Modellierung von digitalen Systemen mit SystemC

Von der RTL- zur Transaction-Level-Modellierung

von
Prof. Dr.-Ing. Frank Kesel

Oldenbourg Verlag München

Prof. Dr.-Ing. Frank Kesel ist seit 1999 Professor für Integrierte Schaltungstechnik an der Hochschule Pforzheim und vertritt dort das Thema Entwurf von digitalen Systemen. Zuvor war er zehn Jahre in der Entwicklung von integrierten Schaltungen bei Philips Semiconductors und der Robert Bosch GmbH tätig.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-verlag.de

Lektorat: Dr. Gerhard Pappert, Johannes Breimeier
Herstellung: Constanze Müller
Einbandgestaltung: hauser lacour
Gesamtherstellung: freiburger graphische betriebe GmbH & Co. KG, Freiburg

Dieses Papier ist alterungsbeständig nach DIN/ISO 9706.

ISBN 978-3-486-70581-2
eISBN 978-3-486-71895-9

Vorwort

Die stetig wachsende Komplexität von integrierten Schaltungen, welche als ASICs, FPGAs oder „Off-The-Shelf"-Standardprodukte entwickelt werden, erfordert auch eine entsprechende Weiterentwicklung der Entwurfsmethodiken und -verfahren, um integrierte Schaltungen in annehmbarer Zeit und mit guter Qualität entwickeln zu können. War in den neunziger Jahren der Entwurf von digitalen Schaltungen auf Register-Transfer-Ebene mit Hardwarebeschreibungssprachen wie VHDL oder Verilog vorherrschend, so steht nun die Entwicklung von kompletten Rechnersystemen auf einem Chip im Fokus. Hierzu ist es insbesondere notwendig, dass solche Systeme simuliert werden können, bevor noch das erste Silizium des Chips vorliegt. Dabei möchte man in der Lage sein, auch die Ausführung von Programmen bis hin zur Ausführung von Betriebssystemen auf dem so genannten „virtuellen Prototyp" des Rechnersystems simulieren zu können. Ferner möchte man auch die Architektur des Systems schnell ändern können, um beispielsweise Optimierungen in der Aufteilung der Anwendung auf Hardware und Software vorzunehmen. Register-Transfer-Modelle des Rechnersystems sind für solche Zwecke zu aufwändig und können keine ausreichende Simulationsleistung erzielen. Bereits in den neunziger Jahre wurde daher versucht, Programmiersprachen wie C oder C++ für die Modellierung der Hardware einzusetzen. SystemC, welches ebenfalls auf C++ beruht, hat sich in den letzten zehn Jahren zu einer der wesentlichen Sprachen für die Modellierung von Systemen entwickelt. Die starke Unterstützung der EDA- und Halbleiterfirmen sowie die Normierung durch den IEEE haben dafür gesorgt. Gegenüber VHDL oder Verilog ist es mit SystemC möglich, die Systeme in der Modellierung stärker zu abstrahieren – beispielsweise auf der System- oder Transaktionsebene –, um damit Änderungen schneller vornehmen und Modelle mit erheblich höherer Simulationsperformanz entwickeln zu können.

Um SystemC erfolgreich für die Modellierung einsetzen zu können, wird eine entsprechende Schulung erforderlich sein; wobei es hier, neben C++ und der SystemC-Bibliothek, insbesondere auch um ein Verständnis der gegenüber der RTL-Modellierung stärkeren Abstraktion auf Transaktionsebene und der hierfür notwendigen Mechanismen geht. Auch an den Hochschulen und Universitäten sollte SystemC Eingang in die Curricula von Studiengängen finden. Ich konnte in den letzten Jahren Erfahrungen mit SystemC in Projekten mit der Industrie und beim Einsatz in der Lehre im Rahmen eines Master-Studiengangs sammeln. Ein wesentliches Problem in der Lehre war es dabei, dass es derzeit zum Thema SystemC nur wenige Lehrbücher gibt, die insbesondere auch den aktuellen Stand von SystemC im Hinblick auf die Modellierung auf der Transaktionsebene darstellen. So war ich der Ansicht, dass es sich lohnen würde, ein Lehrbuch über SystemC zu schreiben, welches für die Lehre an Hochschulen und Universitäten aber auch für die Schulung von Praktikern in der Industrie eingesetzt werden kann.

Der Inhalt des vorliegenden Buches deckt im Prinzip die im IEEE Standard 1666-2011 beschriebene SystemC-Bibliothek sowie die ebenfalls dort beschriebene TLM-2.0-Bibliothek für die Modellierung auf Transaktionsebene ab. Da es sich beim SystemC-Standard selbst um ein Dokument von etwas mehr als 600 Seiten handelt, war mir bei der Konzeption des Buches klar,

dass es nicht das Ziel des Buches sein kann – bei beschränktem Umfang –, den Standard und die damit verbundenen C++-Bibliotheken tatsächlich vollständig darstellen zu können. Ich habe mich daher bewusst darauf beschränkt, im Sinne eines Lehrbuchs die wesentlichen Zusammenhänge zu vermitteln und diese durch viele Quellcode-Beispiele auch zu illustrieren. Damit soll dem Lernenden eine solide Grundlage für eigene SystemC-Projekte vermittelt werden.

Das Konzept des Buches folgt der Art und Weise, wie ich SystemC auch in meinen Vorlesungen an der Hochschule oder in Seminaren vermittle: Ich gehe davon aus, dass ein größerer Teil der Leser schon Kenntnisse im Hardware-Entwurf mit „klassischen“ Hardwarebeschreibungssprachen auf Register-Transfer-Ebene hat. Daher wollte ich zunächst zeigen, dass man mit SystemC auch Hardware auf RT-Ebene modellieren kann und in SystemC auch die gleichen Mechanismen wie in VHDL oder Verilog vorhanden sind – beispielsweise nebenläufige Prozesse oder Signale. Somit können sich vermutlich die meisten Leser auf vertrautem Terrain bewegen. Anschließend wollte ich verschiedene Mechanismen einführen, wie beispielsweise die Interfaces mit ihren Ports und Methoden, die über die Mechanismen von VHDL oder Verilog hinausgehen und erst eine abstraktere Modellierung ermöglichen. Es erschien mir auch wichtig, den SystemC-Simulationsalgorithmus genauer zu besprechen, um dem Leser ein tieferes Verständnis für die Simulation von nebenläufigen Systemen zu vermitteln. Schließlich wollte ich die Möglichkeiten der Modellierung auf Transaktionsebene mit der SystemC-TLM-Bibliothek zeigen. Im Zusammenhang mit SystemC und dem Entwurfsablauf hätte man natürlich noch weitere Aspekte beleuchten können, wie beispielsweise die Verifikation oder die Hardware-Synthese. Dies hätte allerdings das Konzept und den Rahmen des Buches gesprengt, so dass ich darauf verzichtet habe. Zum Thema „Electronic System Level Design“ sind auch eine Reihe von Büchern auf dem Markt verfügbar, welche den gesamten Entwurf von elektronischen Systemen beschreiben. Das vorliegende Buch fokussiert auf SystemC und die „Transaction-Level“-Modellierung.

Für die Qualität eines Buches ist die Durchsicht von kritischen Rezensenten wichtig. An dieser Stelle möchte ich mich bei einigen Personen aus der Industrie und dem Hochschulbereich bedanken, welche die Mühe auf sich genommen haben, das Buch Korrektur zu lesen, und die mir wertvolle Hinweise zur Verbesserung des Buches geben konnten. Namentlich erwähnt seien hier insbesondere Joan Drenth, Manuel Gaiser, Prof. Dr. Joachim Gerlach, Dr. Christian Sebeke sowie Dr. Martin Vaupel. Selbstverständlich gilt auch dem Oldenbourg Verlag mein besonderer Dank für die Möglichkeit, dieses Buch zu veröffentlichen. Gedankt sei auch den Studierenden der Hochschule Pforzheim, welche durch Diskussionen und Anregungen im Rahmen von Vorlesungen, Laboren sowie Projekt- und Abschlussarbeiten auch zum Buch beigetragen haben. Nicht zuletzt gebührt auch meiner Familie mein Dank für das Verständnis und die Unterstützung

Bei der ersten Auflage eines neuen Buches schleichen sich trotz Sorgfalt und Korrekturlesens meistens doch noch Fehler ein. Hierfür möchte ich mich im Voraus entschuldigen. Für entsprechende Hinweise und auch andere Anmerkungen zum Buch bin ich immer dankbar. Senden Sie diese am besten per E-Mail an mich (frank.kesel@hs-pforzheim.de). Zum Schluss wünsche ich Ihnen Freude beim Lesen des Buches und hoffe, Sie für die Arbeit mit SystemC motivieren zu können.

Frank Kesel
Pforzheim, im Juli 2012

Inhaltsverzeichnis

1 Einleitung

Dieses erste Kapitel soll dazu dienen, dem Leser eine Einordnung von SystemC im Entwurf von integrierten elektronischen Systemen zu ermöglichen. Wir beschreiben zunächst, was man unter integrierten elektronischen Systemen versteht und gehen anschließend darauf ein, welche Funktion die Abstraktion im Entwurf der mikroelektronischen Hardware erfüllt. Anschließend skizzieren wir den Ablauf im Entwurf elektronischer Systeme und beleuchten die Rolle von SystemC, wobei wir auch auf die historische Entwicklung von SystemC eingehen. Einer der wesentlichen Vorteile von SystemC gegenüber traditionellen Hardwarebeschreibungssprachen wie VHDL oder Verilog ist die Möglichkeit, durch abstrakte Modellierung zu erheblich leistungsfähigeren Simulationsmodellen zu kommen. Wir werden daher in diesem Kapitel definieren, wie man die Leistung von Simulationsmodellen messen und beurteilen kann. Des Weiteren wird die Modellierung auf der so genannten Transaktionsebene ein Schwerpunkt des Buches sein; diese beschäftigt sich inbesondere mit der Modellierung von Bussystemen. Wir möchten daher beispielhaft in diesem Kapitel ein typisches Bussystem für integrierte elektronische Systeme vorstellen, um dem Leser eine Motivation für die Modellierung auf Transaktionsebene zu geben. Das Kapitel schließt dann mit einer Übersicht über den weiteren Aufbau des Buches.

1.1 Integrierte Elektronische Systeme

Integrierte Elektronische Systeme bestehen aus mikroelektronischer Hardware und dazugehöriger Software. Die Hardware wird in Form von integrierten Schaltungen realisiert, die als ICs (für „Integrated Circuit“) oder umgangssprachlich als „Chips“ bezeichnet werden. Werden ICs anwendungsspezifisch realisiert, so sind dies so genannte ASICs (Application-Specific Integrated Circuit). Häufig werden ein oder mehrere Mikroprozessoren, Speichermodule sowie verschiedene Peripherieeinheiten auf dem Chip integriert. Diese Komponenten werden über Bussysteme verbunden; Abbildung 1.1 zeigt ein beispielhaftes System. Man spricht bei einer solchen Systemintegration auf einem Chip von einem „System-on-Chip“ (SOC, auch „System-on-a-Chip“). Im Kern handelt es sich um ein digitales Rechnersystem, wobei aber auch analoge Schaltungen, wie beispielsweise Analog-Digital-Wandler, auf dem Chip integriert werden können. Ferner befinden sich in der Regel noch Schaltungen für die Takterzeugung und die Spannungsversorgung sowie für die Implementierung von Energiesparmodi auf dem Chip. Eingesetzt werden SOCs insbesondere auch in eingebetteten Systemen (engl.: embedded systems) – also Rechnersysteme, die in einem technischen System arbeiten, aber für den Benutzer nicht in Erscheinung treten –, die ihre Anwendung in Geräten wie Smartphones, DVD-Playern, Tablet-PCs, Haushaltsgeräten aber auch in der Automobiltechnik, der Medizintechnik oder in Maschinensteuerungen finden.

SOCs können eine enorme Komplexität aufweisen: Ein modernes Smartphone mit dem auf Linux basierenden Android-Betriebssystem verfügt über einen leistungsfähigen Doppelkern-

Prozessor, beispielsweise ein ARM Cortex-A9 mit 1.2 GHz Taktfrequenz [5], einen Grafikprozessor zur Ansteuerung des Touchscreen-Displays und über eine Vielzahl von Kommunikationsschnittstellen wie UMTS/GSM, WLAN, Bluetooth und USB. Neben dem Cache-Speicher des Prozessors, welcher sich auf dem Chip befindet, wird an das SOC externer DRAM- und Flash-Speicher im Umfang von einigen Gigabyte angeschlossen. Zusätzlich zum Internet-Zugang über WLAN oder UMTS/GSM sind in der Regel eine oder zwei Digitalkameras mit Videofunktion und ein MP3-Player zu bedienen. Darüber hinaus sind eine Vielzahl von weiteren Programmen wie Office- oder Entertainment-Anwendungen verfügbar, die auf dem System zur Ausführung kommen können.

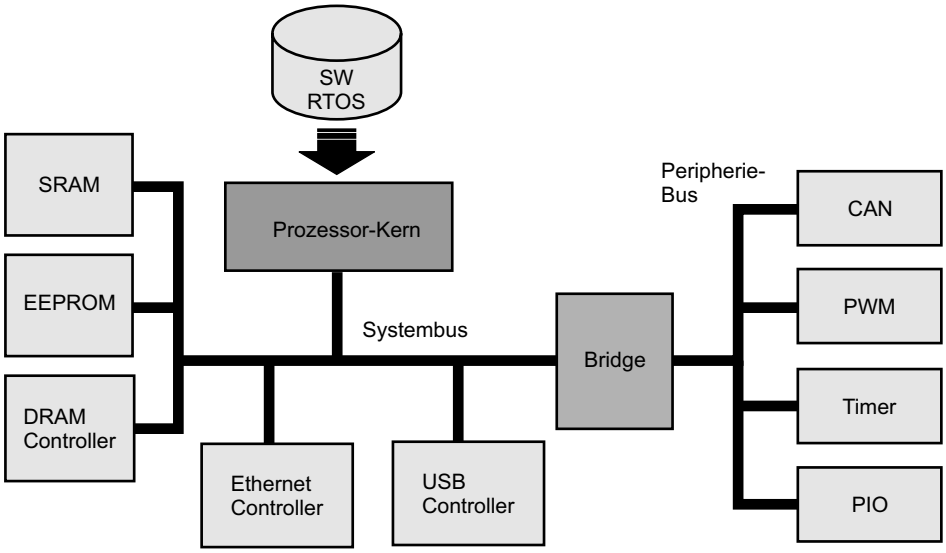

Abb. 1.1: *Exemplarisches Beispiel für ein „System-on-Chip“ mit Prozessor-Kern und Bussystem (Systembus und Peripheriebus über Bridge verbunden). Das System integriert Daten- und Programm-Speicher on-chip (SRAM, EEPROM) und es kann externer DRAM-Speicher angeschlossen werden. Ferner sind verschiedene Peripherieinheiten vorhanden (Ethernet, USB, CAN, PWM, Timer, PIO). Auf dem System läuft ein Echtzeit-Betriebssystem (RTOS), welches die Anwendungssoftware (SW) ausführt.*

Zunehmend verbreitet ist auch die Realisierung von SOCs auf anwenderprogrammierbaren FPGAs (Field-Programmable Gate Array), man spricht dann vom SOPC („System-on-a-Programmable-Chip“). FPGAs werden häufig auch benutzt, um Prototypen für ein ASIC zu implementieren. Die Hersteller von FPGAs, wie Altera [3] oder Xilinx [44], bieten für die Entwicklung von SOPCs vorentwickelte Designinformationen an, die als „IP-Cores“ bezeichnet werden (IP steht für „Intellectual Property“, im deutschen als „geistiges Eigentum“ bezeichnet). Als IP-Cores sind beispielsweise Komponenten wie Prozessor-Kerne, Speicher, Peripherieinheiten oder Bussysteme verfügbar. Mit Hilfe einer entsprechenden Entwurfssoftware kann die Hardware für ein SOPC in kurzer Zeit durch Auswahl und Konfiguration der IP-Cores entwickelt werden. Obgleich es auch Hersteller gibt, die vorentwickelte IP-Cores für den ASIC-Entwurf anbieten, so ist die Entwicklung von ASICs erheblich aufwändiger als die Entwicklung eines FPGAs. Dies liegt im Wesentlichen daran, dass für die Herstellung des ICs ein Satz von Be-

lichtungsmasken notwendig wird (siehe z.B. [18]), wofür wiederum die Layoutdaten notwendig sind. Die Entwicklung dieser Layoutdaten für ein ASIC in einer modernen Submikrometer-IC-Technologie – die Strukurgrößen liegen heute im Bereich von 20 Nanometer – erfordert teure Entwurfssoftware sowie ein größeres Entwicklungsteam. Zusammen mit den Kosten für die Belichtungsmasken können daher die Fixkosten für die ASIC-Entwicklung schon einige Millionen Euro betragen, so dass sehr hohe Stückzahlen für ein rentables Produkt benötigt werden. FPGAs sind demgegenüber vorgefertigte Standardbauelemente und müssen nur programmiert werden – eine Layoutentwicklung ist nicht notwendig. Dies spart für den Anwender gegenüber ASICs erhebliche Fixkosten, so dass FPGAs insbesondere bei kleinen Stückzahlen eingesetzt werden können. FPGAs benötigen aber durch die Programmierbarkeit im Vergleich zu ASICs mehr Chipfläche bei gleicher Funktionalität, so dass die Stückkosten von FPGAs höher sind.

Neben der mikroelektronischen Hardware ist für ein SOC oder SOPC entsprechende Software notwendig. In der Regel wird die Anwendungssoftware unter Kontrolle eines Betriebssystems ausgeführt, welches häufig auch Echtzeitanforderungen erfüllen muss (RTOS: Real-Time Operating System). Vielfach werden auch Betriebssysteme wie Linux oder Windows, die aus dem Desktop- oder Workstationbereich stammen, auf eingebettete Systeme portiert. Für die Peripherieinheiten sind entsprechende Treiber erforderlich und für die Ansteuerung komplexerer Schnittstellen wie USB, CAN oder Ethernet sind entsprechende Software-„Protokoll-Stacks“ notwendig. Der zeitliche und damit finanzielle Aufwand für die Softwareentwicklung – die häufig in C oder C++ erfolgt – eines SOCs oder SOPCs kann dabei den Aufwand für die Entwicklung der Hardware deutlich übersteigen. Ein wesentliches Aufgabengebiet in der Entwicklung eines SOCs ist die Verifikation von Hardware und Software und des Zusammenspiels von Hardware und Software, wobei nicht vergessen werden darf, dass das SOC auch nur ein Teil eines Gesamtsystems ist und beispielsweise auch das Zusammenspiel mit mechanischen Komponenten getestet und verifiziert werden muss – insbesondere in der Automobilelektronik, wo es auch um sicherheitskritische Anwendungen geht. Wir konzentrieren uns im Folgenden hauptsächlich auf den Entwurf der Hardware; eine Übersicht über die Gesamtproblematik des Entwurfs von SOCs und der Verifikation findet sich beispielsweise in [9].

1.2 Entwurf der mikroelektronischen Hardware: Modellierung und Abstraktion

Wenn wir die mikroelektronische Hardware – also die ICs – eines elektronischen Systems betrachten, so steigt die Anzahl der auf einem IC integrierbaren Transistoren nach dem *Moore'schen Gesetz* seit den Anfängen der integrierten Schaltungstechnik in den sechziger Jahren exponentiell an – mit einer Verdopplung der Anzahl der Transistoren etwa alle zwei Jahre. Heute beträgt die Transistordichte ungefähr 5 Millionen MOS-Transistoren pro Quadratmillimeter Chipfläche, so dass man je nach Größe des Chips einige 100 Millionen bis 1 Milliarde Transistoren auf einem Chip unterbringen kann. Während einerseits die Anzahl der integrierbaren Transistoren mit einer Rate von ungefähr 40–50% pro Jahr wächst, kann andererseits die Entwicklungsproduktivität mit diesem Wachstum nicht mithalten. Eine einfache Möglichkeit, die Entwicklungsproduktivität anzugeben, ist die Anzahl der entwickelten Transistoren pro Monat und pro Entwicklungsingenieur („Personenmonat“, wobei diese Metrik allerdings umstritten ist). Man kann davon ausgehen, dass die Entwicklungsproduktivität derzeit bei etwa 20.000

Transistoren pro Personenmonat liegt und sie erhöht sich nur mit etwa 20% pro Jahr. Für einen komplett neu zu entwickelnden Chip mit einer Komplexität von 100 Millionen Transistoren würde man also 5.000 Personenmonate oder 416 Personenjahre benötigen. Wollte man den Chip in einem Jahr entwickeln, so wäre ein Team von 416 Entwicklern notwendig, was nicht mehr vernünftig handhabbar wäre. Daher wird in der Hardwareentwicklung versucht, möglichst viele Komponenten des Gesamtsystems über die schon erwähnten IP-Cores abzudecken; also beispielsweise die Mikroprozessoren, die Speicherblöcke oder das Bussystem. Dennoch tut sich seit geraumer Zeit eine Lücke zwischen den technologisch möglichen Chip-Komplexitäten und den tatsächlich entwickelbaren Design-Komplexitäten auf – dies wird als „Entwurfslücke“ oder im Englischen als „design gap“ bezeichnet. Die Schließung der Lücke muss durch eine Weiterentwicklung der Entwurfsverfahren erreicht werden.

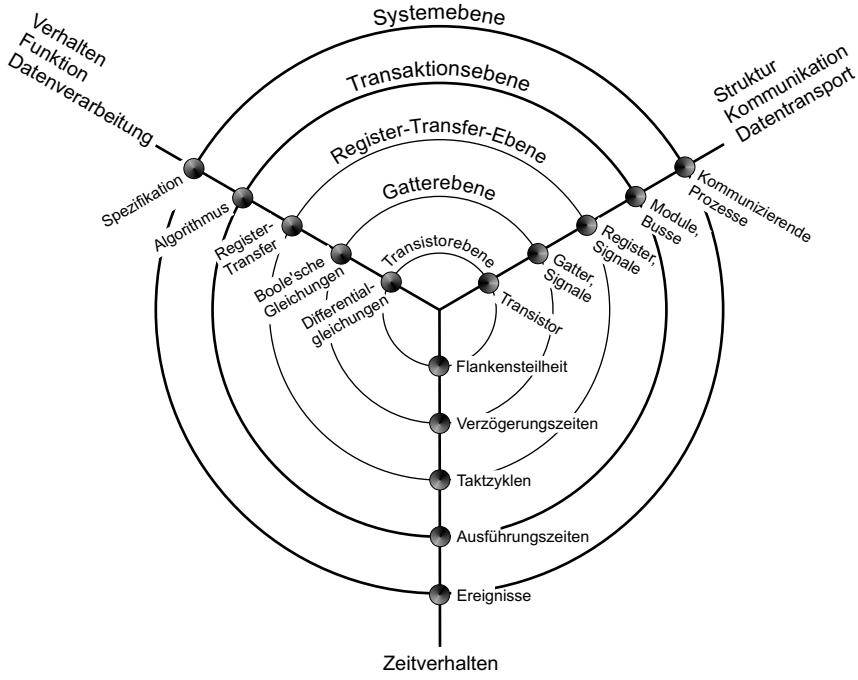

Abb. 1.2: *Modifiziertes Y-Diagramm zur Veranschaulichung der Abstraktionsebenen. Das Diagramm wurde ursprünglich von Gajski [13] und Walker [35] vorgeschlagen.*

Ein zentraler Begriff im Zusammenhang mit Entwurfsverfahren ist die *Abstraktion*. Blickt man etwas weiter in der Elektronik-Entwicklung zurück – speziell die Entwicklung digitaler Elektronik – so wurde Hardware in den sechziger und siebziger Jahren entwickelt, indem man aus Funktionstabellen boole'sche Gleichungen und daraus die Verschaltung von Logik-Gattern ableitete, welche man mit TTL-Bausteinen oder programmierbaren Schaltungen, wie beispielsweise PLAs, realisieren konnte. In den achtziger Jahren wurde mit der Einführung von *Hardwarebeschreibungssprachen*, wie VHDL oder Verilog, ein Abstraktionsschritt vorgenommen: Man beschreibt nicht mehr die Verschaltung von Logik-Gattern und Flipflops einer bestimmten Technologie, sondern beschreibt die Struktur und das Verhalten der Hardware auf der so

genannten „Register-Transfer-Ebene“ (engl.: Register-Transfer-Level, RTL, vgl. [18]). Die Abstraktion in der Hardwareentwicklung ist vergleichbar mit dem Übergang in der Softwareentwicklung von der Assemblerprogrammierung zur Programmierung in einer höheren Programmiersprache wie C oder C++. Durch die RTL-Methodik konnte man Anfang der neunziger Jahre erhebliche Fortschritte in der Produktivität der Entwicklung erzielen und diese Methodik ist auch heute noch gerade im FPGA-Entwurf vorherrschend, allerdings ist die Produktivität dieser Methodik für moderne SOCs nicht mehr ausreichend.

Die Abstraktion im Hardware-Entwurf wird häufig mit Hilfe des in Abbildung 1.2 gezeigten „Y-Diagramms“ veranschaulicht, welches ursprünglich von Gajski und Walker benutzt wurde und welches wir für die folgende Diskussion etwas modifiziert haben. Das Diagramm weist drei Äste oder Achsen auf, die als „Beschreibungs-Domänen“ eines Entwurfs bezeichnet werden; die konzentrischen Kreise werden als *Abstraktionsebenen* bezeichnet. Wir modifizieren das Y-Diagramm dahingehend, dass die ursprünglich verwendete „Geometrie“-Domäne durch eine Achse für das Zeitverhalten ersetzt wird. Eine zu entwickelnde Hardware kann auf unterschiedlichen Abstraktionsebenen beschrieben oder modelliert werden – man spricht dann auch von einem *Modell* des Chips. Bei der *Modellierung* oder Modellbildung wird immer eine mehr oder weniger starke Abstraktion der physikalischen Gegebenheiten vorgenommen. Die Abstraktion im Hardwareentwurf wird im Wesentlichen bezüglich der drei in Abbildung 1.2 gezeigten Domänen vorgenommen. Die Struktur beschreibt die Verbindung oder allgemeiner die Kommunikation und den Datentransport zwischen Komponenten auf einer Abstraktionsebene. Auf Gatterebene ist dies die Verschaltung von Gattern und Flipflops und auf Register-Transfer-Ebene werden komplexere Komponenten wie ALUs oder Register durch VHDL-Strukturbeschreibungen verschaltet. In beiden Fällen werden Signale und Komponenten-Ports zur Verbindung der Komponenten benutzt und man spricht dann bei einer Strukturbeschreibung auch von einer Netzliste. Das Verhalten beschreibt die Funktionalität der Komponenten oder die Datenverarbeitung: Auf Gatterebene sind dies die Boole'schen Gleichungen der Gatter oder die Funktionen der Flipflops. Auf Register-Transfer-Ebene wird die Funktionalität der Komponenten mit Hilfe von Programmiersprachen-Konstrukten, wie Verzweigungen (`IF`, `CASE`) oder Schleifen (`LOOP`), beschrieben.

In einem RTL-Modell wird die so genannte „Mikroarchitektur“ der Hardware beschrieben. Dies ist der Aufbau der Hardware aus getakteten Flipflops und Registern sowie der ungetakteten kombinatorischen Logik, die sich zwischen den Registern befindet. Neben der Struktur und dem Verhalten wird insbesondere auch das Zeitverhalten abstrahiert: Auf der Gatterebene werden die Verzögerungszeiten der Gatter modelliert und man kann beispielsweise die kritischen Pfade in der kombinatorischen Logik und damit die maximale Taktfrequenz der Schaltung bestimmen (vgl. auch [18]). Auf RT-Ebene werden die Verzögerungszeiten der Kombinatorik nicht mehr modelliert, sondern nur noch die Taktänderungszeitpunkte und damit die Zeitpunkte zu welchen die Register Daten übernehmen. Ein RTL-Modell ist daher „taktzyklengenau“ (engl.: cycle accurate); dies bedeutet, dass die reale Chip-Hardware die gleiche Anzahl von Taktschritten für die Abarbeitung benötigen wird wie das RTL-Modell. Ein Modell der Hardware auf Gatterebene ist ebenfalls taktzyklengenau, aber darüber hinaus auch Verzögerungszeit-genau. Der große Vorteil eines RTL-Modells besteht darin, dass es unabhängig von einer Zieltechnologie ist und damit auf beliebige Zieltechnologien „portiert“ werden kann – genauso wie ein C/C++-Code in der Softwareentwicklung portierbar ist. Eine Beschreibung auf Gatterebene ist nicht portierbar. Ein RTL-Modell, welches auf einem FPGA verifiziert wurde, kann daher ebenfalls für die Implementierung in einem ASIC benutzt werden.

Durch Abstraktion verliert man also auf der einen Seite an *Detaillierung* oder *Genauigkeit*, wie beispielsweise die Verzögerungszeiten von Gattern beim Übergang von der Gatterebene auf die RT-Ebene, man gewinnt aber Verständnis über sowie Einsicht in das zu entwickelnde System, so dass man letzten Endes produktiver wird und weniger Fehler in der Entwicklung macht. Wichtig ist insbesondere, dass abstraktere Modelle auch zu einer schnelleren Simulation führen. Nun wird für die physikalische Realisierung der Hardware auf einem Chip aber eine Implementierung mit Logikgattern erforderlich und für eine ASIC-Implementierung werden auch die Verschaltungen der Transistoren und das daraus resultierende Layout benötigt (vgl. [18]). Die Schritte von einer hohen Abstraktionsebene hin zur physikalischen Realisierung nennt man *Verfeinerung* oder auch *Transformation*. Bei jedem Verfeinerungsschritt sind Entwurfsentscheidungen zu treffen: Für ein VHDL-RTL-Modell gibt es eine Vielzahl von möglichen Implementierungen auf der Gatterebene, die alle die durch den VHDL-Code spezifizierte Funktion darstellen, sich aber im Ressourcenaufwand (Anzahl der Gatter und Flipflops, Flächenbedarf des ASICs) und im Zeitverhalten und damit der Leistungsfähigkeit (maximale Taktfrequenz) der Schaltung unterscheiden können. Es ist daher sinnvoll und wünschenswert, diese Verfeinerung nicht manuell durchführen zu müssen, sondern durch entsprechende Entwurfssoftware, um in annehmbarer Zeit verschiedene Alternativen untersuchen zu können. Die Verfeinerung oder Transformation von der RT-Ebene zur Gatterebene wird durch die so genannte „Logiksynthese“ (vgl. [18]) vorgenommen und die anschließende Umsetzung der Gatternetzliste in das Layout eines ASICs oder in die Programmierung eines FPGAs durch weitere Werkzeuge des physikalischen Entwurfs (beispielsweise „Place&Route“, vgl. [18]). Für die Transformation einer RTL-Beschreibung in eine physikalische Realisierung existiert also ein vollautomatischer Weg. Die Auswahl verschiedener Realisierungsalternativen wird dabei über „Randbedingungen“ (engl.: constraints) vom Entwickler gesteuert. In der Softwaretechnik sind diese Schritte vergleichbar mit der Anwendung von Compiler und Linker auf den Quellcode.

Die Weiterentwicklung der Entwurfsmethoden in den vergangenen zehn Jahren wurde durch eine weitere Erhöhung des Abstraktionsniveaus erreicht und kann eingeteilt werden in die weiteren, in Abbildung 1.2 gezeigten Abstraktionsebenen *Transaktionsebene* und *Systemebene*, wobei wieder die drei schon erwähnten Domänen Verhalten, Struktur und Zeitverhalten berücksichtigt werden. Es sind noch weitere Beschreibungs-Domänen denkbar, wie beispielsweise die Repräsentation der Daten, welche vom Rechnersystem verarbeitet werden: Auf RT-Ebene und Gatterebene werden die Daten in der Regel bitgenau dargestellt; man denke beispielsweise an den in VHDL verwendeten binären Datentyp `std_logic_vector`, welcher die genaue Bitbreite spezifiziert. Bei abstrakterer Modellierung würde man die in Programmiersprachen üblichen Datentypen wie beispielsweise die C-Datentypen **`int`**, **`float`** oder **`double`** benutzen. In der Signalverarbeitung kann man beispielsweise so vorgehen, dass man einen Algorithmus zunächst mit Gleitpunktzahlen entwirft und erst später zu ganzen Zahlen oder Fixpunktzahlen übergeht und dann den Einfluss der Zahlendarstellung auf den Algorithmus untersucht. Wir möchten uns im Folgenden aber auf die drei Domänen Verhalten, Struktur und Zeitverhalten konzentrieren.

Die Modellierung auf Transaktionsebene wird im Englischen als „Transaction Level Modeling“ (TLM) bezeichnet; dieser Begriff wurde vor etwas mehr als 10 Jahren durch die EDA-Firmen geprägt (EDA: Electronic Design Automation, also die Hersteller der Entwurfssoftware). Der Begriff „Level“ in TLM ist dabei irreführend: Bei RTL handelt es tatsächlich um *eine* Abstraktions-*Ebene*, die klar definiert ist (vgl. [18]). TLM ist nicht im gleichen Maße genau definiert und häufig wird mit TLM alles bezeichnet, was abstrakter ist als RTL. Es ist insbesondere

im Hinblick auf das Zeitverhalten so, dass TLM mehrere Genauigkeitstufen bei der Modellierung des Zeitverhaltens ermöglicht, so dass man nicht von *einer* Ebene sprechen kann. Dass es als „Transaction Level“ bezeichnet wurde, liegt daran, dass man bei der Einführung dieser Modellierungstechnik um das Jahr 2000 herum den Vergleich zur etablierten RTL-Modellierung anhand der Abstraktions-“Ebene“ ziehen wollte. Es ist allerdings besser im Zusammenhang mit TLM von einer Modellierungstechnik zu sprechen. Da SystemC sich zu einer der wichtigsten Sprachen für TL-Modelle entwickelt hat, trägt auch der IEEE Standard 1666-2011 [21] für SystemC zu einer Klärung bei, was man unter TLM verstehen kann.

TLM zielt insbesondere darauf ab, die Hardware von elektronischen Systemen effizient zu modellieren. Der Fokus liegt dabei auf den Kommunikationseinrichtungen des Systems, also insbesondere den Bussystemen. Eine wesentliche Idee von TLM ist es, bei der Modellierung die Kommunikation (engl.: communication) oder Datenübertragung von der Datenverarbeitung (engl.: computation) zu trennen [12]. Wenn wir also ein typisches SOC wie in Abbildung 1.1 betrachten, so könnnen wir es aufteilen in die datenverarbeitenden oder -speichernden Komponenten, wie Prozessor, Speicher und Peripherieeinheiten, und in die Komponenten, welche für den Datentransport zuständig sind, im Beispiel die beiden Teilbussysteme und die Bridge. Wenn ein synchrones Bussystem auf RT-Ebene modelliert wird, so erfolgt dies durch die Signale für Takt, Steuerleitungen, Adressen und Daten und die entsprechenden Zeitverläufe der Signale, welche durch das Busprotokoll vorgegeben sind. Ein RTL-Modell ist daher taktzyklengenau und modelliert auch alle Signale des Bussystems („Pin-genau“). In einem TLM-Modell verzichtet man dagegen vollständig auf Signale und auch auf die Angabe eines Taktes. Die Busse oder Kommunikationseinrichtungen des Systems werden als abstrakte „Kanäle“ (engl.: channel) modelliert und die Bus-Transaktionen (daher der Name „Transaktionsebene“) durch den Aufruf von so genannten „Interface-Methoden“ der Kanäle. Die TLM-Schnittstellen der Komponenten entsprechen im Grunde den später auf RT-Ebene zu implementierenden Busschnittstellen. Durch ein TL-Modell wird sowohl die Struktur des Busses, im Sinne der einzelnen Signale des Bussystems, als auch das Zeitverhalten, im Sinne von einzelnen Taktschritten für eine Transaktion, gegenüber einem RTL-Modell abstrahiert. Damit kann man insbesondere einen erheblichen Zuwachs in der Leistungsfähigkeit des Simulationsmodells gewinnen. Es wird somit möglich, auf einem TL-Modell der Hardware die Software zu entwickeln oder man kann die Architektur auf ihre Leistungsfähigkeit und andere Merkmale hin bewerten. Darüber hinaus sind Änderungen in der Architektur erheblich schneller durchführbar, verglichen mit einem RTL-Modell. Man spricht in diesem Zusammenhang auch von einem „virtuellen Prototyp“ des Systems.

Ein TL-Modell ist also nicht mehr Pin- und taktzyklengenau wie ein RTL-Modell. Allerdings werden die User-Register in den Peripherieeinheiten modelliert, so dass die Software auf dem Modell entwickelt werden kann. Dies wird im Englischen auch als „Programmer’s View“ bezeichnet. Die Funktionalität in den Komponenten – also den datenverarbeitenden Teilen – wird man ebenfalls nicht als RTL-Mikroarchitektur modellieren, sondern hier wird das Verhalten der Komponente abstrakter im Sinne einer Softwarefunktion beschrieben; man spricht dann auch von einem „algorithmischen“ Modell. Im Hinblick auf die Genauigkeit des Zeitverhaltens gibt es in einem TL-Modell mehrere Genauigkeitsstufen, wobei es hier hauptsächlich um die Modellierung des Busprotokolls geht: Man kann ein spezifisches Busprotokoll buszyklengenau modellieren, welches schon annähernd taktzyklengenau ist, oder man modelliert das Busprotokoll etwas abstrakter, so dass bei den Schreib- und Lesetransaktionen auf dem Bus nur die Auswahl der Bus-Targets (auch: Slaves) über die Dekodierung der vom Initiator (auch: Master)

gelieferten Transaktions-Adressen und das Übertragen der Daten modelliert wird, ohne auf die Spezifika eines bestimmten Busprotokolls einzugehen.

Eine gegenüber der Transaktionsebene noch stärkere Abstraktion findet auf der Systemebene statt. Hier geht es allerdings nicht mehr darum, nur ein Modell der Hardware darzustellen. Systemmodelle werden in einer sehr frühen Phase eingesetzt, wo die Aufteilung in Hardware und Software noch nicht festgelegt ist. Man möchte hier im Wesentlichen die Spezifikation in ein Modell übersetzen, welches auf dem Computer ausgeführt werden kann (engl.: executable specification) und welches bewusst frei von Implementierungsdetails ist. In der Signalverarbeitung kann man beispielsweise MATLAB/Simulink [41] einsetzen, um Simulationsmodelle zu entwickeln. In der Regel geht es aber in dieser Phase auch darum, potentielle Parallelitäten oder Nebenläufigkeiten zu entdecken und diese auch modellieren zu können, beispielsweise durch entsprechende nebenläufige Prozesse. Das Zeitverhalten wird ebenfalls noch stärker abstrahiert, so dass man einen zeitlichen Verlauf unter Umständen auch gar nicht mehr modelliert (engl.: untimed) und die Ausführung der Prozesse über abstrakte Ereignisse steuert.

1.3 Electronic System Level Design

Die Entwicklung von elektronischen Systemen wurde in den achtziger und neunziger Jahren so vorgenommen, dass zunächst die mikroelektronische Hardware und dann in einer späteren Phase die zugehörige Software auf speziellen Entwicklungsversionen oder Emulatoren der Hardware entwickelt wurde. Seit den neunziger Jahren werden auch vermehrt FPGAs als Hardware-Prototypen eingesetzt. Diese Vorgehensweise hat zwei Nachteile: Zum einen kann die Softwareentwicklung erst in einer relativ späten Phase erfolgen und zum zweiten werden hinsichtlich der Hardware-Architektur möglicherweise Entscheidungen getroffen, die sich bei der späteren Integration von Hardware und Software als nicht optimal herausstellen.

Für die Entwicklung moderner SOCs oder SOPCs hat sich der Begriff „Electronic System Level Design“ (oder kurz: ESL-Design) etabliert [9]. Darunter versteht man die Methoden, Vorgehensweisen und Werkzeuge, die heute für die Entwicklung von komplexen elektronischen Systemen benutzt werden (siehe Abbildung 1.3). Wesentlich dabei ist es, dass Hardware und Software zusammen entwickelt werden, um mögliche Probleme bei der Integration von Hardware und Software schon in einer frühen Phase entdecken zu können – man spricht auch vom so genannten „Hardware/Software-Codesign“. Ferner wird der Abstraktionsgrad gegenüber dem Entwurf auf RT-Ebene auf die Transaktionsebene (oder die Systemebene) angehoben, wodurch die Produktivität erhöht werden kann. Man geht davon aus, dass man hierdurch etwa einen Faktor 10 in der Produktivität gewinnt [8].

Ausgehend von einer Spezifikation kann das System zunächst mit entsprechenden Werkzeugen und Modellierungssprachen, wie beispielsweise MATLAB/Simulink, UML/SysML oder C++/SystemC, modelliert werden, ohne zunächst eine konkrete Aufteilung in Hardware und Software vorzunehmen – man erhält die schon erwähnte „ausführbare Spezifikation“. Im einfachsten Fall kann es sich um ein C++-Programm handeln, welches den zu implementierenden Algorithmus beschreibt. Anhand dieses Modells können beispielsweise Entscheidungen über die zu verwendenden Algorithmen aufgrund von Analysen getroffen werden. Hierzu zählen beispielsweise Untersuchungen der Qualität der Algorithmen (z.B. Bitfehlerraten) oder des Zahlenformats in der digitalen Signalverarbeitung (Fixpunkt- oder Gleitpunktzahlen, Bitbreite

etc.), aber auch Untersuchungen hinsichtlich des Aufwands oder Abschätzungen des Energieverbrauchs. Die darauf folgenden und in Abbildung 1.3 gezeigten Entwurfsschritte dienen dazu, das abstrakte Systemmodell durch schrittweise Verfeinerungen in eine Implementierung zu überführen. Diese besteht zum einen aus der mikroelektronischen Hardware und zum anderen aus der auf den integrierten Prozessoren laufenden Software.

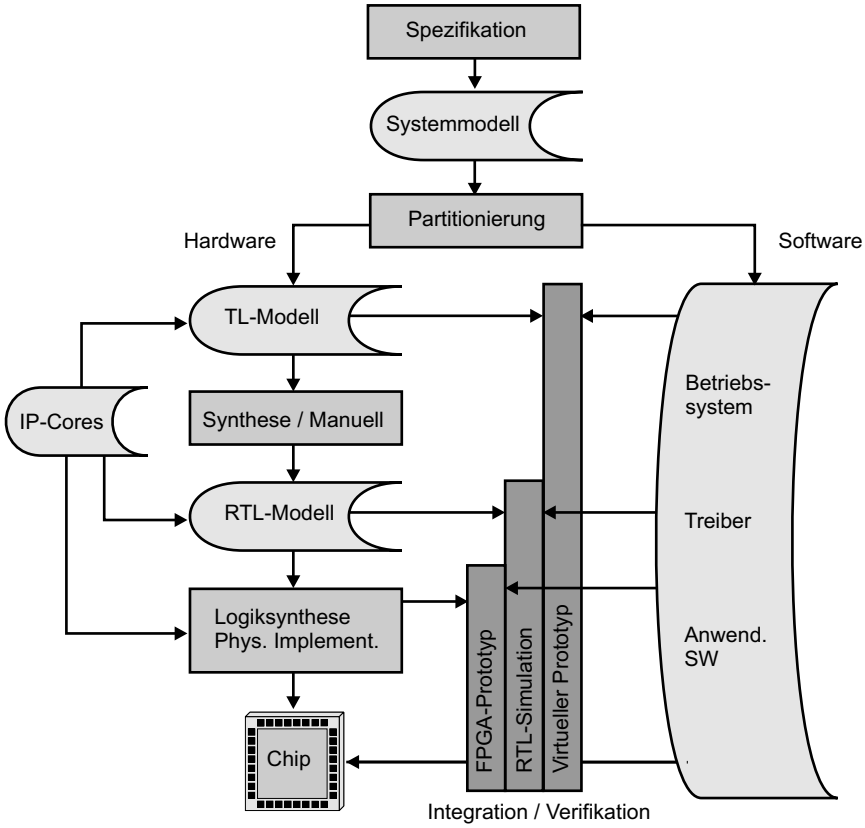

Abb. 1.3: *Entwurfsablauf im ESL-Design.*

Der in Abbildung 1.3 gezeigte „Top-Down"-Ablauf ist idealisiert und wird in der realen, industriellen Entwicklung zumeist nicht streng verfolgt. Möglicherweise liegt schon ein existierendes Produkt oder eine so genannte „Plattform" vor, welche abgewandelt wird, oder es sind bestimmte Komponenten vorhanden, die wieder verwendet werden sollen. Auch ist es in der Regel kein linearer Ablauf von der Spezifikation bis zum fertigen Produkt. Häufig sind Änderungen durchzuführen, die bis zu (wiederholten) Änderungen der Spezifikation reichen können, so dass möglicherweise mehrere Schleifen in diesem Ablauf vorhanden sind. Wir möchten den in Abbildung 1.3 gezeigten, idealisierten Ablauf jedoch benutzen, um den Einsatz von SystemC im Entwurf von elektronischen Systemen einordnen und abgrenzen zu können. Wir können allerdings an dieser Stelle nur einen kurzen Einblick in die ESL-Methodik geben, für eine weitergehende Behandlung des Themas sei beispielsweise auf [9] verwiesen.

Die Entwurfsentscheidungen im Zuge der Verfeinerungen sind Entscheidungen über die Architektur der nächst tiefer liegenden Ebene, wobei es sich dabei sowohl um die Architektur der Hardware wie auch der Software handelt – wir konzentrieren uns im Folgenden hauptsächlich auf die Hardware. Eine erste Entscheidung muss über die Aufteilung des Systems in Hardware und Software getroffen werden, dies wird als *Partitionierung* bezeichnet. Hierfür ist zuvor eine Analyse notwendig, um die Nebenläufigkeiten und Parallelitäten in der Anwendung zu finden und im Sinne einer höheren Leistungsfähigkeit des Systems ausnutzen zu können. Dies wird als „funktionale Dekomposition“ bezeichnet (vgl. [9]) und kann beispielsweise durch ein SystemC-Systemmodell erfolgen, in welchem die Nebenläufigkeiten durch Prozesse modelliert werden.

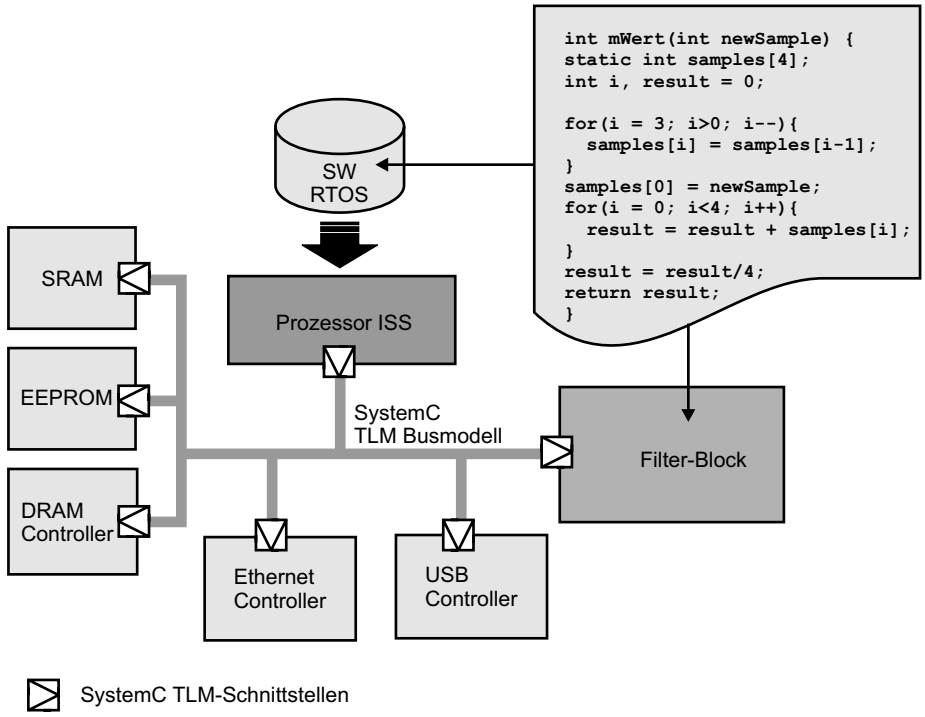

Abb. 1.4: *Beispiel zur Partitionierung und TL-Modellierung. ISS ist ein Instruktionssatzsimulator für einen gewählten Prozessorkern. Die Komponenten des Systems sind mit SystemC-TLM-Schnittstellen versehen und werden über ein TL-Modell des Bussystems gekoppelt. Für die Mittelwertfilter-Funktion kann beispielsweise entschieden werden, ob sie in Software auf dem Prozessor laufen soll oder ob sie als Systemkomponente in Hardware implementiert werden soll.*

Anschließend muss eine Architektur für die Hardware definiert werden und entschieden werden, welche Teile des Systems in Software auf den Prozessoren laufen und welche als Hardware-Module implementiert werden. Abbildung 1.4 zeigt ein Beispiel für ein SystemC-Modell eines Rechnersystems auf Transaktionsebene. Beispielsweise könnte man mit Hilfe dieses Modells untersuchen, ob eine Mittelwertfilterfunktion besser als Software-Funktion oder als spezielle Peripherieeinheit und damit in Hardware implementiert wird. Ist eine Partitionierung und

eine Hardware-Architektur gefunden, so ist es notwendig, diese bewerten zu können. Ferner möchte man in dieser frühen Phase auch schon mit der Software-Entwicklung und deren Integration mit der Hardware beginnen können. Hierzu wird ein Simulationsmodell für die Hardware benötigt, welches eine schnelle Simulation ermöglicht und auch so flexibel ist, dass Änderungen schnell vorgenommen werden können. Hierfür ein VHDL-Modell auf RT-Ebene benutzen zu wollen, verbietet sich aus den schon im letzten Abschnitt erläuterten Gründen, so dass für diese Entwurfsphase virtuelle Prototypen eingesetzt werden, welche auf Transaktionsebene mit SystemC modelliert werden.

Die Modellierung auf Transaktionsebene fokussiert auf die Kommunikationseinrichtungen des Systems. Für die Komponenten des Systems, die über diese Kommunikationseinrichtungen verbunden werden, gibt es verschiedene Möglichkeiten, diese zu modellieren. Um ein leistungsfähiges Simulationsmodell zu erhalten, wird man sich natürlich bemühen, auch die Komponenten auf einer möglichst hohen Abstraktionsebene zu modellieren. Für die Prozessorkerne werden üblicherweise so genannte „Instruktionssatzsimulatoren“ (ISS, engl.: instruction set simulator) verwendet, welche den Binärcode der kompilierten Software ausführen können und darüber hinaus auch an so genannte „Debugger“ angebunden werden könnnen, um Fehler in der Software analysieren zu können. ISS sind von den Herstellern der Prozessoren oder von Drittanbietern erhältlich, häufig auch in C oder C++. Zumeist wird noch ein SystemC-„Wrapper“ benötigt, welcher den ISS an den Bus anbindet, so dass entsprechend der Befehlsausführung des Prozessors TLM-Transaktionen erzeugt werden. Die Firma OVP [31] bietet beispielsweise ISS mit SystemC-Schnittstellen für viele bekannte Prozessoren an, die in SOCs eingesetzt werden. Für die anderen Komponenten des Systems müssen gegebenenfalls Simulationsmodelle mit TLM-Schnittstellen geschrieben werden, wobei das Verhalten der Komponente ebenfalls möglichst abstrakt mit SystemC/C++ modelliert werden sollte. Mittlerweile sind auch für viele IP-Cores von Peripherie-Komponenten und Bussystemen entsprechende SystemC-Modelle verfügbar.

Ist man mit der gefundenen Hardware-Architektur zufrieden, so stellt sich Frage nach der Umsetzung in die physikalische Implementierung. Wie in Abbildung 1.3 gezeigt, muss dazu die Transformation vom TL-Modell zum RTL-Modell erfolgen. Der weitere in Abbildung 1.3 gezeigte Weg von der RT-Ebene mittels Logiksynthese zur physikalischen Implementierung soll hier nicht weiter diskutiert werden, wir verweisen hierzu beispielsweise auf [18]. Für die Komponenten und das Bussystem muss also entweder eine synthesefähige RTL-Beschreibung in VHDL oder Verilog gefunden werden oder gleich eine Implementierung als Netzliste oder Layout in der gewünschten Zieltechnologie. Hat man die Komponenten als IP-Blöcke eines Herstellers vorliegen, so sind diese Informationen in der Regel auch vorhanden. Für eigenentwickelte Komponenten muss man selbst eine synthesefähige RTL-Beschreibung erstellen. Wünschenswert ist eine automatische Umsetzung des TL-Modells in eine RTL-Architektur und letztlich in eine physikalische Implementierung.

Will man eine Komponente, wie in unserem Beispiel der Filter-Block aus Abbildung 1.4, in eine RTL-Beschreibung umsetzen, so sind zwei Aufgaben zu lösen (Abbildung 1.5): Erstens muss für das algorithmische Modell – hier die C-Funktion des Mittelwertfilters – eine Mikroarchitektur gefunden werden und zweitens muss für die abstrakte TLM-Schnittstelle eine Implementierung einer Busschnittstelle für ein konkretes Busprotokoll gefunden werden. Die erste Aufgabe kann automatisiert durch die so genannte „High-Level Synthese“ (HLS) erledigt werden (vgl. z.B. [27]). Die HLS, welche auch als Architektur- oder Verhaltenssynthese (engl.: be-

havioral synthesis) oder als algorithmische Synthese bezeichnet wird, kann für eine algorithmische Beschreibung, die beispielsweise als C/C++-Funktion vorliegt, eine Mikroarchitektur auf RT-Ebene finden, welche den Algorithmus in Hardware implementiert. Die Mikroarchitektur besteht im Wesentlichen aus einem Steuerwerk, einem Datenpfad und Registern. Über entsprechende Randbedingungen kann der Syntheseprozess vom Benutzer gesteuert werden und damit die entstehende Mikroarchitektur beeinflusst werden. Die wichtigsten Parameter sind dabei die Leistungsfähigkeit der Hardware – beispielsweise die Frage, welche Anzahl von Taktschritten für die Abarbeitung benötigt wird – und der Ressourcenverbrauch. Während sich erste Softwarewerkzeuge zur HLS, wie der „Behavioral Compiler“ von Synopsys, in den neunziger Jahren nicht in der industriellen Anwendung durchsetzen konnten, so sind in den letzten Jahren einige neue HLS-Werkzeuge auf den Markt gekommen, die im Kontext des ESL-Designs größere Chancen auf Anwendung haben. Zu nennen wäre hier beispielsweise „C-to-Silicon“ von Cadence (vgl. [8]) oder „Catapult“ von Mentor Graphics (vgl. [30]). Auch für den FPGA-Entwurf werden mittlerweile HLS-Werkzeuge angeboten, beispielsweise das „AutoESL“-Werkzeug von Xilinx.

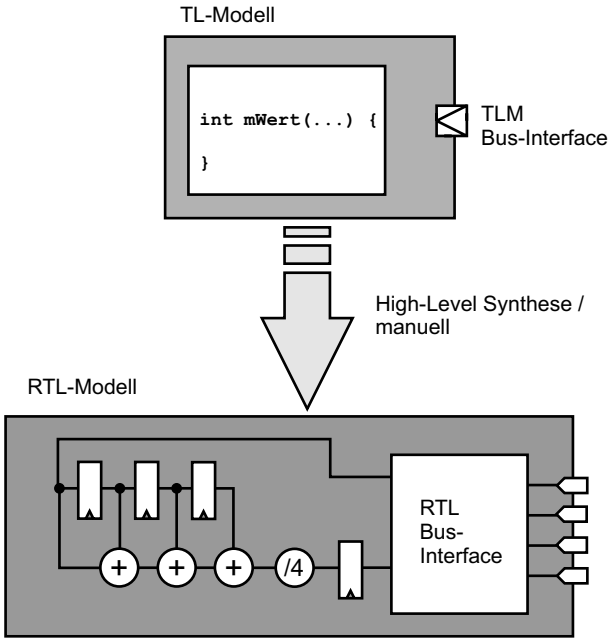

Abb. 1.5: *Transformation vom TL-Modell zum RTL-Modell am Beispiel des Mittelwertfilters. Für die Funktion des Mittelwertfilters muss eine RTL-Architektur gefunden werden und für die TLM-Schnittstelle eine RTL-Implementierung einer Busschnittstelle für das spezifische Busprotokoll. Dies kann entweder automatisiert durch ein Synthesewerkzeug erfolgen oder manuell.*

Für die zweite Aufgabe, der Umsetzung der TLM-Schnittstelle in ein RTL-Bus-Interface, benötigt man entsprechende IP-Blöcke, welche die datenverarbeitende Einheit (im Beispiel das Mittelwertfilter) an den Bus anbindet – beispielsweise an einen AMBA-AHB- oder AXI-Bus. In diesem Bus-Interface können sich auch die User-Register befinden. Darüber hinaus muss das

Bus-Interface das entsprechende Busprotokoll behandeln können und eine Pin-genaue Schnittstelle für die Anbindung an den Bus liefern. Gegebenenfalls müssen auch Daten für die datenverarbeitende Einheit über FIFOs gepuffert werden können. Auch dieser Schritt sollte von den EDA-Werkzeugen übernommen werden können.

1.4 SystemC als Modellierungssprache

In den neunziger Jahren stellte sich die Situation im Systementwurf so dar, dass die digitale Hardware mit Hardwarebeschreibungssprachen wie VHDL [24] oder Verilog [22] entwickelt wurde und die Software mit höheren Programmiersprachen, wie C oder C++. Da man zunehmend dazu überging, Software und Hardware gemeinsam zu entwickeln, kam der Wunsch auf, möglichst nur noch eine Sprache sowohl für die Hardware- als auch für die Softwareentwicklung benutzen zu können. Insbesondere gab es ein ganze Reihe von Ansätzen, C oder C++ hierfür einzusetzen, wobei hiermit in der Regel auch eine abstraktere Modellierung der Hardware – im Vergleich zur RTL-Modellierung mit VHDL oder Verilog – verbunden war. Ferner war mit diesen Ansätzen zumeist auch die automatische Transformation eines C-Modells in eine physikalische Implementierung durch ein Synthesewerkzeug verknüpft. Wesentliche Vorarbeiten zur TL-Modellierung wurden insbesondere von der Forschungsgruppe um Daniel Gajski an der University of California in Irvine geleistet und mündeten in die Sprache SpecC, welche wie SystemC auf C/C++ basiert [14]. Als weiteres Beispiel sei „Handel-C“ erwähnt, welches an der Universität von Oxford in den neunziger Jahren entwickelt wurde und später von den Firmen Celoxica und Agility und schließlich von Mentor Graphics kommerzialisiert wurde [2]. Handel-C implementiert eine Untermenge von C und fügt einige Erweiterungen hinzu, die aus der Programmiersprache Occam kommen. Wie in SpecC und auch in SystemC dienen diese Erweiterungen im Wesentlichen dazu, Eigenschaften der Hardware beschreiben zu können – insbesondere die in der Hardware vorhandenen Parallelitäten und die Kommunikation über die schon erwähnten Kanäle.

Die Ursprünge von SystemC liegen in verschiedenen Vorarbeiten, die in den neunziger Jahren von EDA-Firmen, Universitäten und Instituten geleistet wurden (vgl. auch [43]). Die schon genannten Arbeiten von Daniel Gajski zu SpecC haben die Entwicklung von SystemC maßgeblich beeinflusst. Insbesondere wurde die Entwicklung von SystemC durch das Vorläufer-Projekt Scenic (Synopsys und Universität von Kalifornien in Irvine [37], 1997) beeinflusst, in welchem eine Entwurfsumgebung für die Modellierung von Hardware und Software eines Systems entwickelt wurde. Im Vordergrund stand dabei die Simulation des Systems, es wurde aber auch die automatisierte Transformation von Scenic-Modellen in Hardware-Implementierungen angedacht. Die Basis für Scenic war die Programmiersprache C++. Neben speziellen Hardware-Datentypen wurde in Scenic insbesondere das aus Hardwarebeschreibungssprachen wie VHDL oder Verilog bekannte Konzept der nebenläufigen Prozesse und deren Steuerung über Ereignisse implementiert, um die Parallelitäten der Hardware modellieren zu können. Diese Mechanismen wurden als C++-Klassenbibliothek implementiert.

Da SystemC aus dem Vorläufprojekt Scenic heraus entstand, ist auch SystemC keine neue Programmiersprache, sondern besteht ebenfalls aus C++-Klassenbibliotheken. Um ein SystemC-Modell simulieren zu können wird – wie in VHDL oder Verilog – ein Simulator benötigt, welcher nach dem Prinzip der diskreten, ereignisgesteuerten Simulation arbeitet (vgl. beispielsweise [18]). Im Unterschied zu VHDL oder Verilog ist der Simulator-Kern aber Bestandteil

der Klassenbibliothek, so dass zur Simulation eines SystemC-Modells kein spezielles Simulationswerkzeug benötigt wird – der Simulator-Kern ist Bestandteil des ausführbaren Programms. Weitere wesentliche Bestandteile der SystemC-Bibliothek sind Module, um ein Modell strukturieren zu können, sowie Ports und Kanäle, um die Module und Prozesse miteinander verbinden zu können. Der Mechanismus der Kanäle verallgemeinert die von VHDL oder Verilog bekannten Signale und ist eine wesentliche Grundlage für die Modellierung auf Transaktionsebene. Darüber hinaus sind in der Bibliothek ebenfalls spezielle Hardware-Datentypen vorhanden und Mechanismen für die Modellierung des Zeitverlaufs eines Modells.

Die Arbeiten an SystemC wurden ab 1999 durch die OSCI (Open SystemC Initiative) koordiniert. OSCI war ein Konsortium von EDA-Firmen wie Mentor Graphics, Cadence oder Synopsys, von Halbleiterfirmen wie NXP, TI, ST, AMD, Renesas, Freescale oder Intel und von weiteren Firmen wie ARM oder Qualcomm. Dies gab SystemC den nötigen Schwung, um sich als die wesentliche Sprache für die Systemmodellierung zu etablieren. Die erste Version 1.0 der Klassenbibliothek wurde im Jahr 2000 kostenfrei zur Verfügung gestellt. Während diese erste Version im Prinzip ein „Nachbau“ der schon aus VHDL oder Verilog bekannten Mechanismen war, so wurde SystemC in der Version 2.0 (Veröffentlichung im Jahr 2002) weiterentwickelt, um eine abstraktere Modellierung auf System- oder Transaktionsebene zu ermöglichen. Darüber hinaus wurde ein so genanntes „Language Reference Manual“ (LRM) geschrieben, welches die Grundlage für die Standardisierung von SystemC durch den IEEE im Jahr 2005 war (IEEE Standard 1666-2005 [23]). Im Jahr 2007 wurde die letzte Version 2.2 der Bibliothek veröffentlicht, welche den IEEE Standard 1666-2005 implementiert.

Neben den Arbeiten an der eigentlichen SystemC-Bibliothek entstanden bei der OSCI weitere Arbeitsgruppen, die sich um zusätzliche Aspekte im Zusammenhang mit SystemC kümmern. Zu nennen sind hier beispielsweise die Arbeitsgruppe „SystemC Synthesis“, die sich um die Definition der synthesefähigen SystemC-Konstruktionen kümmert, die Arbeitsgruppe „SystemC Verification“, die sich um Erweiterungen von SystemC für die Verifikation kümmert, und die Arbeitsgruppe „SystemC Analog/Mixed-Signal“, die sich um die Modellierung von analogen oder gemischt analog/digitalen Systemen kümmert. Eine der wichtigsten Arbeitsgruppen für die Systemmodellierung ist „SystemC Transaction Level Modeling“, in welcher eine weitere Klassenbibliothek für die Modellierung auf Transaktionsebene entwickelt wurde. Diese Bibliothek baut auf der eigentlichen SystemC-Bibliothek auf und fokussiert sich auf die effiziente Modellierung von On-Chip-Bussystemen und damit der Kommunikationseinrichtungen des Systems im Sinne der in den vorangegangenen Abschnitten besprochenen TLM-Methodik. Die erste Version wurde im Jahre 2005 als TLM 1.0 veröffentlicht und zur aktuellen Version TLM 2.0.1 aus dem Jahre 2009 weiterentwickelt; diese wird als TLM-2.0-Bibliothek bezeichnet. Begleitend dazu wurde auch ein „Language Reference Manual“ geschrieben. Ein wesentliches Ziel dieser TLM-2.0-Bibliothek ist die so genannte „Interoperabilität“: Die TLM-Schnittstellen der Komponenten sind so ausgelegt, dass Komponenten-Modelle von verschiedenen Herstellern problemlos zu einem Gesamtsystem verbunden werden können. Neben standardisierten Schnittstellen und Interface-Methoden definiert die TLM-2.0-Bibliothek auch ein Basisprotokoll für das Bussystem, welches erweiterbar ist. Ferner werden zwei Modellierungsstile für die Modellierung des Zeitverhaltens des Bussystems unterstützt, welche mit unterschiedlicher Genauigkeit hinsichtlich der Modellierung der einzelnen Bustransaktionen arbeiten. Neben der Interoperabilität war es insbesondere auch ein Ziel, eine möglichst hohe Leistungsfähigkeit in der Simulation von TLM-Modellen zu erzielen. Zwei weitere wesentliche Schritte in der Entwicklung des SystemC-Standards erfolgten im Jahr 2011: Zum einen vereinigten sich OSCI und

Accellera zur „Accellera Systems Initiative“ [1], so dass die SystemC-Aktivitäten in dieser neuen Organisation weitergeführt werden. Die oben erwähnten SystemC-Arbeitsgruppen werden ebenfalls von Accellera weitergeführt und es ist nach wie vor möglich, die entsprechenden Dokumente und C++-Bibliotheken von der Website der Organisation herunterzuladen. Eine weitere wesentliche Neuerung besteht darin, dass es eine neue Version des SystemC-IEEE-Standards gibt (IEEE 1666-2011, [21]). Neben einigen wenigen neuen Mechanismen und Ergänzungen für SystemC wurde insbesondere das TLM-2.0-LRM in den Standard integriert, so dass TLM-2.0 nun Bestandteil des Standards IEEE 1666-2011 ist. Im Juli 2012 wurde daher die Version 2.3 der SystemC-Bibliothek veröffentlicht, die nun auch die TLM-2.0-Bibliothek enthält.

Wie wir schon im vorangegangenen Abschnitt ausgeführt haben, liegt der wesentliche Einsatzbereich von SystemC in der Systemmodellierung und – unter Verwendung der TLM-Bibliothek – in der Modellierung auf Transaktionsebene. Wie wir im Verlauf des Buches noch zeigen werden, kann man mit SystemC aber auch Modelle auf RT-Ebene schreiben und simulieren; dennoch ist die RTL-Modellierung eher die Domäne der „klassischen“ Hardwarebeschreibungssprachen wie VHDL oder Verilog. SystemC bietet auf der anderen Seite, neben der Objektorientierung durch C++, eine Reihe von Mechanismen für die Modellierung auf System- und Transaktionsebene an, wie beispielsweise die Kanäle, welche in VHDL oder Verilog nicht vorhanden sind. SystemC ist also für die neuen ESL-Entwurfsverfahren, die sich eine stärkere Abstraktion zu Nutze machen, besser geeignet, als die traditionellen Hardwarebeschreibungssprachen. Der Fokus von SystemC liegt insbesondere auch darin, die Modellierung von virtuellen Prototypen des Systems zu ermöglichen, die eine deutlich höhere Simulationsleistung aufweisen als dies mit VHDL- oder Verilog-Modellen auf RT-Ebene möglich wäre. Obwohl zunächst die Simulation im Vordergrund steht, so sind doch zunehmend Ansätze erkennbar, SystemC-Modelle auf Transaktionsebene mit Hilfe von Synthesewerkzeugen anschließend automatisch in eine physikalische Implementierung umzusetzen.

1.5 Leistungsfähigkeit von Simulationsmodellen

Die Leistungsfähigkeit eines Simulationsmodells kann beurteilt und verglichen werden, indem man die Ausführungszeit der Simulation auf dem Computer misst. Da das Simulationsprogramm auf einem Computer mit einem Betriebssystem läuft, können neben dem Simulationsprogramm noch weitere Anwendungen oder Prozesse während der Messung ausgeführt werden. Man unterscheidet daher zwischen der Ausführungszeit, die zwischen Beginn und Ende der Simulation verstreicht (im Englischen als „elapsed time“ oder „wall clock time“ bezeichnet), und der Zeit, welche der Computer tatsächlich für das Simulationsprogramm benötigt (im Englischen als „CPU time“ bezeichnet). Will man beispielsweise einen fairen Vergleich von verschiedenen Computern mit einem Benchmark-Programm durchführen, so sollte man die CPU-Zeit hierzu benutzen. Allerdings ist die Ermittlung der CPU-Zeit für den Anwender, je nach Betriebssystem, nicht immer ganz einfach. Für unsere Zwecke genügt es, wenn wir im Folgenden die Ausführungszeit als „elapsed time“ betrachten. Wir bezeichnen diese gemessene Ausführungszeit mit T_A. Da wir ein Modell der Hardware simulieren, ist es für eine Beurteilung der Simulationsleistung auch wichtig, welche Dauer der Modellzeit oder der simulierten Zeit simuliert wird. Dies ist die Zeitdauer, welche die Hardware später tatsächlich für den simulierten Ablauf benötigen wird; wir bezeichnen diese Zeit mit T_S. Ein Maß für die Leistungsfähigkeit eines Simulationsmodells der Hardware könnte nun beispielsweise der Quotient T_A/T_S sein,

wobei ein geringerer Wert eine höhere Simulationsleistung bedeutet. Beträgt dieser beispielsweise $T_A/T_S = 100.000$, so bedeutet dies, dass wir für eine Sekunde simulierte Zeit eine Ausführungszeit von $100.000\,\mathrm{s} = 27{,}8\,\mathrm{h}$ benötigen.

Wie schon erwähnt wurde, hängt die Simulationsleistung vom Abstraktionsgrad des Modells ab. Je abstrakter und damit ungenauer das Modell ist, desto kleiner wird die Ausführungszeit T_A des Modells sein (bei gleicher simulierter Zeit T_S) und damit eine höhere Simulationsleistung erzielen. Für die Beurteilung der TL-Modellierungstechnik wird sehr häufig ein Vergleich zu einem entsprechenden RTL-Modell gezogen. Wie wir schon erwähnt haben, wird sowohl für die Simulation eines TL-Modells als auch eines RTL-Modells im Grunde der gleiche Simulationsalgorithmus benutzt – die diskrete, ereignisgesteuerte Simulation. Die Ausführungszeit T_A eines Simulationsmodells für einen solchen Simulator hängt dabei im Wesentlichen davon ab, wie viele Ereignisse in der zu simulierenden Zeit T_S auftreten. Ereignisse sind bei einem RTL-Modell Änderungen von Signalwerten (vgl. auch [18]). Wir können bei den zu simulierenden elektronischen Systemen davon ausgehen, dass diese synchron sind und im einfachsten Fall mit einem Takt mit einer Taktfrequenz f_c betrieben werden. Die Anzahl N der zu simulierenden Taktzyklen ist dabei durch $N = T_S \times f_c$ bestimmt. Bei einer Taktfrequenz von $f_c = 100\,\mathrm{MHz}$ und einer simulierten Zeit von $T_S = 1\,\mathrm{s}$ ergeben sich also 100 Millionen zu simulierende Taktzyklen. Wir können davon ausgehen, dass die Anzahl der Ereignisse und damit auch die Ausführungszeit eines RTL-Modells proportional zur Anzahl der Taktzyklen ist. Wenn wir also eine geringere Taktfrequenz von beispielsweise $f_c = 100\,\mathrm{kHz}$ simulieren, so können wir davon ausgehen, dass die Ausführungszeit T_A – bei gleicher simulierter Zeit T_S – um einen Faktor 1000 niedriger sein wird, da wir eine um den gleichen Faktor geringere Anzahl von Taktzyklen simulieren müssen. Der Quotient T_A/T_S wird also kleiner – und damit die Simulationsleistung größer – wenn die simulierte Taktfrequenz des RTL-Modells verringert wird. Wenn wir also die Simulationsleistung über den Quotienten T_A/T_S beurteilen möchten, so sollten wir wissen für welche modellierte Taktfrequenz f_c dies ermittelt wurde.

$$P = f_{sim} = N/T_A = \frac{T_S}{T_A} \times f_c \tag{1.1}$$

Man geht daher häufig dazu über, als Metrik für die Simulationsleistung P nach Gleichung 1.1 die Anzahl N der simulierten Taktzyklen pro Sekunde Ausführungszeit anzugeben. Wie man erkennen kann, handelt es sich auch um eine Frequenz und sie wird daher auch als *Simulationsfrequenz* f_{sim} bezeichnet. Die Einheit ist Hertz, sie wird im Englischen aber häufig als „cycles-per-second“ (cycles/s oder „cps“) angegeben, um sie von der Taktfrequenz des Systems zu unterscheiden. Wir geben die Simulationsfrequenz daher im Folgenden ebenfalls in Zyklen/s an. Die Simulationsfrequenz ist ein Charakteristikum für die Leistungsfähigkeit des Modells und sie ist unabhängig von der simulierten Taktfrequenz f_c und eignet sich daher gut für den Vergleich von Simulationsmodellen. Da aber $N = T_S \times f_c$ ist, können wir nach Gleichung 1.1 auch einen Zusammenhang zwischen Simulationsfrequenz, Taktfrequenz, Ausführungszeit und simulierter Zeit herstellen. Daraus lässt sich erkennen, dass gilt: $T_S/T_A = f_{sim}/f_c$. Das Verhältnis von simulierter Zeit zu Ausführungszeit entspricht also dem Verhältnis von Simulationsfrequenz zu simulierter Taktfrequenz. Bei einem RTL-Modell eines elektronischen Systems, wie es in Abbildung 1.1 zu sehen ist, kann man davon ausgehen, dass $f_{sim} \approx 1\,\mathrm{kZyklen/s}$ ist und wir somit in einer Sekunde Ausführungszeit ungefähr tausend Taktzyklen simulieren

können. Bei einer Taktfrequenz von beispielsweise $f_c = 100\ \mathrm{MHz}$ und einer zu simulierenden Zeit von $T_S = 1\ \mathrm{s}$ ergibt sich nach Gleichung 1.1 eine Ausführungszeit von $T_A = T_S \times \frac{f_c}{f_{sim}} = 100.000\ \mathrm{s} = 27{,}8\ \mathrm{h}$. Wir müssen also mehr als einen Tag warten, bis die Simulation beendet ist, da das Verhältnis von Ausführungszeit zu simulierter Zeit 100.000 beträgt. An diesem – durchaus realistischen – Zahlenbeispiel ist nochmals deutlich erkennbar, dass RTL-Modelle nicht sinnvoll als virtuelle Prototypen für die Entwicklung der Systemarchitektur oder die Entwicklung von Software eingesetzt werden können.

Mit Modellen auf Transaktionsebene können Simulationsfrequenzen von $f_{sim} \approx 1\ \mathrm{MZyklen/s}$ und damit eine um den Faktor tausend höhere Simulationsleistung gegenüber RTL-Modellen erreicht werden. Somit verringert sich das Verhältnis von Ausführungszeit zu simulierter Zeit im obigen Zahlenbeispiel auf einen Wert von 100, so dass für eine Sekunde simulierter Zeit nur noch 100 Sekunden Ausführungszeit notwendig werden. Um hohe Simulationsleistungen für das Modell des Gesamtsystems erzielen zu können, müssen neben den TLM-Schnittstellen und TLM-Busmodellen auch die Komponentenmodelle im Hinblick auf die Ausführungszeiten optimiert werden. Dies betrifft in erster Linie die Modelle für die Mikroprozessoren, also die Instruktionssatzsimulatoren. Ein ISS muss den vom Compiler erzeugten Binärcode der Software ausführen. Dies kann beispielsweise dadurch erfolgen, dass man den Binärcode in einem C++-Modell des Prozessors interpretiert und die Befehle ausführt. Wesentlich höhere Simulationsleistungen können erzielt werden, indem man den Binärcode des zu simulierenden Prozessors in Befehle des Prozessors des Host-Rechners (beispielsweise ein x86-PC) während der Laufzeit übersetzt (engl.: dynamic binary translation, siehe beispielsweise [31]). In Verbindung mit leistungsfähigen Host-Rechnern können damit für 32-Bit RISC-Prozessormodelle Ausführungszeiten erreicht werden, welche in der Größenordnung der simulierten Zeit liegen oder sogar noch darunter. Die Modelle weisen Simulationsfrequenzen von einigen hundert MZyklen/s auf.

Um solche extrem hohen Simulationsleistungen für das Gesamtsystem zu erzielen, ist es bei einem Modell eines eingebetteten Systems, welches typischerweise auch sehr viele Peripherieeinheiten aufweist, auch notwendig, die Simulationsleistung der Peripherie-Modelle zu optimieren. Hierzu sollte man sich von zyklengenauen Modellen, wie sie für die RTL-Modellierung typisch sind, lösen und versuchen, die zeitlichen Abläufe abstrakter zu modellieren. Je nach Funktionalität einer Peripherieeinheit wird man aber möglicherweise um eine genauere zeitliche Modellierung, die nahe an einer Zyklengenauigkeit ist, nicht herumkommen. Es hängt daher auch von der Funktionalität des Modells ab, welche Simulationsleistungen erzielt werden können. Darüber hinaus muss auch auf eine effiziente Codierung der SystemC/C++-Modelle und eine optimierte Compilierung geachtet werden.

1.6 Bussysteme für SOCs

Wie wir schon in den vorangegangenen Abschnitten erwähnt haben, zielt die TL-Modellierung mit der SystemC-TLM-2.0-Bibliothek im Wesentlichen darauf ab, SOC-Bussysteme effizient und interoperabel modellieren zu können. Wir werden im Verlauf des Buches auf die TL-Modellierung mit der TLM-2.0-Bibliothek noch genauer eingehen. Aus diesem Grund möchten wir an dieser Stelle schon einige wesentliche Merkmale von SOC-Bussystemen darstellen, die wir für das Verständnis der TLM-2.0-Bibliothek benötigen. Die Erläuterungen illustrieren wir anhand des AMBA-Bussystems, welches von der Firma ARM entwickelt wurde und die typischen Eigenschaften eines SOC-Bussystems aufweist. Die sehr gut lesbare und verständli-

che Dokumentation des AMBA-Bussystems [4] kann von der Website der Firma ARM bezogen werden [5]; wir können an dieser Stelle keine vollständige Beschreibung des AMBA-Bussystems geben.

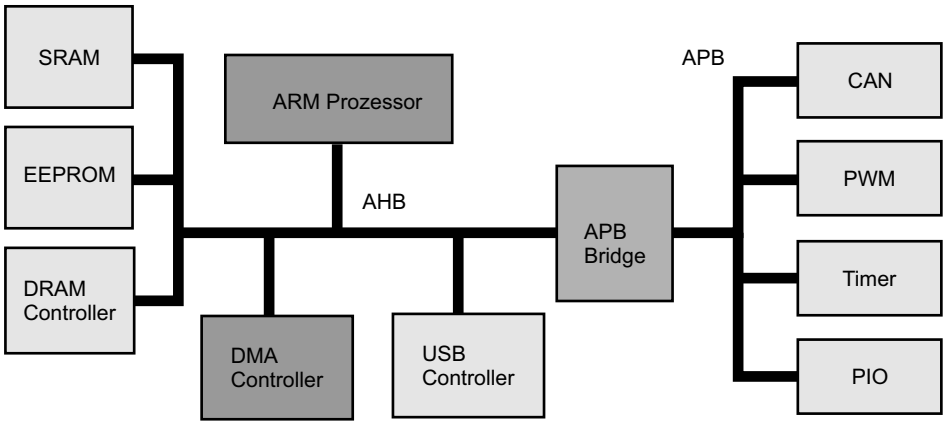

Abb. 1.6: *Beispiel für ein Prozessorsystem mit AMBA-Bussystem, bestehend aus AHB-Systembus und APB-Peripheriebus.*

Das Akronym AMBA steht für „Advanced Microcontroller Bus Architecture“ und der AMBA-Bus wurde von der Firma ARM entwickelt, um Systeme mit ARM-Mikroprozessoren aufbauen zu können. Das Bussystem ist auf SOCs zugeschnitten und daher ein typisches On-Chip-Bussystem. Viele Merkmale des AMBA-Bussystems wurden jedoch von Off-Chip-Bussystemen übernommen, wie beispielsweise dem aus den PCs bekannten PCI-Bussystem (Peripheral Component Interconnect).

Die AMBA-Spezifikation mit der Versionsnummer 2.0 aus dem Jahr 1999 [4] spezifiziert drei Teilbussysteme: AHB (Advanced High-performance Bus), ASB (Advanced System Bus) und APB (Advanced Peripheral Bus). Bei allen drei Teilbussen handelt es sich um synchrone Bussysteme, so dass die Transaktionen auf den Bussen synchron zu einem Bustakt stattfinden. Ferner handelt es sich um so genannte „Shared-Bus“-Systeme, so dass pro Zeitschritt genau eine Transaktion auf dem gemeinsamen Busmedium stattfinden kann. Da solche Systeme nur eine vergleichsweise geringe Busbandbreite aufweisen, wurden in späteren Erweiterungen des AMBA-Systems mit dem „Multi-Layer AHB“ und dem AXI-Protokoll (Advanced eXtensible Interface) die möglichen Bandbreiten erhöht. Wir werden darauf nicht eingehen, sondern beschränken uns auf die Diskussion des AHB-Busses.

Eine typische Konfiguration eines Systems mit dem AMBA-Bus zeigt Abbildung 1.6: Das gesamte Bussystem wird hier in die beiden Teilbusse AHB und APB aufgeteilt. Über den schnellen AHB-Systembus wird der ARM-Prozessor mit den On-Chip- und Off-Chip-Speichern sowie schnellen Peripherieeinheiten verbunden; langsamere Peripherieeinheiten werden über den APB-Bus und eine so genannte „Bridge“ mit dem AHB-Bus verbunden. Lese- oder Schreib-Transaktionen auf dem Bus können nur von einem Master ausgeführt werden, wobei ein Master durch eine Adresse einen Slave anspricht. Das AMBA-System ist speziell für 32-Bit-RISC-Prozessoren entwickelt worden, bei denen die Adressbreite 32 Bit beträgt und somit maximal

$2^{32}\,\mathrm{Byte} = 4\,\mathrm{GB}$ adressiert werden können. Die Adressen der Peripherieeinheiten liegen im Adressraum des Hauptspeichers (so genanntes „memory mapped I/O“). Die Datenbreite des Busses ist typischerweise ebenfalls 32 Bit, diese kann jedoch für eine höhere Busbandbreite erhöht werden.

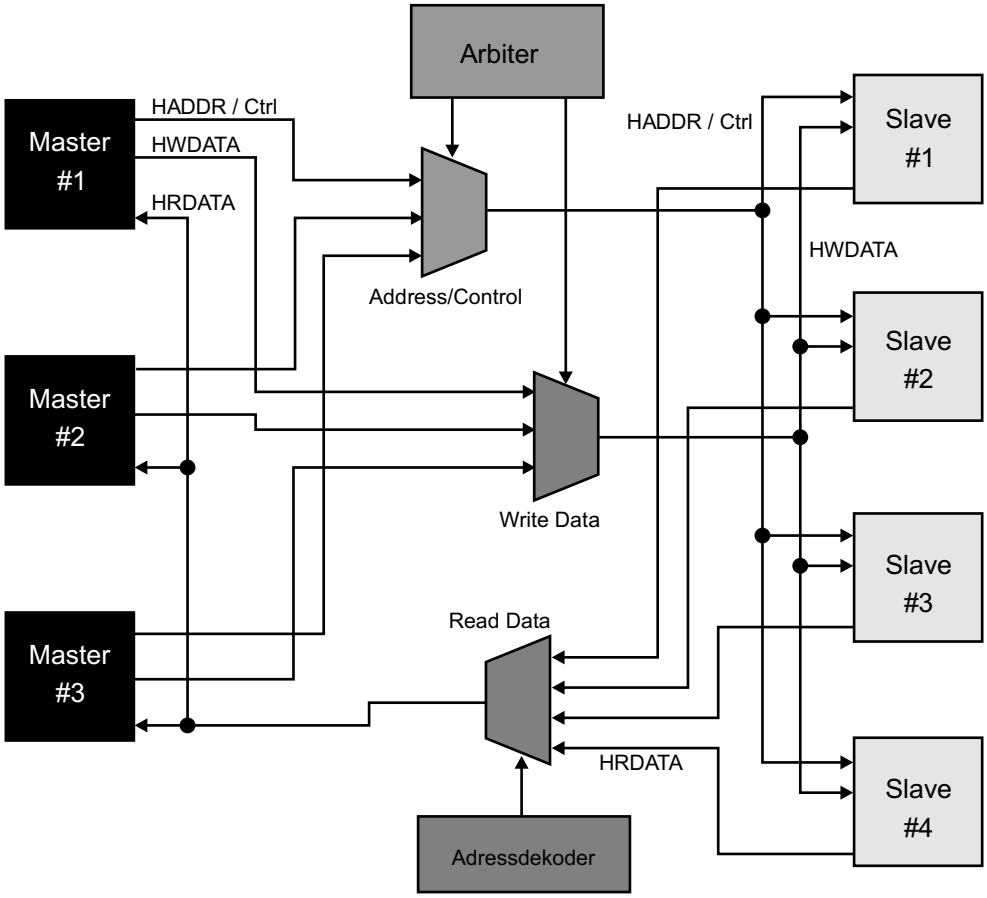

Abb. 1.7: *AHB-Bussystem. Die Abbildung zeigt schematisch die Multiplexer-Struktur des Bussystems (Quelle: ARM [4]). Die Adressen (HADDR) und Steuersignale (Ctrl) sowie die Schreibdaten (HWDATA) müssen über entsprechende Multiplexer, welche vom Arbiter gesteuert werden, auf die Slaves aufgeschaltet werden. Die Lesedaten (HRDATA) müssen über Multiplexer auf den Master aufgeschaltet werden. Die Abbildung zeigt nicht alle notwendigen Signale des AMBA-Busses, insbesondere nicht die einzelnen Steuersignale.*

Der AHB-Bus ist ein so genannter „Multi-Master“-Bus; dies bedeutet, dass mehr als nur ein Master Transaktionen initiieren kann. Im Beispiel von Abbildung 1.6 wäre beispielsweise eine DMA-Einheit (DMA: Direct Memory Access) als zweiter Bus-Master vorhanden. Für die Verwaltung mehrerer Master wird ein so genannter „Arbiter“ (dt.: Schiedsrichter) benötigt, welcher aus den Transaktionsanforderungen der verschiedenen Master den Master bestimmt,

welcher als nächstes den Zugriff auf den Bus erhält. Während Off-Chip-Bussysteme in der Regel als „Tristate-Busse“ implementiert werden, so werden On-Chip-Bussystemen zumeist als so genannte „Multiplexer-Busse“ ausgelegt. Abbildung 1.7 zeigt die Multiplexer-Struktur des AHB-Busses. Die einzelnen Bus-Master treiben ihre Adress- und Steuersignale und signalisieren dem Arbiter ihren Transaktionswunsch. Der Arbiter wählt einen Master aus und steuert die Multiplexer, so dass die Adressen, die Steuersignale und die vom Master zu schreibenden Daten auf die Slaves aufgeschaltet werden. Ein zentraler Adressdekoder dekodiert die vom Master gelieferte Adresse für die aktuelle Transaktion und erzeugt für den zugehörigen Slave ein Auswahlsignal. Des Weiteren steuert der Adressdekoder auch die Multiplexer, um die Lesedaten auf den Master aufzuschalten.

Bei einer Transaktion können Daten in verschiedenen Größen übertragen werden, üblicherweise 1, 2 oder 4 Byte; dies wird durch entsprechende Steuersignale vom Master angezeigt. Neben einzelnen Transaktionen (engl.: single transfer) auf dem Bus ist es auch möglich, eine Folge von Transaktionen mit aufeinander folgenden Adressen durchzuführen; dies wird als „Burst“ bezeichnet. Ein Burst besteht wiederum aus Einzel-Transaktionen, diese werden als „Beat“ bezeichnet. Die Adresse des nächsten Beats wird aus der Adresse des vorhergehenden Beats plus der Datengröße berechnet. Der AHB-Bus lässt Bursts zu, die aus 4, 8 oder 16 Beats bestehen.

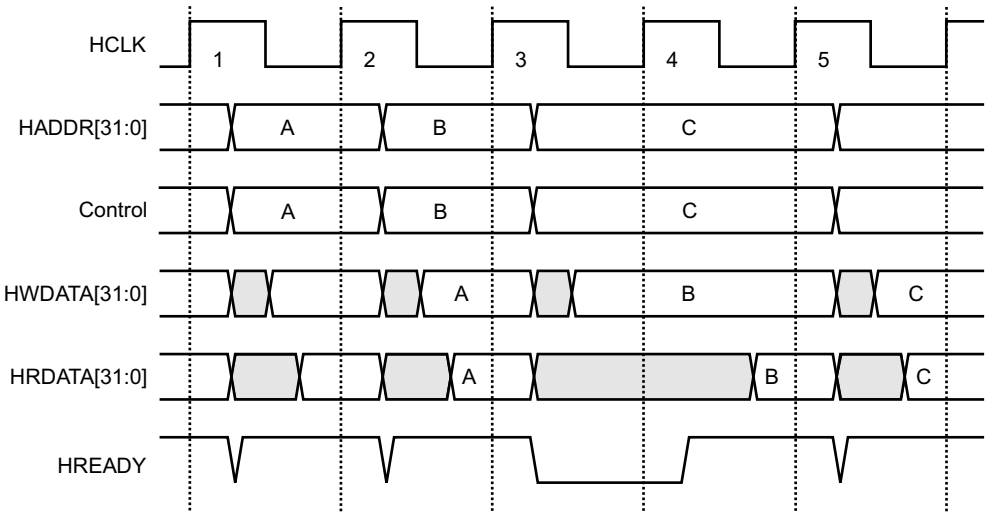

Abb. 1.8: *Timing-Diagramm für Einzel-Transaktionen auf dem AHB-Bussystem (vgl. auch [4]). Die Abbildung zeigt drei Transaktionen für die Adressen A, B und C. Für jede Transaktion sind jeweils die beiden Fälle einer Schreib- und einer Lesetransaktion im gleichen Diagramm gezeigt mit den zugehörigen Steuersignalen (Control) sowie den Schreib- (HWDATA) und Lesedaten (HRDATA). In einer realen Transaktion findet entweder eine Schreib- oder eine Lesetransaktion statt; dies wird durch die Steuersignale angezeigt. In den grauen Bereichen liegen keine gültigen Daten vor. Bei der Transaktion B wird vom Slave durch deaktivieren des HREADY-Signals ein „Wait-State“ eingefügt.*

Abbildung 1.8 zeigt eine Folge von drei Einzel-Transaktionen auf dem AHB-Bus. Grundsätzlich übernehmen sowohl die Master als auch die Slaves Daten mit der steigenden Taktflanke des Taktes HCLK. Der Bus arbeitet also wie eine synchrone Schaltung, wobei die steigende

Flanke die aktive Taktflanke ist. Die Einzel-Transaktionen oder die Beats eines Bursts werden so durchgeführt, dass sie mit der so genannten „Adress-Phase“ beginnen. In dieser Phase treibt der Master, welchem der Bus zugeteilt wurde, die Adresse und die entsprechenden Steuersignale nach der steigenden Taktflanke auf den Bus. Mit der nächsten steigenden Taktflanke muss der adressierte Slave die Adresse und die Steuersignale speichern. Im auf die Adress-Phase folgenden Taktzyklus beginnt die Daten-Phase der Transaktion. Handelt es sich um eine Schreib-Transaktion, so muss der Master nun die zu schreibenden Daten auf den Bus legen, welche der Slave mit der nächsten Taktflanke übernehmen kann. Im Fall einer Lese-Transaktion wird der Slave die vom Master zu lesenden Daten so auf den Bus legen, dass der Master diese mit der nächsten Taktflanke übernehmen kann. Der Slave zeigt durch das HREADY-Signal an, dass er die zu schreibenden Daten übernommen hat oder dass die zu lesenden Daten gültig sind.

Der AHB-Bus implementiert ein so genanntes „Pipelining“, um einen höheren Durchsatz zu erzielen. Dies bedeutet, dass die Adress-Phase der folgenden Transaktion sich mit der Daten-Phase der vorhergehenden Transaktion überlappt. Dieses Pipelining ist in Abbildung 1.8 anhand der drei Transaktionen gezeigt; wir identifizieren diese Transaktionen durch ihre (beliebigen) Adressen A, B und C. Im Taktzyklus 1 findet die Adress-Phase der Transaktion A statt. Taktzyklus 2 ist daher die Daten-Phase von Transaktion A und gleichzeitig auch die Adress-Phase von Transaktion B. Durch dieses Überlappen von Adress- und Daten-Phase kann ein Taktzyklus eingespart werden. Grundsätzlich ist es möglich, dass ein Slave nicht in der Lage ist, innerhalb von einem Taktzyklus Daten zu übernehmen oder Daten zu liefern. In diesem Fall hat der Slave die Möglichkeit, die Daten-Phase durch deaktivieren des HREADY-Signals zu verlängern. Dies wird als so genannter „Wait-State“ bezeichnet. Abbildung 1.8 zeigt diesen Fall für die Transaktion B: Der Slave deaktiviert im Taktzyklus 3 das HREADY-Signal und signalisiert so dem Master, dass er nicht in der Lage ist, die Daten-Phase von Transaktion B in diesem Taktzyklus zu beenden. Daraufhin muss nun der Master seinerseits die Adress-Phase von Transaktion C verlängern, bis der Slave im Taktzyklus 4 das HREADY-Signal aktiviert und somit dem Master das Ende der Daten-Phase von Transaktion B signalisiert. Der Master kann nun in Taktzyklus 5 die nächste Transaktion beginnen. Der Slave beendet die Daten-Phase von Transaktion C ohne „Wait-State“.

Die Spezifikation [4] des AMBA-Busses definiert das so genannte „Busprotokoll“. Wir konnten hier nur ansatzweise in aller Kürze einige grundlegende Regeln für die Durchführung von Transaktionen beschreiben; das gesamte Spezifikationsdokument umfasst 230 Seiten. Eine entsprechende Busschnittstelle für einen Master oder einen Slave muss die Protokoll-Regeln in entsprechender Hardware implementieren. Der eigentliche Bus besteht dann gemäß Abbildung 1.7 aus den Multiplexer-Strukturen, dem Arbiter und dem Adressdekoder, für welche ebenfalls Hardware zu implementieren ist. Für die Busschnittstellen und die Bus-Hardware werden daher entsprechende IP-Blöcke angeboten, so dass diese vom Anwender in der Regel nicht selbst entwickelt werden müssen.

1.7 Ziele und Aufbau des Buches

SystemC ist auf dem Weg, eine wesentliche Rolle im ESL-Design und in der Modellierung von elektronischen Systemen zu spielen. Wichtig für die Verbreitung von SystemC ist auch die Schulung von Entwicklern in der Industrie und von Studierenden an Hochschulen. Die erwähnte Methodik der Hardwareentwicklung auf RT-Ebene mit VHDL oder Verilog ist in der Industrie und auch in der Hochschulausbildung weit verbreitet – dies gilt aber nicht im gleichen Maße für SystemC und die Modellierung auf Transaktionsebene. Insbesondere die TL-Modellierung hat erst in jüngster Zeit Eingang in die Standardisierung gefunden. Es handelt sich also um ein recht neues Thema, welches ein gewisses Umdenken gegenüber der Entwicklung mit VHDL oder Verilog erfordert. Das vorliegende Buch setzt sich zum Ziel, eine verständliche Einführung in die Modellierung von digitalen Systemen mit SystemC nach dem IEEE Standard 1666-2011 zu geben. Ein spezieller Fokus soll dabei auf der stärkeren Abstraktion bei der Modellierung der Hardware auf Transaktionsebene liegen. Das vorliegende Buch ist allerdings kein Buch über die schon angesprochene ESL-Designmethodik. Im Fokus des Buches stehen die *Modellierung* auf Transaktionsebene und auch auf RT-Ebene mit SystemC und die *Simulation* dieser Modelle. Die in den vorangegangenen Abschnitten angesprochenen Entwurfswerkzeuge, wie High-Level-Synthese, und die Entwurfsmethodik des ESL-Design bleibt anderen Büchern vorbehalten. Ebenso werden wir auch nicht das Thema Verifikation und die SCV-Bibliothek für SystemC oder das Thema SystemC-AMS behandeln.

Das Buch soll in erster Linie ein Lehrbuch für SystemC sein, verbunden mit einer Einführung in den IEEE Standard 1666-2011. Erfahrungsgemäß lernt man am besten anhand von Beispielen, wobei die Beispiele einfach genug sein müssen, um verständlich zu sein. Wir werden daher die Modellierungstechniken von SystemC anhand von vielen Code-Beispielen illustrieren, die überschaubar und nachvollziehbar sind. Ein Lehrbuch muss in erster Linie gut verständlich sein und dem Leser einen anregenden Einstieg in ein neues Thema bieten. Man sollte sich auf die zentralen und möglicherweise auch schwierig zu verstehenden Probleme konzentrieren und auf nebensächliche Themen für einen besseren Überblick eher verzichten. Es ist daher eine schwierige Gratwanderung für ein Lehrbuch, auch gleichzeitig noch ein vollständiges Nachschlagewerk zu sein. Da der IEEE-Standard 1666-2011 (siehe [21]) selbst schon über 600 Seiten umfasst, ist es verständlich, dass dieses Buch – bei begrenztem Seitenumfang – nicht den kompletten Standard ausführlich darstellen und damit ein vollständiges Nachschlagewerk sein kann. Das Ziel war es, die aus Sicht des Autors wesentlichen Mechanismen der SystemC- und der TLM-2.0-Bibliothek detailliert und verständlich zu behandeln. Mit diesem erworbenen Wissen sollte es dem Leser möglich sein, anhand von zusätzlichem Material auch die nicht behandelten Inhalte der Bibliotheken zu verstehen. Empfehlenswert ist in diesem Zusammenhang beispielsweise die Doulos-Website [17]. Ferner kann auch das IEEE-Standard-Dokument zum 1666-Standard [21] als Nachschlagewerk und Referenz benutzt werden; es war dem Autor beim Schreiben des Buches auch häufig eine wertvolle Hilfe. Das Dokument kann über die Accellera-Website [1] bezogen werden.

Idealerweise bringt der Leser dieses Buches ein gewisses Verständnis für den „klassischen“ Entwurf von digitalen Systemen mit VHDL oder Verilog mit (siehe beispielsweise [18]). Ferner wäre es sinnvoll, wenn Programmierkenntnisse in C++ vorhanden wären. Für den Lernerfolg wichtig ist es, dass der Leser SystemC auch von der praktischen Seite kennen lernt. Es ist empfehlenswert, die Beispiele begleitend zur Lektüre zu kompilieren und zu simulieren und dann auch zu verändern oder zu erweitern. Ferner sind am Ende der Kapitel Kontrollfragen und

Übungsaufgaben angegeben, die der Überprüfung des Wissenstandes und der praktischen Arbeit mit SystemC dienen. Einige Quellcodes in den Beispielen wurden aus Platzgründen auf die wesentlichen Zeilen gekürzt; die vollständigen Quellcodes der Beispiele sowie die Lösungen der Übungsaufgaben können vom Autor oder über die Website des Verlags bezogen werden. In den im Buch abgedruckten Quellcode-Listings ist jeweils vermerkt, in welcher Datei der Quellcode zu finden ist.

Für die Simulation der Beispiele benutzen wir den in der SystemC-Bibliothek vorhandenen Simulator – außer einem C-Compiler werden keine weiteren Werkzeuge benötigt. Die Beispiele wurden mit Visual Studio 2008 unter Windows XP bzw. Windows 7 entwickelt; eine Anleitung für die Einrichtung von SystemC-Projekten und der Installation der Bibliotheken findet sich im Anhang des Buches. Wir gehen davon aus, dass die Beispiele auch unter Linux mit einem GNU-C++-Compiler übersetzt werden können, obgleich wir dies nicht geprüft haben. Eine Anleitung, wie die SystemC-Bibliotheken unter Linux installiert werden können findet sich in der Dokumentation der Bibliotheken. SystemC-Modelle können auch von speziellen Simulationswerkzeugen, wie beispielsweise „Modelsim“ von Mentor Graphics, ausgeführt werden. Obgleich dies vom Autor mit einzelnen Beispielen getestet wurde, so haben wir eine entsprechende Anleitung nicht in das Buch aufgenommen. Hierzu sei auf die Dokumentation der Werkzeuge verwiesen.

Da die Entwicklung von SystemC-Modellen bedeutet, in C++ zu programmieren, geben wir im zweiten Kapitel eine Zusammenfassung der wichtigsten Konstruktionen von C++, die für die Programmierung mit den SystemC-Bibliotheken wichtig sind. Dies ist zunächst das zentrale Konzept der Klassen und Objekte in der objektorientierten Programmierung und wie dies in C++ realisiert ist. Anschließend stellen wir die Strukturierung des Codes durch einen hierarchischen Klassenaufbau vor und stellen dem das Konzept der Vererbung in C++ gegenüber. Neben der Instanzierung von Objekten zur Laufzeit besprechen wir dann die virtuellen Funktionen und abstrakten Basisklassen; dies ist insbesondere eine wichtige Grundlage für die Modellierung auf Transaktionsebene. Schließlich werden noch Template-Klassen und die Übergabe von Funktionsargumenten durch Referenzen behandelt; von letzterem wird ebenfalls in TLM regen Gebrauch gemacht. Die Leser, welche über solide C++-Kenntnisse verfügen, können dieses Kapitel überspringen. Für Leser, die über wenige oder keine C++-Kenntnisse verfügen, kann das Kapitel eine Übersicht liefern, welche Kenntnisse für das Verständnis der folgenden Kapitel benötigt werden. Das zweite Kapitel kann natürlich kein C++-Lehrbuch ersetzen. Sollte der Leser über keine C++-Kenntnisse verfügen, so wäre zunächst das Erlernen von C++ sinnvoll, beispielsweise mit entsprechenden Lehrbüchern ([10], [20] oder [25]).

Obgleich die Schwerpunkte von SystemC eher in der Transaction-Level- oder System-Modellierung liegen, kann SystemC auch für die Modellierung auf Register-Transfer-Ebene genutzt werden und damit wie eine „klassische“ Hardwarebeschreibungssprache. Als Einstieg in SystemC haben wir daher im dritten Kapitel die Modellierung auf RT-Ebene gewählt, so dass sich der in VHDL oder Verilog erfahrene Leser zunächst auf vertrautem Terrain bewegen kann und sich mit dem C++-Wissen aus dem zweiten Kapitel leicht zurecht finden können sollte. Für Leser, die noch keine Erfahrungen in der RTL-Modellierung haben, sollte dieses Kapitel eine Übersicht darüber liefern können. Wir geben in diesem Kapitel auch einen Überblick über die SystemC-Bibliothek und zeigen dann, vergleichend zu einem VHDL-Modell, die RTL-Modellierung mit SystemC. Anschließend werden wir die wesentlichen Konstruktionen für die RTL-Modellierung wie Module, Prozesse, Ports und Signale noch etwas ausführlicher disku-

tieren. Wir zeigen auch anhand von Beispielen, was neben dem SystemC-Modell noch benötigt wird, um das Modell simulieren zu können. Dabei diskutieren wir auch die Modellierung der Zeit und wie wir Simulationsergebnisse betrachten können. Das Kapitel schließt dann mit einer Darstellung der von SystemC bereitgestellten „hardwarenahen“ Datentypen.

Im vierten Kapitel werden wir die SystemC-Ports, welche in der RTL-Modellierung der Verbindung von Modulen über Signale dienen, etwas genauer betrachten. Die Kanäle stellen eine Verallgemeinerung der auch aus VHDL oder Verilog bekannten Signale dar. Wir werden daher in diesem Kapitel die primitiven und hierarchischen Kanäle einführen und zeigen, was unter der Bindung von Ports und Kanälen zu verstehen ist. Sind Ports und Kanäle gebunden, so können aus einem Modul heraus über den Port so genannte Interface-Methoden des Kanals aufgerufen werden. Mit Hilfe dieser Interface-Methoden können dann beispielsweise Daten vom Modul an den Kanal übertragen werden. Die Mechanik der Ports mit ihren „Interfaces“ und den Bindungen an die Kanäle führt über das hinaus, was aus VHDL oder Verilog bekannt ist und ist auch eine wesentliche Grundlage für die Transaction-Level-Modellierung. Wir verlassen daher im vierten Kapitel schon die RT-Ebene, so dass auch die Beispiele schon deutlich abstrakter sein werden als im dritten Kapitel. Ergänzend werden auch hierarchische und mehrfache Bindungen und die für hierarchische Bindungen notwendigen Exports besprochen.

Um fehlerfreie SystemC-Modelle schreiben zu können, ist ein Verständnis der Arbeitsweise des SystemC-Simulators unerlässlich. Dies gilt sowohl für RTL-Modelle und in noch stärkerem Maß für Transaction-Level-Modelle. Der SystemC-Simulator ist die zentrale Instanz, welche für die Ausführung der Prozesse verantwortlich ist. Wir möchten daher im fünften Kapitel den Simulationsalgorithmus und die damit zusammenhängenden Mechanismen darstellen. In diesem Kontext sollen dann auch die verschiedenen Möglichkeiten der Steuerung der Prozessausführung durch statische oder dynamische Sensitivitäten und mit Hilfe von Ereignisobjekten besprochen werden. Für die Ereignisobjekte gibt es keine Entsprechung in der RTL-Modellierung mit VHDL oder Verilog, dort sind Ereignisse immer an Signale gebunden. Die Ereignisobjekte werden im Wesentlichen für die abstraktere Modellierung auf Transaktions- und Systemebene benötigt, da man dort keine Signale mehr benutzt. Spezielle Themen im Zusammenhang mit der Simulation und der Ausführung von Prozessen sind „Event Finder“, „Event Queues“ oder dynamische Prozesse. Des Weiteren ist es für den Anwender möglich, so genannte „Callback-Funktionen“ zu schreiben, welche vom Simulator aufgerufen werden. Die Ausgabe von Fehler- und Diagnosemeldungen kann mit dem SystemC-Report-Handler erfolgen, welcher in diesem Kapitel ebenfalls besprochen wird.

Das sechste Kapitel beschäftigt sich mit den Grundlagen der Transaction-Level-Modellierung. Wir werden zunächst, ohne Benutzung der TLM-2.0-Bibliothek, anhand der Modellierung eines einfachen Bussystems zeigen, was die TL-Modellierung im Wesentlichen von der RTL-Modellierung unterscheidet. Anschließend geben wir eine Übersicht über die TLM-2.0-Bibliothek. Der Kern dieser Bibliothek besteht aus den Sockets, die eine Erweiterung der Ports darstellen, und zugehörigen Interface-Methoden. Durch Bindung von Sockets von Modulen oder Kanälen können wiederum die Interface-Methoden über die Sockets aufgerufen werden. Für den Transport von Daten bei der Modellierung von Bustransaktionen wird den Interface-Methoden ein so genanntes Transaktionsobjekt übergeben. Neben den zu transportierenden Daten sind in diesem Transaktionsobjekt weitere Merkmale der Transaktion gespeichert, wie beispielsweise die Startadresse oder die Datenlänge. Wir stellen in diesem Kapitel die Sockets und die Interface-Methoden sowie das Transaktionsobjekt der TLM-2.0-Bibliothek vor. Anhand ei-

nes einfachen Beispielsystems, bestehend aus einem Mikroprozessor, Speicher und einer Peripherieeinheit zeigen wir den praktischen Einsatz der TLM-2.0-Bibliothek. Ferner wird gezeigt, wie man die Simulationsleistung ermitteln kann.

Im siebten Kapitel gehen wir genauer auf die Modellierung der zeitlichen Abläufe auf der Transaktionsebene ein. In der TLM-2.0-Bibliothek werden hierzu die beiden Modellierungstile „Loosely-Timed“ (LT) und „Approximately-Timed“ (AT) vorgeschlagen. Der LT-Stil, den wir als ersten besprechen, modelliert den zeitlichen Ablauf einer Transaktion sehr einfach und ermöglicht eine zeitliche Entkopplung von Prozessen, so dass eine höhere Simulationsleistung erzielt werden kann. Mit dem AT-Stil kann der zeitliche Ablauf von Transaktionen genauer modelliert werden, so dass beispielsweise auch das im Abschnitt 1.6 angesprochene Pipelining modelliert werden kann. Allerdings erfordert der AT-Stil einen höheren Simulationsaufwand. Im Zusammenhang mit dem AT-Stil werden wir noch die Verwendung eines „Memory-Managers“ für die Transaktionsobjekte und von „Payload Event Queues“ besprechen.

Noch ein Hinweis zum Schluss: Obgleich dieses Buch in deutscher Sprache geschrieben ist, benutzen wir für viele Begriffe die englischen Bezeichnungen. Zum einen lassen sich viele technische Begriffe nur sperrig in die deutsche Sprache übersetzen und zum anderen ist dann im Vergleich mit der umfangreichen englischsprachigen Literatur eher klar, was gemeint ist.