. Dr. Ing. Bernd Schwarz

Abgegeben am 25. Juli 2008

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Anders als bei der reinen Softwareentwicklung können eingebettete Systeme je nach Anforderung sowohl dem Entwurf von Software als auch der Hardware unterliegen. Dies ist darauf zurückzuführen, dass die Ausführungszeit einer Aufgabe als reine Softwareimplementierung womöglich den Echtzeitanforderungen nicht genügt. Die Leistung der in eingebetteten Systemen eingesetzten Prozessoren ist aufgrund von Dimensions-, Kosten- und Energiebeschränkungen geringer, als die eines im PC-Segment verwendeten Prozessors. Daher ist der Einsatz von dedizierten Hardwarekomponenten unumgänglich, um die Echtzeitanforderungen zu erfüllen. Diese sind schneller in der Ausführung und ermöglichen zudem eine parallele Abarbeitung einer Aufgabe und belasten nicht den Hauptprozessor eines Systems. Den Entwurf solcher Systeme nennt man Hardware/Software Co-design.

Die Geschwindigkeitsvorteile einer Hardwarerealisierung gegenüber einer Softwarerealisierung wird in [13] verdeutlicht. Eine CRC-32-Prüfsummenbildung wurde als Software und Hardware implementiert und die Ausführungszeiten miteinander verglichen. Die Softwarerealisierung wurde auf einem 32-Bit Xilinx Microblaze Prozessor ausgeführt. Die Hardwarelösung wurde in VHDL für ein Spartan3 FPGA von Xilinx realisiert. Das Ergebnis ist, dass die Hardwarerealisierung eine etwa 80-fach schnellere Lösung des CRC-32-Algorithmus darstellt.

Mobile Viedeoanwendungen (Videokonforenz, Videotelefonie, usw.) erfordern komplexe Videokompressionsverfahren. Die Ausführung solcher Verfahren als Software führt zu einer starken Belastung eines mobilen Prozessors. Zusätzlich verhindern die geringe Dimension und Bauweise mobiler Endgeräte den Einsatz von zusätzlichen Hardwarekomponenten, wie Grafik- oder Decoderkarten. Eine Lösung hierfür ist die Integration dieser Hardwarefunktionen auf denselben Baustein, auf dem sich auch der Hauptprozessor befindet. Solche Systeme werden als System on Chip(SoC) bezeichnet. Abbildung 1.1 zeigt anhand einer Mobiltelefon-Plattform die Verteilung von Funktionen eines solchen Systems und die schematische Zusammenführung.

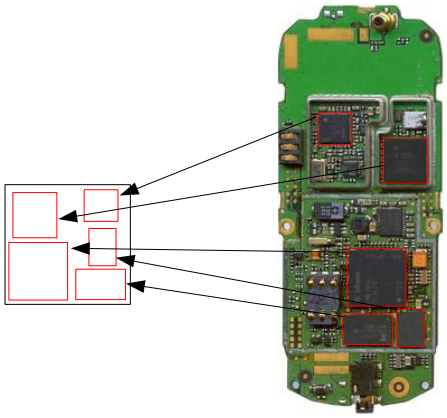

Abbildung 1.1: Rechts: Funktionen des Systems verteilt auf verschiedenen Bausteinen, Links: Alle Funktionen(Prozessor, Speicher, GSM-Modul,...) auf einem Chip

Die Realisierung von SoCs kann auf FPGAs oder ASICs erfolgen. FPGAs sind konfigurierbare Logikschaltkreise. Die Konfiguration kann mit einer Hardwarebeschreibungssprache wie VHDL durchgeführt werden [6]. Im Gegensatz zu ASICs (festverdrahtete Schaltungen) können FPGAs mehrmals rekonfiguriert werden. Aufgrund ihrer hohen Rechenleistung und der Flexibilität ermögicht der Einsatz von FPGAs die Verwendung einer einzigen Hardware-Plattform für verschiedene Aufgaben [22].

Diese Arbeit konzentriert sich auf „System on a programmable Chip“. Basis dieser Systeme bildet ein FPGA. Das FPGA wird mit einem Softcore-Prozessor, einem Bussystem, Speicher und speziellen Hardwarebeschleunigern konfiguriert(Abbildung 1.2). Diese Komponenten sind als sogenannte IPs(Intellectual Property) in verschiedenen IP-Bibliotheken verfügbar. Es werden bereits erprobte Komponenten wiederverwendet und zusätzlich anwendungsspezifische Hardware neu entwickelt.

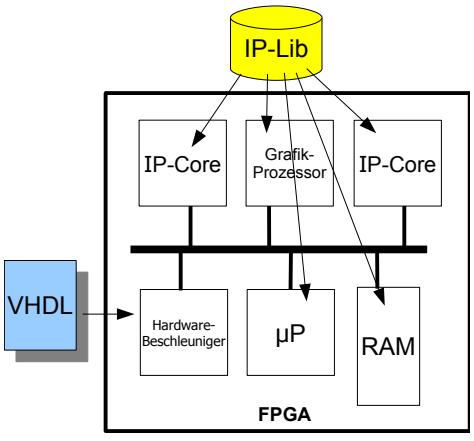

Abbildung 1.2: SoC-System auf einem FPGA

Der Einsatz eines SoC-Systems hat den Vorteil, dass die Performanz der Hardware und die Flexibilität der Software bezüglich der schnellen Anpassung an neue Anforderungen ausgenutzt wird. Jedoch steigen die Anforderungen an SoC-Systeme stetig und erhöhen damit die Komplexität. Die Handhabung dieser Komplexität und die kürzer werdende Entwicklungszeit(Time-To-Market) fordern eine neue Designmethodik für SoC-Systeme. Im Kapitel „**Ziele dieser Recherche**“ wird eine Recherchezusammenfassung zu einer Designmethodik (ESL) gegeben, das den aktuellen Stand im SoC-Entwurfsablauf darstellt.

1.2 Ziele dieser Recherche

Die Softwareentwicklung und die Hardwareentwicklung sind beim klassischen Entwurf von SoC-Systemen zwei verschiedene Prozesse. Abbildung 1.3 stellt die einzelnen Schritte bei beiden Vorgehensweisen dar.

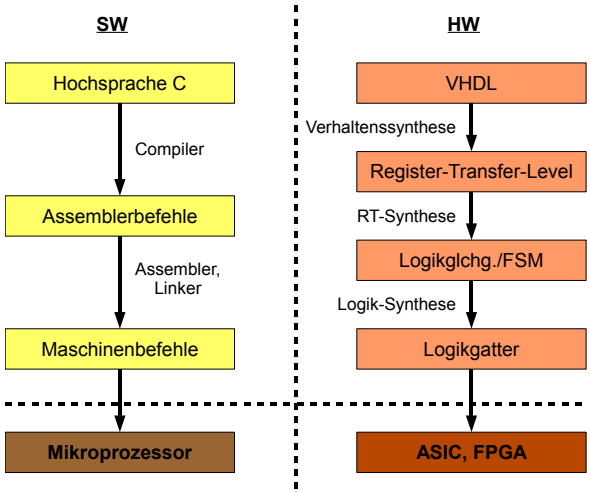

Abbildung 1.3: Entwurfsschritte beim Hardware- und Software-Entwurf

Bei steigender Komplexität eines solchen Systems ist die Notwendigkeit einer einheitlichen Sicht auf das System gefordert [27]. Dazu wird ein Entwurfsablauf benötigt, der die Schritte von dieser Sicht aus bis zur endgültigen Implementierung beschreibt [1]. Electronic System-Level Design(ESL) definiert eine Methodik für den Entwurf eines SoCs. In [1] wird ESL folgendermaßen definiert: *„ESL ist die Anwendung geeigneter Abstraktionen, um die Übersicht über ein System zu verbessern und die Wahrscheinlichkeit einer erfolgreichen Implementierung der Funktionalität kosteneffizient zu erhöhen ohne dabei die bestehenden Einschränkungen zu verletzen“*.

Traditionell wird in der Elektronikentwicklung die Erstellung von Hardwarebeschreibungscode(VHDL/Verilog) per direkter Codeeingabe durchgeführt [1]. Dies ist aber meist mühsamer als die Erstellung von grafischen Modellelementen. In der reinen Softwareentwicklung ist die automatische Code-Generierung mittels UML bereits weit verbreitet [9]. Werkzeuge wie Matlab/Simulink ermöglichen die automatische C-Code-Generierung für Mikroprozessoren, als auch die Generierung von VHDL-Code für FPGAs [6]. Das langfristige Ziel ist es, mit einer einzigen Modellierungssprache ein System zu entwickeln und mit einem Werkzeug automatisch Code für Hard- und Software zu generieren. Der Entwickler soll also das System mit einer abstrakten Sprache modellieren. Ein Werkzeug soll dabei die Aufteilung in Hardware und Software durchführen [15] und den passenden Code erzeugen. Abbildung 1.4 stellt dies schematisch dar.

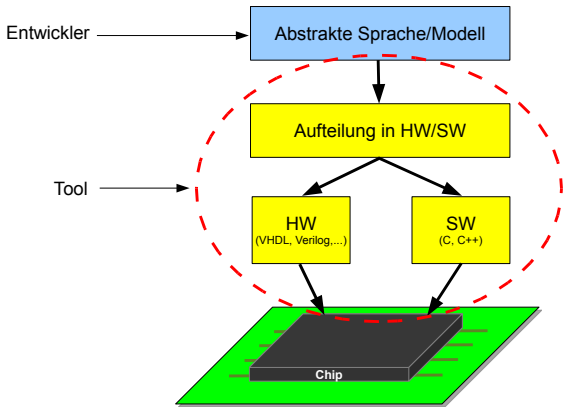

Abbildung 1.4: Aufteilung des Designprozesses zwischen Entwickler und Tool

Ziel dieser Arbeit ist es, die ersten Rechercheergebnisse zum ESL-Design aufzuzeigen. In Kapitel 2 wird zunächst ein Überblick über den Entwurfsablauf mit ESL gegeben. In diesem Kapitel wird auch kurz auf die Hardware/Software-Partitionierung eingegangen. Für den ESL-Designschritt Spezifikation und Modellierung wird in Kapitel 3.1 SysML als eine Sprache zur Systemmodellierung vorgestellt. Der Vorteil von SysML ist die Unabhängigkeit von der Disziplin. Somit soll die Nutzbarkeit von SysML für Systeme bestehend aus Hard- und Software gezeigt werden. In Kapitel 3.2 wird als zweite Modellierungssprache das UML Profile for SoC vorgestellt. Als eine Zwischensprache, die Simulations- und Analysemöglichkeiten bietet, wird SystemC in Kapitel 4 herangeführt. In Kapitel 5 wird eine Zusammenfassung und ein Ausblick gegeben. Hierzu wird als Ausblick in Bezug auf das langfristige Ziel beim ESL-Design, nämlich die Automatisierung des Entwurfsablaufs, auf Ziele für AW2 und Seminar sowie der Erprobung von den hier vorgestellten Mechanismen im Projekt eingegangen.

2 Electronic System-Level Design

Ausgehend davon, dass Designabsichten im menschlichen Gedanken entstehen, definiert ESL ein top-down Entwurfsablauf, das in sechs Schritte unterteilt ist. Dabei ist der traditionelle isolierte Entwurf von Hardware und Software mit ESL nicht mehr möglich. Viel mehr wird die parallele Entwicklung angestrebt. Die einzelnen Schritte sind: 1. Spezifikation und Modellierung, 2. Analyse(vor der Partitionierung), 3. Partitionierung, 4. Analyse(nach der Partitionierung) und Debugging, 5. Verifikation, 6. HW/SW Implementierung. [1]

Während der „Spezifikation und Modellierung“ werden Dokumente erstellt, die die Absichten des Systems oder des Produktes und die Einschränkungen beschreiben. Auf der obersten Ebene werden diese Spezifikationen und Anforderungen in einer natürlichen Sprache definiert. Dabei muss diese Sprache die Handhabung von Mehrdeutigkeit bezüglich der Implementierung unterstützen, damit in den nächsten Verfeinerungsschritten auf funktionale sowie nicht-funktionale Anforderungen eingegangen werden kann, ohne diese mit konkreten Implementierungen beschränken zu müssen. Zusätzlich zu dieser natürlichen Spezifikationssprache gibt es einen Satz an ausführbaren Spezifikationssprachen. Diese sind unter anderem MATLAB M-Code, SystemC, SystemVerilog, SDL, UML, XML und BlueSpec. Das Spezifizieren von modernen, komplexen Systemen ist mit einer grafischen Modellierungssprache für einen Menschen einfacher zu Handhaben, als mit Code. Daher wird in Kapitel 3 für diesen Schritt auf Erweiterungen von UML eingegangen.

In der pre-partitionalen Analysephase wird auf Grundlage der Spezifikation die korrekte Funktion des Systems analysiert. In diesem Schritt werden die Systemspezifikation verfeinert, sowie zusätzliche und ausführlichere Einschränkungen an das System definiert. Obwohl eine statische Analyse nützlich ist, gibt eine dynamische Analyse basierend auf Spezifikationen und Modellen ein besseres Verständnis über Details und Einschränkungen des Systems her. UML Modelle sind teilweise zwar auch ausführbar, aber dennoch eingeschränkt in ihrer Analysefähigkeit. SystemC als Zwischensprache dient hier als ein Modell des Systems, das simuliert und analysiert werden kann. Dem OMG Ansatz der Modell-getriebenen Entwicklung(MDA) folgend ist eine Transformation eines UML Modells in SystemC-Code und die Anreicherung dieses Codes mit weiteren Informationen eine Beschleunigung dieser beiden ESL-Schritte [17]. Hierzu wird SystemC in Kapitel 4 vorgestellt.

Die Ausführung einer Applikation auf einem Mikroprozessor genügt in einigen Fällen nicht den Performanz- und Echtzeitanforderungen. Somit ist die Zuordnung von Teilen dieser Ap-

plikation auf dedizierte Hardware nötig. In diesem Schritt des ESL-Designs wird das System in Hard- und Software aufgeteilt. Dieser Schritt wird HW/SW-Partitionierung genannt. HW/SW-Partitionierung ist ein Prozess der Aufteilung einer Applikation in Mikroprozessor („Software“) und mehrerer spezieller HW-Elemente, um eine Implementierung zu erreichen, die die oben genannten Anforderungen am Besten erfüllt [7]. D.h. es findet eine sukzessive Zuordnung von Funktionen des Systems zu Hardware bzw. Software statt. Eine Funktion des Systems wird dabei entweder als Hardware realisiert oder in Form von Software auf einem Prozessor ausgeführt. Während ein HW-Element eine Funktion des Systems erfüllt, können mehrere Funktionen als Software auf einem Prozessor ausgeführt werden (Abbildung 2.1).

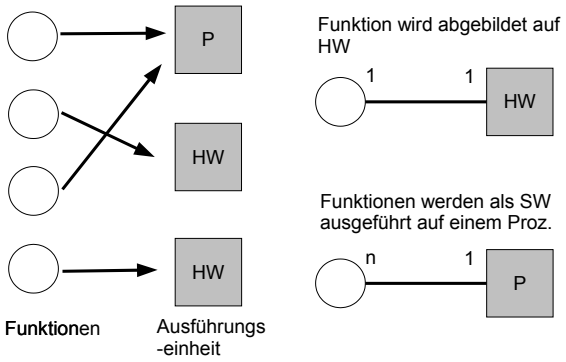

Abbildung 2.1: links: Schematische Darstellung der Zuordnung von Funktionen zu Ausführungseinheiten, rechts: Zuordnungsbeschränkung

In [25] wird eine sequenzielle Partitionierung vorgestellt. Hierbei wird entweder softwareorientiert oder hardwareorientiert partitioniert. Bei der softwareorientierten Partitionierung werden alle Funktionen zunächst auf Software abgebildet. Es findet dann eine Umlagerung von Funktionen in Hardware statt, wenn diese Funktion als Software den Echtzeit- und Performanzanforderungen nicht genügt. Bei der hardwareorientierten Partitionierung werden die Applikation komplett als Hardware realisiert und nach und nach in Software umgelagert, wenn die HW-Ausführung zu komplex ist und damit zu viele Ressourcen verbraucht oder eine SW-Realisierung den Anforderungen genügt. In [15] wird das Tool „COOL“ gezeigt, das anhand der ganzzahligen Programmierung eine automatische Partitionierung durchführt.

Im Analyseschritt nach der Partitionierung, wird die Wirkung der Partitionierung geprüft. Im fünften Schritt muss eine Verifikation auf das nun aufgeteilte System durchgeführt werden. Im letzten Schritt werden konkrete Hard- und Softwareimplementierungen produziert, die zum Einsatz kommen sollen. Diese letzten Schritte gehören nicht zum Umfang dieser Arbeit und bieten Recherchegrundlage für die Veranstaltung AW2.

3 Modellierung

3.1 SysML

In der reinen Softwareentwicklung hat sich UML als graphische Modellierungssprache bereits bewährt und ist zum Standard geworden. Die Komplexität eines Systems lässt sich durch die Verwendung eines UML-Modells besser handhaben, da ein UML-Modell für Menschen einfacher zu verstehen ist als Code [1]. Durch die XMI-Fähigkeit (XML Metamodel Interchange) bietet UML eine Toolportabilität und ist durch die Stereotypen erweiterbar. Eine Erweiterung von UML um die Sichten des Systemengineerings ist SysML [27]. UML ist für die SW-Entwicklung geschaffen worden [9]. Es sind Erweiterungen für die Modellierung von Hardware erforderlich. SysML 1.0 wurde im Mai 2006 von der OMG und der INCOSE spezifiziert. SysML soll zur Spezifikation, Analyse, Design, Verifikation und Validation von komplexen Systemen dienen [19].

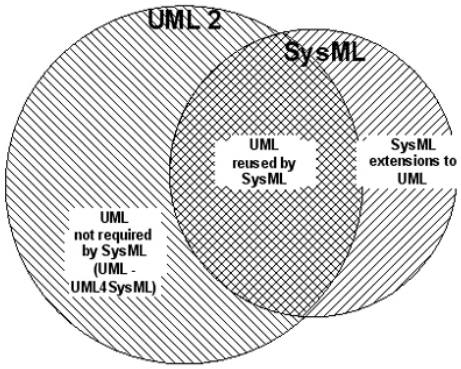

Abbildung 3.1: SysML Untermenge und Erweiterung [20]

Wie in Abbildung 3.1 dargestellt, bildet SysML eine Untermenge von UML 2. Zudem definiert SysML noch einige Erweiterungen. Die Verwendbarkeit von SysML bezüglich SoC-Design wird hier vorgestellt. Auf die einzelnen Erneuerungen von SysML wird in [27] und [9] näher eingegangen.

Als eine Erneuerung von SysML gegenüber UML ermöglichen Requirement Diagramme die Spezifikation von funktionalen sowie nicht-funktionalen Anforderungen. Für den ESL-Entwurf hat dies den Vorteil, dass Anforderungen in Bezug zu Systemelementen gesetzt werden können. Die Rückführung einer Funktion des Systems und somit die Verifikation ist hiermit geboten [9].

Aufgrund ihrer hohen Verknüpfung mit der Objektorientierten Softwareentwicklung wird bei SysML auf die Verwendung von Klassen verzichtet. Anstatt dessen kommen Systembausteine, so genannte **Blöcke** zum Einsatz. Mit diesen werden die Elemente des Systems modelliert. Damit lässt sich die Struktur eines Systems beschreiben. Hierzu wird die Information, um welche Art Element es sich handelt (SW oder HW), weggelassen. Mit dem **Assembly** Konstrukt (**Internes Blockdiagramm**) bietet SysML die Möglichkeit, Hardware und Software-basierte Komponenten oder auch analoge und digitale Elemente zu modellieren. Assembly ist eine stereotypisierte Klasse, die zur Beschreibung eines Systems aus miteinander verbundenen Teilen dient. In einem Assembly Modell wird zusätzlich der Informationsfluss zwischen den einzelnen Komponenten des Systems dargestellt. Als weitere neue Eigenschaft können hier auch **kontinuierliche Informationsflüsse** modelliert werden. SysML bietet einen Mechanismus, Elemente unterschiedlicher Modellbereiche miteinander zu verbinden. Die so genannte **Zuteilung** wird dazu verwendet, ein Verhalten auf eine Struktur abzubilden. Mit UML 2 kann die Kontrolle über die Ausführung von Aktivitäten von außen nicht beeinflusst werden. Durch den Einsatz von kontinuierlichen Aktivitäten in SoC-Systemen ist eine Kontrolle dieser Aktivitäten von außen gefordert. Hierzu erweitert SysML die Kontrollmechanismen für Aktivitätsdiagramme um den **Kontrolloperator**. Dieser ermöglicht das Starten bzw. Enden von Aktionen unabhängig vom derzeitigen Ausführungsstand der Aktivität. Obwohl SysML Elemente definiert, die eine Modellierung bezüglich Hardware nicht einschränken, behebt es nicht das alte Problem von UML, der mehrdeutigen Interpretation. SysML führt auch keine erweiterte Semantik ein. Bei der Modellierung mit SysML gilt ähnlich wie in UML, je höher der Abstraktionsgrad, desto schwächer die Semantik. Zusätzlich ist SysML wie auch UML kein Verfahren für das Vorgehen beim Entwurf eines SoC. Vielmehr kann SysML in der Spezifikations- und Modellierungsphase des ESL-Entwurfablaufs verwendet werden. Aufgrund der Analysebeschränkungen von SysML ist die Umsetzung in SystemC-Code die eigentliche Herausforderung dieser Ebene. Ein Tool ist gefordert, das eine automatische Umsetzung des SysML-Modells in ausführbaren Code übernimmt, um dem sich immer stärker verbreitenden MDA-Ansatz der OMG zu folgen. Dabei sei auf Kapitel 4 verwiesen, wo auf SystemC als Zwischensprache eingegangen wird.

3.2 UML Profile for System on a Chip

Die Entstehung des UML Profile for SoC (UML4SoC) ist ähnlich wie bei SysML darauf zurückzuführen, dass UML 2 zu softwarelastig ist und ihre Konstrukte keine ausreichende Beschreibung von SoC-Elementen bieten. Aufgrund dessen ist ein neuer Satz von Stereotypen definiert worden, der ein direktes Abbild von SoC-Elementen darstellt. Diese Stereotypen basieren auf UML Basiskonstrukten und wurden 2006 von der OMG und der USoCF (UML for System on Chip Forum) als „UML Profile for System on a Chip“ in der Version 1.0 spezifiziert. UML4SoC ist unabhängig von einer Hardware-Beschreibungs-Sprache. Dennoch ist UML4SoC gezielt auf die Konzeption von SystemC (siehe Kapitel 4) ausgerichtet, da sich dessen Abstraktionslevel und die Beschreibungselemente auf Systemebene bewährt haben.

Das Modul ist der Hauptbestandteil eines SoC-Modells. Module enthalten weitere Module, Prozesse, die das Verhalten eines Moduls beschreiben (insbesondere Parallelität), Kanäle für die Kommunikation mit anderen Modulen und Ports, als Schnittstellen. Kanäle sind von Modul abgeleitet und können demnach dieselben Elemente enthalten. Eine wesentliche Erneuerung sind Clock- und Reset-Ports. Diese ermöglichen die Darstellung der in der Hardwareentwicklung erforderlichen Clock- und Reset-Signale.

Mit Data wird der Typ der Information zwischen den Modulen bzw. Kanälen festgelegt. Diese Datenabhängigkeit kann auch über Templates definiert werden (siehe Abbildung 3.2). Damit wird eine generische Datenabhängigkeit ermöglicht.

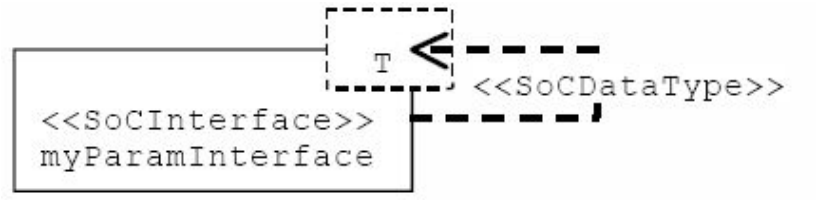

Abbildung 3.2: Template Darstellung in UML4SoC

Die Ausrichtung der UML4SoC Elemente zu SystemC hat den Vorteil, dass eine Transformation eines UML4SoC Modells in SystemC-Code mit Hilfe eines Tools realisierbar wäre. Jedoch gab es zur Zeit der Entstehung dieser Arbeit noch kein etabliertes Werkzeug. In [14] wird ein Framework vorgestellt, mit dem bereits anhand von Klassen- und State-Machine-Diagrammen der UML 2 automatisch SystemC-Code generiert wird.

Es ist gezeigt worden, dass UML4SoC eine praktikable Möglichkeit ist, zumindest kleinere SoC-Systeme zu modellieren und in SystemC-Code umzusetzen [21].

4 SystemC

SystemC ist keine Sprache. Es ist eine Klassenbibliothek der bereits etablierten Programmiersprache C++. Diese Klassenbibliothek ermöglicht mit dem Rest des C++ Sprachraums die Modellierung und Simulation von komplexen elektronischen Systemen. SystemC ist open source und damit kostenfrei erhältlich. 2005 wurde SystemC von der IEEE standartisiert. Bei der Hardwarebeschreibung wird mit SystemC auf höheren Abstraktionsebenen modelliert, sodass bei der Simulation eine hohe Geschwindigkeit erreicht wird und die auf der Hardware auszuführende Software mitsimuliert werden kann. Um typische Eigenschaften von Hardware beschreiben zu können, erweitert SystemC die Sprache C++ um weitere Makros und Funktionen. Hauptelemente von SystemC sind Module(SC_MODULE), die nach außen über Ports zugänglich sind und wiederum Module enthalten können. Mit Prozessen(SC_METHOD, SC_THREAD) wird das Verhalten von Modulen beschrieben. Mit Prozessen wird in SystemC auch Parallelität realisiert. [4]

Mit SystemC ist es also möglich, ein System bestehend aus Hardware und Software zu beschreiben [4]. Zusätzlich ist SystemC mit C++ eine ausführbare und simulierbare Sprache. Die Verwendung von SystemC als die oberste Modellierungssprache für ein System ist ungeeignet [1]. SystemC besteht aus Programmcode. Ein Vorteil einer grafischen Modellierungssprache ist die bessere Verständlichkeit für die Entwickler. Zusätzlich möchte man im Modell die Beziehung zwischen Anforderungen und den Systemelementen darstellen. Dies ist mit SystemC nicht möglich. Darüber hinaus sind in der anfänglichen Designphase konkrete Implementierungsdetails eher unwesentlich und eher beschränkend. Diese müssten bei der Modellierung in SystemC aber berücksichtigt werden. Die Verwendung einer grafischen Modellierungssprache in der frühen Entwicklung und die automatische Umsetzung in ein ausführungs- und simulierfähigen SystemC-Code vereinfacht und beschleunigt somit die Entwicklung eines Systems bestehend aus Hardware und Software [1].

In [14] wurde die automatische Umsetzung eines UML 2 Modells in SystemC-Code durch ein Framework realisiert. Zusätzlich kann mit dem Werkzeug SystemCrafter SC die Simulation von Hardware- und Software-Partitionen basierend auf SystemC-Beschreibungen durchgeführt werden. Der Hardware-Teil des Systems wird vom SystemCrafter SC in synthetisierbaren VHDL- oder Verilog-Code übersetzt. Der Software-Teil wird als C++-Code von einem passenden Compiler übersezt [24].

5 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurde eine Methodik für den SoC-Entwurf vorgestellt. Zunächst wurden einige Ansätze im Bereich des Hardware/Software-Codesign untersucht. Verschiedene Paper (siehe Literaturverzeichnis) wurden studiert, um einen Ansatzpunkt zu erhalten. Hierzu wurden Grundlagen zum Ablauf beim ESL-Entwurf recherchiert und zusammengetragen, da dies den state-of-the-art beim Entwurf von SoCs darstellt [1]. Es wurde auf die ersten 3 Schritte beim ESL-Design näher eingegangen. Für die Spezifikations- und Modellierungsphase des ESL-Design-Ablaufs wurde die Nutzbarkeit von grafischen Modellierungssprachen für den SoC-Entwurf erläutert. SysML und UML4SoC wurden als Erweiterungen der UML hinsichtlich der Systemmodellierung präsentiert. Der Bezug von UML4SoC und SystemC wurde erklärt. SystemC wurde als eine analyse- und simulationsfähige Zwischensprache eingeführt, das für den zweiten Schritt im ESL-Design nützlich sein kann. Für den letzten recherchierten Schritt im ESL-Design wurde ein kurzer Überblick beim Vorgehen bei der Partitionierung eines Systems in Hard- und Software gegeben.

Für die Veranstaltungen AW2 und Seminar stellen die restlichen Schritte des ESL-Designs eine Recherchegrundlage dar. Zusätzlich ist die detailiertere Betrachtung der vorgestellten Schritte hinsichtlich Automatisierung und Werkzeuge ein weiterer Arbeitspunkt. In dieser Arbeit wurden die Grundlagen des ESL-Designs recherchiert. Die vorgestellten Modellierungssprachen müssen auf ihrer Anwendbarkeit geprüft werden. Diese Verifikation kann in der Veranstaltung Projekt erfolgen. Dabei ist eine Anwendung im FAUST-Kontext, wie z.B. dem Carolo-Cup Fahrzeug bzw. der Spezifikation einer Plattform für das Carolo-Cup Fahrzeug zu wählen.

Das Feld der dynamisch rekonfigurierbaren Systeme bietet zusätzliches Recherchebedürfnis. Im Hinblick auf die steigende Anzahl an mobilen Multimediaanwendungen, besteht die Idee einer zur Laufzeit rekonfigurierbaren Plattform. Dabei soll hier für echtzeitkritische Anwendungen (H.264 Dekodierung, Videokonferenz) ein rekonfigurierbarer Baustein mit einem Hardware-Modul dynamisch geladen werden. Diese Idee kann bereits im Projekt begonnen und bis zur Masterarbeit möglicherweise generalisiert werden. Nützlich wäre hierbei die Zusammenarbeit mit Personen aus den Projekten um Google Android. Damit könnten direkt aktuelle Anwendungen auf dieser Plattform erprobt werden.

Literaturverzeichnis

[1] Brian Bailey, Grant Martin, and Andrew Piziali. *ESL Design and Verification*. Morgan Kaufmann, 2007.

[2] Massimo Baleani, Frank Gennari, Yunjian Jiang, Yatish Patel, Robert K. Brayton, and Alberto Sangiovanni Vincentelli. Hw/sw partitioning and code generation of embedded control applications on a reconfigurable architecture platform. Technical report, University of California, Berkeley.

[3] Carsten Bieser. *Konzept einer bibliotheksbasiert konfigurierbaren Hardware-Testeinrichtung für eingebettete elektronische Systeme*. PhD thesis, Universität Fridericana Karlsruhe, Juli 2007.

[4] David C. Black and Jack Donovan. *SystemC: From The Ground Up*. Springer, 2004.

[5] Aske Brekling, Michael R. Hansen, and Jan Madsen. A timed-automata semantics for a system-level mpsoc model. Paper, 2006.

[6] Ralf Gessler and Thomas Mahr. *Hardware-Software-Codesign*. vieweg, 2007.

[7] Scott Hauck and Andre DeHon. *Reconfigurable Computing*. Morgan Kaufmann, 2008.

[8] B. Knerr, M. Holzer, and M. Rupp. Hw/sw partitioning using high level metrics. Technical report, Vienna University of Technology Institute for Communications and RF Engineering, 2004.

[9] Andreas Korff. *Modellierung von eingebetteten Systemen mit UML und SysML*. Spektrum, 2008.

[10] Lobna Kriaa, Aimen Bouchhima, Wassim Youssef, Frederic Petrot, Anne-Marie Fouillart, and Ahmed Jerraya. Service based component design approach for flexible hardware/software interface modeling. Paper, 2006.

[11] Thomas Mader. Design methodology for embedded systems. Technical report, Institut für Informatik, Universität Innsbruck, 2006.

[12] Shankar Mahadevan, Michael Storgaard, Jan Madsen, and Kashif Virk. Arts: A system-level framework for modeling mpsoc components and analysis of their causality. Technical report, Technical University of Denmark, 2005.

[13] Ramin Jeyrani Mameghani. Hw-sw-codesign zur fehlerfreien Übertragung von bild- und steuerdaten zwischen einer fpga-basierten zeilenkamera und einem pc. Studienarbeit an der HAW-Hamburg im Studiengang Elektrotechnik, 2008.

[14] Grant Martin and Wolfgang Müller. *UML for SoC Design*. Springer, 2005.

[15] Peter Marwedel. *Eingebettete Systeme*. Springer, 2007.

[16] Daniel Mattsson and Marcus Christensson. Evaluation of synthesizable cpu cores. Master's thesis, Chalmers Universitäy of Technology, 2004.

[17] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren. Uml for esl design - basic principles, tools, and applications. Paper, 2006.

[18] OMG. Omg uml profile for system on a chip(soc). OMG Available Specification v1.0, 06 2006.

[19] OMG. Omg systems modeling language(omg sysml). OMG Available Specification v1.0, 09 2007.

[20] Jörn Priebe. Uml vs sysml. URL: http://www.arrogant.info/serendipity/archives/223-UML-vs-SysML.html, 2007.

[21] Maha Ramanan. Soc, uml and mda - an investigation. URL: http://jerry.c-lab.de/uml-soc, 2006.

[22] Chris Rowen. *Engineering the Complex SoC*. Pearson Education, 2004.

[23] Lesley Shannon and Paul Chow. Simppl: An adaptable soc framework using a programmable controller ip interface to facilitate design reuse. Paper, 2007.

[24] SystemCrafter. Systemcrafter sc: Technology. Artikel, 2006.

[25] Jürgen Teich. *Digitale Hardware/Software-Systeme*. Springer, 2007.

[26] Yves Vanderperren and Wim Dehaene. Sysml and systems engineering applied to uml-based soc design. Technical report, Katholieke Universiteit Leuven, EE Department (ESAT-MICAS), 2005.

[27] Tim Weilkiens. *Systems Engineering mit SysML/UML*. dpunkt, 2006.

[28] Wayne Wolf. *High Performance Embedded Computing*. Morgan Kaufmann, 2007.

Abbildungsverzeichnis