Beschreibungssprachen für den Entwurf eingebetteter Systeme

1. Bastian Schneider
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Deutschland
bastian.schneider@hs-furtwangen.de

2. Marius Stuber
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Deutschland
marius.stuber@hs-furtwangen.de

3. David Wichmann
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Deutschland
david.wichmann@hs-furtwangen.de

***Abstract*—In den meisten technischen Systemen finden sich eingebettete Systeme zur Regelung und Steuerung sich wiederholender Aufgaben. Für die Beschreibung eines solchen Systems, in welchem Soft- und Hardware eng zusammenarbeiten müssen, eignen sich herkömmliche Programmiersprachen nur bedingt. In diesem Artikel werden einige geläufige Beschreibungssprachen zum Entwurf solcher eingebetteter Systeme sowie deren Funktionsweise genauer beschrieben. Hierbei wird auch auf die unterschiedlichen Abstraktionsebenen beim Entwurf Bezug genommen, mit denen eine zunehmende Hardwarenähe erzielt werden kann.**

***Schlüsselworte*—Eingebettete Systeme, Beschreibungssprache, Hardwarebeschreibung, Systembeschreibung, VHDL, Verilog, SystemVerilog, SystemC**

I. Einleitung

Eingebettete Systeme finden sich in einer Vielzahl technischer Systeme wieder, in welchen sie sich häufig wiederholende Regelaufgaben übernehmen. Mit eingebetteten Systemen wird zum Beispiel die Geschwindigkeitsregelanlage in Fahrzeugen oder die Steuerung elektrischer Küchengeräte realisiert. Es können auch mehrere Aufgaben gleichzeitig von einem System gesteuert werden. So werden in einer Smartwatch verschiedene Aufgaben, wie unter anderem die Pulsmessung und digitale Uhrzeitanzeige, über ein eingebettetes System umgesetzt. In der Industrie finden eingebettete Systeme sogar bei der Steuerung von Industrierobotern Anwendung, um Arbeitsprozesse zu automatisieren.

A. Funktionsweise eingebetteter Systeme

Eingebettete Systeme sind Computersysteme im Kontext eines technischen Systems. Sie erfüllen hierbei dedizierte Aufgaben – meist Regel-, Steuerungs- oder Überwachungsaufgaben – und kommunizieren über Schnittstellen mit dem übergeordneten System und dem Nutzer [1]. Diese Schnittstellen belaufen sich dabei auf Eingänge für Sensoren und Ausgaben an Aktoren sowie Ein- und Ausgänge zur Kommunikation mit dem Nutzer.

Abbildung 1 veranschaulicht diese Schnittstellen und deren Rolle im eingebetteten System. Die Signale der Sensoren müssen eventuell vor Verarbeitung über Analog-Digital-Wandler in für den Prozessor nutzbare Signale umgewandelt werden. Selbiges gilt auch in umgekehrter Richtung für die Ausgabe an Aktoren [1], [2]. Zusätzlich können Eingaben eines Nutzers an den Prozessor des Systems übermittelt werden.

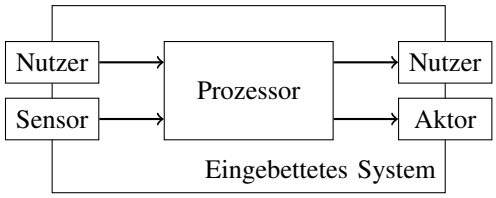

Abbildung 1. Schnittstellen eines eingebetteten Systems

Des Weiteren spielt in einem eingebetteten System die zeitliche Komponente eine wichtige Rolle. Reagiert etwa ein Geschwindigkeitsregler zu langsam auf einen Bremsvorgang des Nutzers, welcher die Regelung abschalten soll, kann dies für den Nutzer unter Umständen lebensgefährlich sein, wenn der Geschwindigkeitsregler weiter das Tempo hält. Die Relevanz der schnellen Verarbeitung lässt sich gut am Beispiel eines anderen eingebetteten Systems im selben technischen Umfeld verdeutlichen. Registriert ein Aufprallsensor am Auto eine Kollision, muss das Airbagsystem im Bruchteil einer Sekunde entscheiden, ob der Airbag ausgelöst werden soll oder nicht, da selbst eine Verzögerung von 10 Millisekunden bei einem Unfall lebensbedrohliche Folgen haben kann [3]. Außerdem muss die Entscheidung zuverlässig sein, da ein fehlerhaftes Auslösen während der Fahrt zu einem Unfall führen kann.

II. Beschreibungssprachen für eingebettete Systeme

Auf die besonderen Bedingungen, die für eingebettete Systeme gelten, wie die Relevanz des Zeitverhaltens, kann mit regulären Programmiersprachen nicht Bezug genommen werden [1]. Aus diesem Grund wurden diverse, hardwarenähere Sprachen entwickelt, welche sich für den Entwurf eingebetteter Systeme besonders eignen.

Hierbei gibt es jedoch auch Unterschiede bei den verschiedenen Beschreibungssprachen entsprechend ihrer Anwendungsbereiche. Die Nähe des Entwurfs zur Hardware spielt bei der Auswahl der Sprache eine große Rolle.

A. Abstraktionsebenen der Beschreibungssprachen

Für die Beschreibung von Hardwaresystemen haben sich im Lauf der Jahre verschiedene Abstraktionsebenen entwickelt. Während in den 1970er Jahren Schaltkreise durch explizite Transistorbeschreibungen entworfen wurden, ist bis zu den 2000er Jahren die Abstraktion stetig gestiegen, was die Produktivität der Entwicklung erhöht hat [1]. So hat sich erst durch Zusammenfassen von Transistoren zu Logikblöcken die Gatter-/Logik-Ebene entwickelt. Darauf folgend wurden diese Blöcke zu Bausteinen höherer Abstraktion zusammengefasst und es hat sich die Beschreibung auf der Register-Transfer-Ebene (engl. Register-Transfer-Layer, RTL) entwickelt. Zu Beginn der 2000er Jahre haben sich daraufhin Hardware- und System-Beschreibungssprachen entwickelt [1].

Konkret lassen sich fünf Ebenen mit steigender Abstraktion aufteilen, welche bei der Entwicklung solcher Systeme Verwendung finden: Die Transistor-Ebene, die Logik-Ebene, die RT-Ebene, die algorithmische Ebene und die System-Ebene.

- Transistor-Ebene
  Auf der Transistor-Ebene wird die Hardware mit elektronischen Bauelementen beschrieben. Hier werden Bausteine wie die dieser Ebene namensgebenden Transistoren aber auch Kondensatoren oder Widerstände verwendet, um eine elektronische Schaltung zu erzeugen [4]. Dies erfordert bei größeren Schaltungen einen sehr großen Aufwand, bietet jedoch auch durch die höchste Hardwarenähe der Abstraktionsebenen die beste Möglichkeit zur manuellen Optimierung.
- Logik-Ebene
  Auf der Logik-Ebene werden Logikgatter durch Kombination logischer Bauteile erstellt. Diese dienen dazu, logische Eingangssignale in weitere logische Ausgangssignale umzuwandeln. Gegenüber der Transistor-Ebene können die Signale auf dieser Ebene nicht mehr beliebige Werte annehmen, sondern sind an einen vordefinierten Wertebereich (zum Beispiel low, high, undefined) gebunden [4].
  Beschrieben werden können Systeme auf dieser Ebene auf zwei verschiedene Arten. Durch die Beschreibung der Struktur des Endsystems können direkt die Logikbauteile dieser Ebene zusammengeschaltet werden, indem man die Verbindungen der Ein- und Ausgänge beschreibt. Demhingegen kann auch das Verhalten dieser Ebene beschrieben werden, indem man die Funktion zum Beispiel als eine boolesche Gleichung darstellt, entsprechend der Form „a = b or (c and d)“ [4]. Diese Verhaltensbeschreibung lässt sich dann durch geeignete Synthese-Werkzeuge in einen Strukturplan umwandeln, welcher in einer Schaltung realisiert werden kann.
- RT-Ebene
  Wie der Name Register-Transfer-Ebene andeutet, wird auf dieser Ebene der Transfer und die Verarbeitung von Daten zwischen Registern beschrieben. Hierbei werden als elementare Bausteine Speicherelemente wie Register oder verarbeitende Elemente wie arithmetisch-logische Einheiten (ALUs) zusammengesetzt und verbunden [5].
  Während, wie auch auf der Logikebene, die Struktur durch genaue Beschreibung und Verknüpfung der Bauelemente beschrieben werden kann, wird das Verhalten eines Systems auf dieser Ebene zum Beispiel in Form eines Zustandsautomaten beschrieben [4].
- Algorithmische Ebene
  Bei der Beschreibung auf algorithmischer Ebene liegt eine klare Distanz zur Hardwarerealisierung vor, da hier kein Bezug auf die Struktur dieser genommen wird [4]. Das System wird durch einen oder mehrere parallele Algorithmen (zum Beispiel aus Funktionen) beschrieben. Durch Beschreibung von Blöcken und deren Kommunikation durch Signale kann eine Strukturbeschreibung erstellt werden. Für die Verhaltensbeschreibung werden demhingegen übliche Operationen aus Variablen und Operatoren verwendet [4].
- System-Ebene
  Auf der höchsten Abstraktionsebene, der System-Ebene, werden Systeme als Ganzes unter Verwendung von zum Beispiel Speicher- und Prozessor-Einheiten sowie deren Verbindung über Busse beschrieben [5].
  Die Struktur eines Systems auf System-Ebene kann zum Beispiel durch Verwendung eines Blockdiagramms beschrieben werden [5]. Die Bausteine für diese Beschreibung sind zum Beispiel die oben genannten Speicher- und Prozessoreinheiten sowie Busse. Zur Verhaltensbeschreibung auf System-Ebene können die einzelnen Prozesse des Systems in einer herkömmlichen Programmiersprache (wie C oder C++) implementiert werden [5]. Um hierbei die Parallelität der verschiedenen Prozesse des Systems zu bewahren, muss für die Verhaltensbeschreibung eine Möglichkeit der Nebenläufigkeit und der Kommunikation zwischen den Prozessen gegeben werden [5].

Die Bauteile jeder dieser Abstraktionsebenen setzen sich jeweils aus einer Kombination von Bauteilen der nächstniedrigeren Abstraktionsstufe zusammen. Diese Entwicklung lässt sich beispielhaft an den Schaltplänen und den daraus resultierenden Komponenten in Abbildung 2 zeigen. Der obere Teil zeigt eine Schaltung auf Transistorebene und wie diese auf der Logik-Ebene zu einem einzelnen Baustein (Und-Gatter) zusammengefasst wird. Darunter werden entsprechend Bausteine der Logik-Ebene zu einem Baustein der nächsthöheren Abstraktionsebene, der RT-Ebene, zusammengefasst.

Wie in Abbildung 2 zu sehen, nimmt der Beschreibungsaufwand mit steigender Abstraktion ab. Abbildung 3 zeigt, wie sich diese Abstraktionsebenen praktisch auf die Entwicklung von Hardwaresystemen auswirken. Die Beschreibung auf höherer Ebene reduziert den Entwicklungsaufwand von Hardwaresystemen beachtlich. Bei stetig komplexer werdenden Systemen ist eine abstraktere Beschreibung fast unvermeidbar. Beim Anstieg der Komplexität ist die volle Beschreibung auf RT- oder Gatter-Ebene mit extremem Aufwand verbunden,

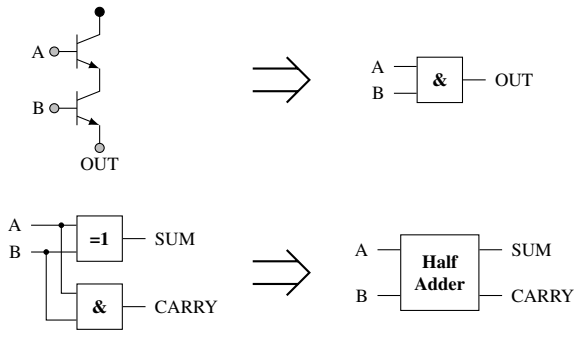

Abbildung 2. Zusammenfassung von Bausteinen auf höheren Abstraktionsebenen

weshalb auf eine höhere Abstraktion ausgewichen werden sollte.

Da jedoch, wie bereits erwähnt, eine abstraktere Beschreibung auch performanzbezogene Nachteile mit sich bringt, ist eine Entwicklung auf hardwarenaher Ebene weiterhin unumgänglich. Folgend werden häufig verwendete Beschreibungssprachen für den Entwurf eingebetteter Systeme vorgestellt, welche die Beschreibung über unterschiedliche, in Abbildung 4 dargestellte Abstraktionsebenen ermöglichen.

III. VHDL

VHDL ist eine Hardwarbeschreibungssprache, ausgelegt auf Entwicklung von Software nahe eines Hardwareentwurfs [7]. Aus diesem Grund unterscheidet sich VHDL von Programmiersprachen wie C, C++, Java oder Pascal erheblich. Sie ist darauf ausgelegt, durch Beschreibung und Kombination paralleler Schaltungsteile nebenläufige Prozesse abzubilden, diese als Schaltung umzusetzen und zu simulieren [7]. Erreicht wird dies durch die Kombination einzelner Bauteile zu Schaltungen.

VHDL wurde 1980 vom U.S. Department of Defense in Auftrag gegeben, um einen größerwerdenden technologischen Rückstand in der Entwicklung von sehr schnellen, integrierten Schaltkreisen (engl. Very High Speed Integrated Circuits)

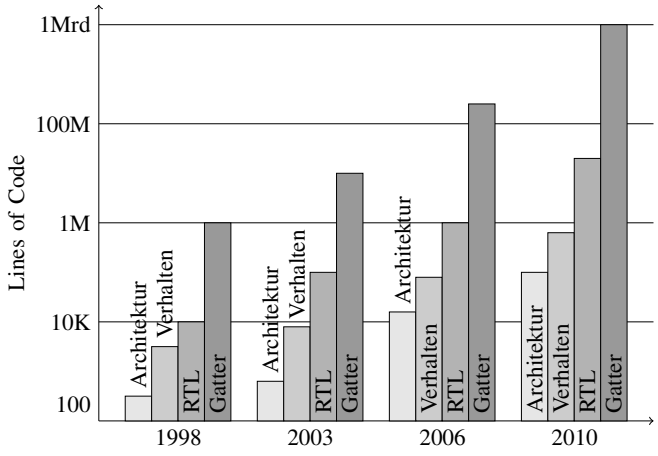

Abbildung 3. Umfang verschiedener Beschreibungen für ein mittelgroßes System (nach [1])

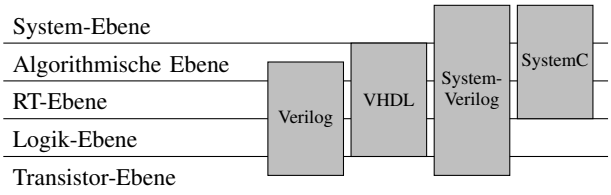

Abbildung 4. Abstraktionsebenen verschiedener Sprachen (nach [6])

gegenüber der Privatwirtschaft aufzuholen [8]. Diese Technologie war für das Department of Defense vorallem für die Entwicklung modernerer Waffensysteme von Bedeutung, weil in diesen immer mehr elektronische Subsysteme eingesetzt wurden, um die Effektivität, Reaktionsgeschwindigkeit und Anpassungsfähigkeit zu verbessern [8].

A. Einsatzgebiet

VHDL soll in allen Phasen einer Schaltungsentwicklung eingesetzt werden können. Dazu zählt die Planung, Entwicklung, Verifikation, Synthese und das Testen der Entwicklung sowie deren Wartung und Erweiterung. [9]

Als Teil der Schaltungsentwicklung wird VHDL außerdem oft für die Beschreibung von FPGAs eingesetzt [7]. FPGA steht für Field Programmable Gate Array. Eine besondere Art von Schaltkreis, in den eine logische Schaltung geladen werden kann.

Für die Unterstützung der Planung besteht außerdem die Möglichkeit, Zustandsdiagramme in VHDL zu übersetzen. So können diese unter anderem für die Abbildung von endlichen Automaten genutzt werden, welche sich wiederum für die Darstellung steuerungsdominanter Systeme eignen [7]. VHDL ist somit theoretisch in der Lage, aus Zustandsdiagrammen direkt Hardware zu erzeugen.

Weil es in solchen Diagrammen allerdings schwer ist, verteilte Systeme darzustellen und der aus ihnen erzeugte Programmcode in der Praxis selten überzeugt, ist die Effizienz dieser Funktion eher eingeschränkt [7].

Für die Verhaltensbeschreibung auf der Logik-Ebene wurden VHDL Synthese-Werkzeuge erstellt, die ermöglichen, diese von einer höheren Abstratktionsebene zu synthetisieren. Diese Möglichkeit wurde unter anderem geschaffen, weil die Logik-Ebene aufgrund ihrer Hardwarenähe nicht für jeden Programmierer leicht zu erfassen ist [7]. Die Beschreibung auf Logik-Ebene ist somit nicht mehr erforderlich. Weil sich diese für einen Hardware-Entwickler allerdings besser erschließt als eine abstraktere Ebene, ist eine Verhaltensbeschreibung auf Logik-Ebene auch weiterhin nützlich und wird von VHDL unterstützt [7].

B. Kommunikation und Verarbeitung

Weil der Anwendungsbereich von VHDL in der Schaltungsentwicklung liegt, wird in solchen Schaltungen, anders als in Programmiersprachen, für die Übermittlung von Daten zwischen Bauteilen nicht von Werten, sondern von Signalen gesprochen [7].

Neben den Signalen gibt es auch Variablen und Konstanten als Objektklassen. Der Wert einer Variablen kann sich während der Ausführung verändern. Wird die Variable gelesen, kann nur ihr aktueller Wert zurückgegeben werden. Mögliche ältere Werte werden überschrieben [7]. Während Signale zu verschiedenen Zeitpunkten auch verschiedene Typen annehmen können, können Variablen nur Werte des Wertebereichs ihres Typs annehmen. [7]. Variablen sind somit, im Gegensatz zu Signalen, typisiert. Demhingegen nehmen Konstanten nur einen festen Wert eines Wertebereichs ihres Typs an [9].

Für die Datenübermittlung in einem Ausdruck gelten für VHDL andere Bedingungen als für Programmiersprachen. Während in anderen Sprachen Zuweisungen auf bereits genutzte Variablen erneut durchgeführt werden können, ist dies bei einer Hardwarelösung und somit auch bei VHDL nicht möglich. Grund dafür ist das Fehlen eines Registers, das die Variable zwischenspeichern würde [7]. Eine erneute Zuweisung würde aus Sicht einer Hardwareumsetzung in einer Rückkopplung des Ausgangs auf den Eingang der Schaltung resultieren, daher unterbindet VHDL dies [7].

Für gewöhnlich wird in VHDL nebenläufig programmiert [7]. Es bietet allerdings ebenfalls die Möglichkeit, Programmcode squenziell auszuführen. Dazu wird die Process-Anweisung verwendet. Innerhalb dieser Anweisungen wird Code nicht nebenläufig, sondern sequenziell ausgeführt [9]. Eine solche Anweisung entspricht der Verhaltensbeschreibung einer Schaltung und wird deshalb innerhalb einer Architektur (siehe Unterabschnitt III-E) verwendet.

C. Datentypen und Operationen

Wie in Programmiersprachen gibt es auch in VHDL vordefinierte Typen mit zugehörigen Wertebereichen. Neben Typen wie boolean, integer, character und string besitzt VHDL auch Typen, die besonders in der Schaltungsentwicklung wichtige Rollen einnehmen. So gibt es den Typ BIT, der aus den Werten 0 und 1 besteht oder den Typ BIT_VECTOR, der aus einer Sequenz von Nullen und Einsen variabler Länge besteht [9].

Es stehen ebenfalls Operationen bereit, ähnlich denen von Programmiersprachen. So können in VHDL auch Schleifen, Verzweigungen oder Fallunterscheidungen in einer gewohnten Syntax genutzt und zusätzlich auch auf Signale angewendet werden [9].

D. Bausteine der Systeme und Hierarchie

Code kann in VHDL auf der algoritmischen-, der Register Transfer- oder der Logik-Ebene beschrieben werden.

Auf der algorithmischen Ebene werden Anweisungen mit Hilfe von Konstrukten der Verhaltensmodellierung beschrieben, die keinen Zusammenhang mit der späteren Schaltungsstruktur besitzten. Eine Beschreibung enthält somit keine Hinweise auf die Schaltungsstruktur in die sie später eingebunden wird und auch keine Takt- oder Rücksetzsignale [4].

Möchte man in VHDL eine Schaltung auf RT-Ebene beschreiben so kann die Beschreibung der algorithmischen Ebene um ein Taktsignal in Form einer clock und gegebenenfalls ein Rücksetzsignal ergänzt werden. Somit ist der Unterschied zu der algorithmischen Ebene, dass ein zeitliches Schema für den Ablauf definiert wird [4]. VHDL nutzt auf dieser Ebene außerdem Funktionen mit denen Zeiten genau verwaltet, Störimpulse unterdrückt und setup und hold constraints definiert werden können [7].

Auf der Logik-Ebene wird in VHDL eine Schaltung durch die Verknüpfung logischer, digitaler Signale und ihres zeitlichen Verhaltens beschrieben [7]. Dabei wird die Verknüpfung mit logischen Operatoren geschaffen. Diese Beschreibung ist aufgrund ihres niedrigen Abstraktionsniveaus sehr hardwarenah. Dadurch ist es in VHDL möglich, verschiedene Datenflüsse zu beschreiben und Nebenläufigkeiten zu behandeln, wie es mit keiner der gebräuchlichen Programmiersprachen möglich wäre [7].

E. Schaltungsaufbau mit VHDL

Eine Schaltung wird in VHDL mit den Beschreibungsteilen Entity, Architecture, Component und Package modelliert.

In einer Entity werden die Ein- und Ausgänge der Schaltung beschrieben. Es können somit eingehende Signale angelegt und ausgehende Signale beobachtet werden [7]. Die Beschreibung entspricht den Ein- und Ausgängen der tatsächlichen Hardware und ist somit die höchste hierarchische Stufe. Jeder Entity können mehrere Architectures zugeordnet werden [7].

In einer Architecture wird die Funktion der Schaltung definiert. Die Funktion kann hierbei auf verschiedene Arten, auch miteinander vermischt, beschrieben werden. Es ist somit möglich eine Schaltung vollständig aus einer Datenfluss-, Verhaltens- oder Strukturbeschreibung oder aus Teilen in verschiedenen Beschreibungen anzulegen [7]. Ein Vermischen der Beschreibungsform kann allerdings schnell verwirrend wirken und sollte daher mit Bedacht eingesetzt werden [7].

Eine fertige Architecture mit zugewiesener Entity kann auch in einer weiteren Architecture verwendet werden. Eine so verwendete Schaltung wird dann Component genannt [9]. Dabei werden die Ein- und Ausgänge innerhalb der Architecture, in der die Component verwendet werden soll, mit Signalen verbunden.

In Packages können verschiedene Elemente wie Components, geteilte Variablen oder Konstanten zusammengefasst und einem anderen Projekt verfügbar gemacht werden. Alle Elemente, welche in Pakete aufgenommen werden können sind im IEEE Standard [9] aufgelistet.

F. Simulation mit VHDL

Bei der Simulation wird in VHDL eine Schaltung mit einer vorgegebenen Taktrate simuliert. Dabei werden alle Reaktionen auf Änderungen von Ein- und Ausgängen während eines Taktes ermittelt. Signale, die in der Beschreibung explizit deklariert werden, werden für die Simulation in Variablen geschrieben, die den Wert des Signals zu einem bestimmten Zeitpunkt entsprechen. Wann immer das Signal während der Simulation aufgerufen wird, wird somit stattdessen der Wert der Variable übermittelt [9]. Damit das Verhalten durch diese Änderungen im Bezug auf die Nebenläufigkeit korrekt dargestellt werden kann, wird dabei auch der Zeitverlauf

innerhalb des Taktes simuliert [4]. Treten Ereignisse durch die Nebenläufigkeit gleichzeitig auf, wird von VHDL, wie im Betrieb, auch in der Simulation keine bestimmte Reihenfolge beachtet, da dies im Betrieb ebenfalls nicht berücksichtigt werden kann und somit eine Reihenfolge nicht den tatsächlichen Ablauf simulieren würde. [9]. VHDL bietet für die Simulation außerdem Funktionen, die aus Debuggern geläufiger Programmiersprachen bekannt sind. So können Breakpoints gesetzt werden und Werte von Signalen und Variablen während den Schritten der Simulation überwacht werden.

G. Synthese von VHDL-Code

VHDL-Code der auf der RT-Ebene beschrieben wird kann in eine Netzliste auf Logik-Ebene umgewandelt werden. Dazu wird ein externes Synthese-Werkzeug und technologiespezifische Gatterbibliotheken benötigt, weil sich die Synthese abhängig von der einzusetzenden Hardware verändert [4]. In diesen Bibliotheken sind unter anderem technische Informationen über die Fläche, Laufzeit und Verlustleistung der Hardware hinterlegt.

In VHDL gibt es Konstrukte, die hardwaretechnisch nicht umgesetzt werden können. Anders als bei der Simulation, können diese Konstrukte nicht synthetisiert werden und müssen vor einer Synthese gegebenenfalls ersetzt werden [4]. Die Synthese ist ebenfalls dahingehend eingeschränkt, dass nur Schaltungen synthetisiert werden können, für die eine Gatterbibliothek verfügbar ist.

IV. SystemC

SystemC ist, anders als VHDL oder Verilog, keine eigene Sprache, sondern eine Klassenbibliothek für C++. Durch die Integrierung von Hardwarebeschreibung in eine Programmiersprache ermöglicht SystemC eine Modellierung eines gesamten Systems auf Software- sowie Hardware-Seite und hierbei auch auf verschiedenen Abstraktionsebenen. [10]. Es bietet Möglichkeiten der Modellierung von der System- bis zur RT-Ebene.

SystemC wurde 1999 von Mitarbeitern verschiedener Firmen sowie der University of California, Irvine entwickelt, welche sich ein Jahr später zu der Open SystemC Iniviative (OSCI) zusammenschlossen [7]. Seit 2005 ist SystemC durch IEEE im Standard 1666-2005 oder seit 2012 unter der neuen Revision 1666-2011 [10] standardisiert.

Durch die Verwendung der SystemC-Klassenbibliothek wird C++ zum Beispiel um eine hierarchische Aufteilung des Systems in Module, einen Simulationskern, hardwarebezogene Datentypen, Scheduling und Synchronisierung von Events sowie Kommunikationskanäle erweitert [6], [10].

A. Datentypen

In der SystemC-Klassenbibliothek finden sich verschiedene Datentypen, welche zur Modellierung von Hardware benötigt werden. So werden zum Beispiel Integer-Typen mit einer eingeschränkten Genauigkeit (sc_int, sc_uint), Festkommazahlen (sc_fixed, sc_ufixed), ein vierwertiger Logik-Typ (sc_logic) und zwei- und vierwertige Logik-Vektoren (sc_lv, sc_bv) mitgeliefert [10]. Diese Typen haben keine äquivalenten Datentypen in C++ selbst, da zum Beispiel für hardwarenahe Logikoperationen die Fälle 0 und 1 (entsprechend bool) nicht ausreichend sind und um die Werte Hochohmig und Unbekannt erweitert werden müssen [6]. Aus diesem Grund wurde diese Auswahl an Datentypen in SystemC integriert.

B. Bausteine der Systeme und Hierarchie

Systeme setzen sich aus einer Reihe von SystemC-Modulen (SC_MODULE) zusammen. Diese Module sind die Grundbausteine der Systeme und beinhalten weitere Strukturen wie Ports, Prozesse, Kanäle (siehe Unterabschnitt IV-C), Events, Daten und Instanzen anderer Module. Die Module sowie die darin liegenden Strukturen werden als C++-Klassen implementiert. Die Hierarchie der Module wird hierbei durch die zuletzt genannte Instanziierung von Modulen innerhalb anderer Module festgelegt [10].

C. Kommunikation

Die Kommunikation zwischen Modulen oder Prozessen innerhalb eines Moduls findet bei SystemC durch Kanäle statt. Der Zugriff auf diese Kanäle geschieht über Interfaces, welche von diesen Kanälen eingebunden werden. Dieser Aufbau orientiert sich am objektorientierten Typsystem von C++ [10]. Somit kann ein solcher Kanal auch im Nachhinein, unabhängig von den benutzenden Modulen, überarbeitet oder durch einen anderen Kanal mit dem selben Interface ausgetauscht werden.

SystemC bietet auch eine Auswahl vordefinierter Kanäle wie einen Mutex-Kanal, welcher nur von einem Prozess gleichzeitig ge- und entsperrt werden kann, einen Clock-Kanal, der ein Zeitsignal sendet, oder einen FIFO-Kanal, welcher als ein First-In-First-Out-Buffer dient [10].

Auf diese Weise kann ein SystemC-Kanal in einem finalen System für ein einfaches Kabel, aber auch für ein komplexes Bussystem zur Datenübertragung stehen. Dies ermöglicht zum Beispiel, einen einfachen Bus im Lauf der Entwicklung durch eine detailliertere Hardware-Implementierung zu ersetzen und so auch rückwirkend die Performanz der Datenübertragung zu steigern [6].

D. Simulation mit SystemC

Für die Simulation von Echtzeitmodellen ist eine Möglichkeit der Zeitabschätzung notwendig. Der Kern von SystemC bietet eine solche Möglichkeit, indem er die Laufzeit misst, welche über die Klasse sc_time dargestellt werden kann. Der Kern kann hierbei die Zeit in Einheiten von Sekunden bis zu Femtosekunden (10^{-15} Sekunden) messen [10]. Auf diese Zeit des Kerns kann über den in Unterabschnitt IV-C genannten Clock-Kanal zugegriffen werden.

Da die Nebenläufigkeit im Hardwaresystem gegenüber der Simulation meist überwiegt, wird bei SystemC, wie auch bei Simulatoren anderer Sprachen, jede nebenläufige Einheit so lange simuliert, bis diese nicht mehr weiterarbeiten kann, da sie vom Ergebnis einer anderen Einheit abhängig wird [6]. Diese Abhängigkeiten werden bei SystemC durch Events

realisiert. Die Simulation wird in SystemC vom mitgelieferten Simulationskern übernommen [10].

Bevor die Simulation startet, werden in der Elaborationsphase angelegte Strukturen erzeugt. Anschließend wird der Simulationskern gestartet [10]. Er läuft nach dem in Abbildung 5 gezeigten Schema ab. Der Kern simuliert hierbei die Zeit, in welcher die einzelnen Komponenten laufen. Dieser Ablauf der Prozesse findet in der Evaluationsphase statt. Auf diese Weise werden alle Prozesse abgearbeitet und am Ende deren Ausgaben aktualisiert (Update). Entsprechend dieses Updates können nun eventuell weitere Prozesse voranschreiten. Währenddessen wird die simulierte Zeit nicht fortgesetzt, um die Nebenläufigkeit zu simulieren. Wird als Folge des Updates kein neuer Prozess ausgeführt, wird die Simulationszeit fortgesetzt und erneut evaluiert. Sollten keine Prozesse mehr laufen müssen, wird die Simulation beendet [6].

E. Synthese von C++ und SystemC-Code

Für die Synthese von C++ Code mit SystemC wird ein externer SystemC-Synthetisierer benötigt. Hierbei gilt es allerdings, einiges im Bezug auf die zugrunde liegende Sprache zu beachten.

SystemC als Klassenbibliothek baut auf vielen Konstrukten der C++ Programmiersprache auf. Auch wenn viele dieser Konstrukte bei der Hardwarebeschreibung durch eine hohe Abstraktion hilfreich sind, können nicht alle auch für die Synthese in einem Hardwaresystem eingesetzt werden. In einem üblichen C++ Programm liegen alle Daten auf einem gemeinsamen Speicher mit gemeinsamem Adressraum. Mit diesem Prinzip funktioniert die Zeigerstruktur der Sprache. In einem Hardwaresystem kann eine solche Speicherung nicht garantiert werden, weshalb sich diese Struktur nicht für die Synthese eignet [11]. Unter dem gleichen Aspekt der Speichernutzung können Methoden der dynamischen Speicherzuordnung (engl. dynamic memory allocation) nicht synthetisiert werden, da die Garantie, dass dies im Zielsystem funktioniert, nicht gegeben werden kann [11]. Die Speicherverwaltung muss also in einem synthetisierten SystemC-Projekt statisch sein. Zeigerzugriffe auf Speicheradressen innerhalb des Codes können nicht verwendet werden. Eine Ausnahme hierzu besteht, wenn eine Zeigeradresse bei der statischen Quellcodeanalyse eindeutig einem Ziel zugewiesen werden kann, in welchem Fall die Synthese funktioniert [11]. Es zeigt sich ein Problem der Synthese bei laufzeitabhängigen Daten. Dementsprechend ist auch eine Verwendung virtueller Funktionen in der Synthese nicht möglich, da sich das Verhalten hier, abhängig von Objekten zur Laufzeit, verändern kann [11].

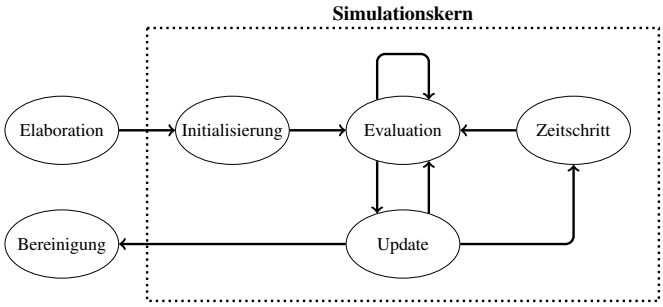

Abbildung 5. Schematischer Ablauf des Simulationskerns (nach [6])

F. SystemC für Transaction-Level-Modellierung

Das Transaction-Level ist keine direkte Abstraktionsebene, sondern die Beschreibung von Transaktionen von Prozessen. Mit Transaktionen wird die Kommunikation der Prozesse beschrieben. Es dient hierbei der Strukturbeschreibung über drei Abstraktionsebenen hinweg, die System-, die algorithmische- und die RT-Ebene [1]. SystemC bietet mit diversen Kommunikationsstrukturen wie Kanälen, Ports und Interfaces eine breite Palette an Möglichkeiten für die Modellierung auf dieser Ebene. Der Transaction-Level-Modeling-Standard (TLM-Standard) wird ebenso von der OSCI definiert. Diese Standardisierung wird ebenfalls im selben IEEE Standard [10] wie die Standardisierung von SystemC vorgenommen.

G. SystemC für RT-Ebenen-Modellierung

Abseits der Transaktions-Ebene kann auch ein Modell auf Register-Transfer-Ebene mit SystemC erstellt werden. Zur Erstellung eines RT-Modells wird eine genaue Angabe der Pins und ein Taktsignal benötigt [1]. RTL-Prozesse werden als Prozesse in den SystemC-Modulen angelegt. Die Pingenauigkeit wird durch die von SystemC gelieferten, definierbaren Bitvektoren realisiert [1]. In diese Module kann ein Taktsignal eingeführt werden, womit folglich auch die Zyklusgenauigkeit geboten wird, an welcher sich die Ausführung der Prozesse orientiert [1].

V. VERILOG

Verilog ist eine Hardwarebeschreibungssprache zur Beschreibung von elektronischen Schaltungen. 1983 wurde die Sprache vom Unternehmen Automated Integrated Design Systems unter dem Namen Verilog HDL entwickelt [12]. Durch die steigende Popularität entwickelte sich 1995 daraus der Verilog-Standard IEEE 1364-1995, welcher 2001 und 2005 jeweils aktualisiert wurde [13]. Der aktuelle Verilog-Standard ist in IEEE 1364-2005 definiert. Verilog HDL wurde anfänglich als logische Simulationssprache entwickelt [12]. In Verilog können Beschreibungen von der Transistor-Ebene bis hin zur algorithmischen Ebene entwickelt werden.

A. Datentypen

Verilog bietet eine Vielzahl an vordefinierten Datentypen zur Modellierung von Hardwaresystemen. Während einige dieser Datentypen synthetisierbar sind, wurden auch weitere Typen definiert, die nur der abstrakteren Modellierung und Simulation dienen [12]. Verilog-Datentypen sind in verschiedene Kategorien unterteilbar. Als Ausgangspunkt für einen Großteil dieser Typen dienen einfache Werte, welche diese Datentypen speichern [12]. Diese Werte umfassen die logischen Zustände 0 und 1, Unbekannt und Hochohmig, welche zudem mit einer Stärkestufe genauer spezifiziert werden können. Bei den Datentypen wird unterschieden zwischen Verbindungstypen wie

Kabeln (wire), Variablen zur Zwischenspeicherung, Vektoren und Arrays.

B. Bausteine der Systeme und Hierarchie

In Verilog besteht jedes System aus mindestens einem Modul, auch Top-Level-Modul genannt [14]. Dieses Modul kann dann hierarchisch Instanzen von Untermodulen beinhalten, sodass auch mehrere Hierarchie-Ebenen entstehen können. Da ein Untermodul unabhängig vom Gesamtsystem entwickelt werden kann, muss gewährleistet werden, dass Anforderungen wie benötigte Ausführzeiten auch innerhalb des übergeordneten Moduls erfüllt sind [13].

Ein Modul beginnt bei der Beschreibung mit dem Schlüsselwort module und endet mit endmodule. Dabei enthält es die eingehenden (input), ausgehenden (output) und bidirektionalen (inout) Anschlüsse. Diese Anschlüsse können von jedem Datentyp sein, jedoch sind nur spezielle Typen synthetisierbar (Kabel, Register, Integer) und können als Schnittstelle vom Modul gesehen werden [12], [14].

C. Kommunikation

Module bieten eine Schnittstelle mit Anschlüssen für die Kommunikation nach außen. Auf diese Schnittstelle greift ein übergeordnetes Modul zu, welches das Modul instanziiert. Im Untermodul werden die Ein- und Ausgänge verwendet und verarbeitet, sodass nach außen nur die Schnittstelle sichtbar ist. Um mehrere Untermodule zu verknüpfen, wird ein übergeordnetes Modul zur Kommunikation verwendet [12]. Dieses übergeordnete Modul kann die Ein- und Ausgänge der Untermodule miteinander verbinden, sodass eine übersichtlichere und abstraktere Modellierung möglich ist.

D. Verilog für Gatter- und Transistor-Ebenen-Modellierung

Verilog ermöglicht die Beschreibung von Hardware auf verschiedenen Abstraktionsebenen [14]. So kann Hardware mit Transistoren beschrieben werden oder abstrakter bis zur algorithmischen Ebene. Diese Ebenen sind in Verilog über drei Stufen realisierbar.

Im Verilog-Standard von IEEE ist die hardwarenahe Beschreibung auf Gatter- und Transistor-Ebene festgelegt. Hardware wird auf dieser Ebene unter anderem mit Transistoren und Logikbausteinen beschrieben [14]. Dabei sind auch physikalische Signalstärken beschreibbar.

E. Verilog für Strukturberschreibungen

Neben der hardwarenahen Abstraktionsebene ist in Verilog die Beschreibung auf RT-Ebene möglich. Ziel der RT-Ebene ist es, eine optimale Beschreibung mit möglichst wenigen Logikgattern zu erzeugen [13]. Dafür wird in dieser Ebene kombinatorische Logik eingesetzt.

Bei der abstrakteren Modellierung von Hardware wird zwischen der Struktur- und Verhaltensbeschreibung unterschieden [13]. Die reine Strukturbeschreibung von Hardware umfasst keine Verhaltensbeschreibungen [12]. Bei der Modellierung von Hardware über die Struktur werden allein Untermodule und deren Verbindungen beschrieben. Die Strukturbeschreibung wird eher für Logikblöcke einfacher Komplexität eingesetzt [13].

F. Verilog für Verhaltensbeschreibungen

Statt ein Hardwaresystem über Logikblöcke und Transistoren zu beschreiben, dient die Verhaltensbeschreibung von Verilog zur Modellierung von komplexeren Systemen [14]. Dabei wird unter anderem sequenzielle Logik eingesetzt. Durch die Verwendung von sequenzieller Logik können beispielsweise Flip-Flops und Schieberegister beschrieben werden [13]. Zu beachten ist, dass nicht jedes über die Verhaltensbeschreibung modellierte System synthetisierbar ist. Die Verhaltensbeschreibung erstreckt sich von der RT-Ebene bis hin zu algorithmischen Operationen. Auf dieser algorithmischen Ebene stehen Funktionen wie Schleifen zur Verfügung, welche nicht synthetisierbar sind [12]. Die algorithmische Ebene dient demnach der Simulation von Hardwarebeschreibungen.

G. Zeit und Nebenläufigkeit

Zeit spielt bei der Simulation von eingebetteten Systemen eine wichtige Rolle. Verilog bietet daher für die Simulation eine zeitliche Präzision von Sekunden bis Femtosekunden [12], [14]. Über die Direktive timescale kann eine Zeit definiert werden, um Simulationszeiten und Verzögerungen zu beschreiben.

Für Nebenläufigkeit stehen in Verilog diverse Sprachkonstrukte zur Verfügung. Dabei wird zwischen synthetisierbaren und nicht synthetisierbaren Sprachkonstrukten unterschieden. Neben kontinuierlichen Zuodnungen von Verbindungen dienen unter anderem prozedurale Zuweisungen zur nebenläufigen Beschreibung von Hardware [12]. Bei prozeduralen Zuweisungen können verschiedene Sprachkonstrukte verwendet werden, welche zum Teil synthetisierbar sind oder nur der Simulation dienen [12].

H. Simulation und Synthese

Die funktionale Verifikation von Hardwarebeschreibungen geschieht in Verilog über die Simulation in Prüfständen. Das zu testende System wird in einer Umgebung als Subsystem instanziiert und überprüft [12]. Dabei werden die Ein- und Ausgabewerte vom Prüfstand kontrolliert. In der Verhaltensbeschreibung stehen diverse Sprachkonstrukte für die Simulation mit Verilog bereit [14].

Die Synthese von Verilog-Code ist ein iterativer Prozess [13]. Dabei wird die RT-Ebenen-Beschreibung so lange optimiert, bis die Anforderungen Flächenbedarf, Geschwindigkeit und Energieverbrauch erfüllt sind. Um eine synthesefähige Beschreibung zu erstellen, dürfen nur synthesefähige Sprachkonstrukte verwendet werden [13]. Die Synthese wird hierbei, wie auch bei VHDL und SystemC, durch ein externes Werkzeug durchgeführt.

I. SystemVerilog

SystemVerilog ist eine Beschreibungs-, Spezifikations- und Verifikationssprache für Hardware [15]. SystemVerilog wurde

anfangs als Erweiterung des Verilog-Standards IEEE 1364-2005 entwickelt [16]. Seit 2009 ist Verilog im SystemVerilog-Standard IEEE 1800-2009 mit inbegriffen [17].

SystemVerilog erweitert Verilog hinsichtlich der Verifikation von Hardwarebeschreibungen. Zu den Erweiterungen zählen unter anderem assertorische Sprachkonstrukte, um Simulationen zu erweitern [17]. Des Weiteren unterstützt SystemVerilog objektorientierte Programmierung, um eine höhere Abstraktion zu erreichen [17].

VI. Anforderungen an Beschreibungssprachen für eingebettete Systeme

Durch die Analyse der Beschreibungssprachen VHDL, Verilog, SystemVerilog und SystemC können nun notwendige Funktionen für die Beschreibung und Modellierung eingebetteter Systeme definiert werden. Im Gegensatz zu herkömmlichen Programmiersprachen weichen die Funktionen von Beschreibungssprachen weitgehend ab. Die folgenden Anforderungen eines eingebetteten Systems sollten von Beschreibungssprachen unterstützt werden.

A. Hierarchie und Abstraktion

Abbildung 3 beschreibt den wachsenden Umfang von Beschreibungen eingebetteter Systeme. Hieraus zeigt sich, dass eine Abstraktion von Systemen fast unvermeidbar ist, da sonst die Komplexität von Beschreibungen zu groß werden kann. Wie die vorgestellten Beschreibungssprachen sollte ein System auf verschiedenen Abstraktionsebenen beschrieben werden können. In Kombination hierzu ist ein hierarchischer Entwurf von Komponenten hilfreich, um die Anzahl an Komponenten in einer Beschreibung gering zu halten, ebenso wie die Komplexität [2].

B. Beschreibung von nebenläufigen Systemen

Eingebettete Systeme sind verteilte, nebenläufige Systeme, bestehend aus mehreren Komponenten [2]. Die nebenläufige Beschreibbarkeit von Komponenten eines Systems ist damit essenziell. Darüber hinaus sollte eine Beschreibungssprache die Beschreibbarkeit der Nebenläufigkeit einfach und verständlich halten, um menschliches Fehlverhalten bei fehlendem Verständnis über die Nebenläufigkeit zu reduzieren [2]. Die vorgestellten Beschreibungssprachen unterstützen die Beschreibung von Nebenläufigkeit.

C. Zeitliches Verhalten von eingebetteten Systemen

Viele eingebettete Systeme sind zugleich Echtzeitsysteme. Somit müssen zeitliche Anforderungen von eingebetteten Systemen abbildbar sein [2]. Eine Beschreibungssprache für eingebettete Systeme unterscheidet sich in diesem Punkt von Standardtheorien in der Informatik zur zeitlichen Modellierung wie der Landau-Notation [2]. Diese Theorien stellen die Zeit in Systemen abstrakter dar, sodass zwar Laufzeiten beschrieben werden können, aber keine realen Ausführungszeiten. Bei eingebetteten Systemen als Echtzeitsysteme sind jedoch genauere Ausführungszeiten wichtig [2]. Die vorgestellten Sprachen unterstützen deswegen Zeiteinheiten von Sekunden bis Femtosekunden für die Simulation.

D. Synthese

Die Synthese beschreibt den Prozess der Transformation einer Verhaltensbeschreibung in eine Strukturbeschreibung [1]. Durch die Automation stellt die Synthese somit einen wichtigen Schritt bei der Beschreibung eingebetteter Systeme dar, um die Produktivität beim Entwurf eingebetteter Systeme zu erhöhen [1], [5].

Die Synthese kann dabei auf verschiedenen Abstraktionsebenen stattfinden. Von der Transistor-Ebene bis zur System-Ebene können Verhaltensbeschreibungen in Strukturbeschreibungen transformiert werden [1]. Dabei kann die Synthese auf der gleichen Abstraktionsebene bleiben oder die tiefer liegenden Ebenen anschneiden [1].

Die Synthese in den vorgestellten Beschreibungssprachen erfolgt dabei über externe Synthetisierer.

E. Simulation und Verifikation

Beim Entwurf eines eingebetteten Systems wird zuvor eine Spezifikation mitgegeben. Um sicherzustellen, dass ein fertiges Produkt den Anforderungen entspricht (Verifikation), sind Prüfmethoden notwendig. Dabei ist auch zu beachten, dass die Spezifikation vollständig und korrekt ist, da Spezifikationsfehler zu Produktfehlern führen können [1], [18]. Bei eingebetteten Systemen kann zwischen drei Verifikationsmethoden unterschieden werden: formale Verifikationen, Simulationen und prototypische Implementierungen [18].

Die formale Verifikationsmethode wird als vollständig bezeichnet, da durch mathematische Beweise die Abwesenheit von Fehlern gezeigt werden kann [18]. Da die formale Verifikation für komplette Systeme sehr aufwendig ist, werden eher nur Teilsysteme beschrieben [18].

Bei der Simulation von eingebetteten Systemen werden Ausgabedaten anhand von sinnvollen Eingabedaten getestet [1]. Dabei sind ausführbare Verhaltens- und Strukturmodelle simulierbar. Durch die steigende Komplexität ist es dabei nur schwer möglich, alle Szenarien eines Systems zu simulieren, weshalb die Simulation als unvollständige Verifikationsmethode bezeichnet wird [18]. Durch den Verzicht auf Vollständigkeit ist jedoch im Gegensatz zur formalen Verifikationsmethode die Simulation von ganzen Systemen möglich [18]. Dabei ist zu beachten, dass sich mit steigender Komplexität der Simulation die Simulationszeit erhöht.

Prototypische Implementierungen helfen dabei, exaktere Abschätzungen nichtfunktionaler Anforderungen gegenüber formalen und simulativen Verifikationsmethoden zu geben [18]. Dabei wird das System in einer realen Umgebung getestet. Somit ist diese Form der Verifikation erst sehr spät im Entwurfsfluss verfügbar [1], [18].

Die Systeme werden auch nach dem Entwurf, bei der Fertigung erster Hardware-Chips, durch Black-Box- und White-Box-Tests überprüft. Hierbei werden in der Spezifikation beschriebene Signale an die Eingänge des Chips gelegt und die Ausgänge des Chips gemessen. Die Messdaten werden anschließend mit der Spezifikation verglichen [1]. Während der Tester bei einem Black-Box-Test keine Kenntnis über den Inhalt des Chips besitzt, hat er bei einem White-Box-Test

Kenntnis über die Algorithmen und Struktur der Soft- und Hardware des Systems [1]. Demnach kann ein Test gezielter durchgeführt werden.

VII. Fazit

A. Spezielle Anforderungen eingebetteter Systeme

Die Anforderungen eingebetteter Systeme unterscheidet sich von denen herkömmlicher Softwaresysteme stark. Zwei wesentliche Faktoren beim Entwurf eingebetteter Systeme spielen das Zeitverhalten und der Platzverbrauch der entworfenen Schaltung, wobei auch diese beiden Anforderungen untereinander in Zusammenhang stehen.

Eingebettete Systeme müssen oft Echtzeitanforderungen erfüllen, das heißt sie müssen in einem festen Zeitintervall ihre Arbeit erledigt haben [1], wie in Unterabschnitt I-A am Beispiel des Airbagsystems aufgezeigt. Auch der verbrauchte Platz der finalen Schaltung spielt hierbei eine Rolle. Ist die Schaltung des Systems kleiner, wird nicht nur im Rahmen des umliegenden Systems Platz eingespart. Durch die Verkürzung der Datentransportwege zwischen Komponenten wird auch die Geschwindigkeit erhöht, mit der die transportierten Informationen ihr Ziel erreichen. Hinzu kommt bei diesem Entwurf der Kostenfaktor, da durch Realisierung mit weniger Material und Bauteilen die Kosten des eingebetteten Systems gesenkt werden können.

Hier liegt folglich ein besonderer Fokus von System- und Hardwarebeschreibungssprachen. Das Zeitverhalten kann so auch in sehr kleinen Intervallen bei der Synthese getestet werden. Eine strukturelle Beschreibung auf niedriger Abstraktionsebene ermöglicht zusätzlich eine gute Optimierung des Platzverbrauchs einer Schaltung.

B. Unterschied der System- und Hardwarebeschreibungssprachen

Eine Systembeschreibung wird für den Entwurf einer Architektur aus Software und Hardware genutzt. In ihr wird beschrieben, wie die Bausteine miteinander interagieren und Daten untereinander ausgetauscht werden. Demgegenüber bietet eine Hardwarebeschreibungssprache die Möglichkeit eines Entwurfs, welcher näher an der tatsächlichen Hardware liegt. So kann zum Beispiel ein konkretes Bauteil für ein übergeordnetes System entworfen werden. Beide Sprachen bieten die Möglichkeit, auf verschiedenen Ebenen entweder das Verhalten oder die Struktur zu beschreiben. Je niedriger dabei die Abstraktion ist, desto näher kommen sich die Verhaltens- und Strukturbeschreibung [1].

Systembeschreibungssprachen können dafür eingesetzt werden, eingebettete Systeme sowie deren umliegendes System und die Kommunikation zwischen diesen zu beschreiben. Demhingegen eignen sich Hardwarebeschreibungssprachen zur Beschreibung des eingebetteten Systems sowie dessen Bausteine.

Literatur

[1] O. Bringmann, W. Lange, und M. Bogdan, *Eingebettete Systeme*, 3. Aufl., Serie De Gruyter Studium. Berlin und Boston: De Gruyter Oldenbourg, 2018.

[2] P. Marwedel, *Embedded System Design*, 3. Aufl., Serie Embedded Systems. Cham: Springer International Publishing, 2018.

[3] B. Bouyssounouse und J. Sifakis, *Embedded Systems Design*. Berlin und Heidelberg: Springer Berlin Heidelberg, 2005.

[4] G. Lehmann, B. Wunder, und M. Selz, *Schaltungsdesign mit VHDL*. Poing: Franzis, 1994.

[5] D. D. Gajski, S. Abdi, A. Gerstlauer, und G. Schirner, *Embedded System Design*. Boston, MA: Springer US, 2009.

[6] D. C. Black, J. Donovan, B. Bunton, und A. Keist, *SystemC: From the Ground Up*, 2. Aufl. Boston, MA: Springer US, 2010.

[7] W. Lange und M. Bogdan, *Entwurf und Synthese von Eingebetteten Systhemen*. München: Oldenbourg Wissenschaftsverlag GmbH, 2013.

[8] VHSIC Program Office, „Very High Speed Integrated Circuits - VHSIC - Final Program Report 1980 – 1990," United States Department of Defense, Tech. Rep., 1990. [Online]. Abrufbar: https://apps.dtic.mil/dtic/tr/fulltext/u2/a230012.pdf

[9] *IEEE Standard for VHDL Language Reference Manual*, IEEE Std 1076-2019 (Revision of IEEE Std 1076-2008), 2019.

[10] *IEEE Standard for Standard SystemC Language Reference Manual*, IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005), 2012.

[11] M. Meredith, *High-Level Synthesis: From Algorithm to Digital Circuit*. Dordrecht: Springer Netherlands, 2008, Kap. High-Level SystemC Synthesis with Forte's Cynthesizer, S. 75–97.

[12] B. J. LaMeres, *Quick Start Guide to Verilog*. Cham: Springer International Publishing, 2019.

[13] V. Taraate, *Digital Logic Design Using Verilog: Coding and RTL Synthesis*. New Delhi: Springer India, 2016.

[14] *IEEE Standard for Verilog Hardware Description Language*, IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001), 2006.

[15] *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012), 2018.

[16] S. Sutherland, S. Davidmann, und P. Flake, *SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Boston, MA: Springer US, 2006.

[17] C. Spear und G. Tumbush, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Boston, MA: Springer US, 2012.

[18] C. Haubelt und J. Teich, *Digitale Hardware/Software-Systeme*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.