Hardware/Software Codesign

SystemC und Transaction-Level Modeling

Florian Eibensteiner

Embedded Systems Design
FH Hagenberg

2018

R 4309

Inhalt

Literatur

David Black and Jack Donovan.
SystemC: From the Ground Up.
Springer, 2004.

Thorsten Grötker.
System design with SystemC.
Kluwer, 2002.

Frank Ghenassia.
Transaction-Level Modeling with SystemC.
Springer, 2005.

Lukai Cai and Daniel Gajski.
Transaction level modeling: An overview.
In *Proceedings of CODES+ISSS '03*, pages 19–24, New York, NY, USA, 2003. ACM.

Sudeep Pasricha and Mohamed Ben-Romdhane.
Using tlm for exploring bus-based soc communication architectures.
In *Proceedings of ASAP '05*, pages 79–85, Washington, DC, USA, 2005. IEEE Computer Society.

Inhalt

RTL Design – What's beyond?

"We all know that RTL design is tedious, complicated, and inefficient. We've known it for twenty years, in fact. To paraphrase Winston Churchill: RTL is the worst possible way to design electronics - except for all of the other ways that have been tried. (OK, and we know - Churchill was actually paraphrasing someone else. See? IP re-use works, even in politics!)" [1]

[1] Source: HLS versus OpenCL (http://www.eejournal.com/archives/articles/20130312-highlevel/)

Inhalt

Entstehung

- Entwicklung ab 1999 durch „Open SystemC Initiative“ (OSCI)
- Non-Profit Organisation
- Ziel: eine Sprache für Systementwurf, System-to-Silicon, basierend auf C/C++
- Source Code und Dokumentation ist frei erhältlich
- Standardisiert: IEEE 1666 – 2005 LRM
- Aktuelle Version: SystemC 2.3.2 (including TLM)
- Erweiterung zu SystemC: SCV 2.1, OVL 2.8.1, SystemC-AMS 2.0
- `http://www.accellera.org`

Was ist SystemC

Eigenschaften

- Standardisierte Entwurfs- und Verifikationsbibliothek in C/C++ für System-Level Design (Hardware und Software)
- Ermöglicht zyklen- und bitgenaue Modellierung und Simulation in C++, Simulator ist bereits integriert
- Verwenden von C++ Klassenbibliotheken (sehr flexibel)
- Bindeglied zwischen Konzept- und Bitebene
- Verschiedene Abstraktionsebenen möglich
- Modularisierung und Partitionierung ermöglicht Hardware/Software Codesign

Was ist SystemC

Eigenschaften

- Architektur-Exploration
- Softwareteile sind auch später verwendbar (Firmware)
- Einsatz von IP Bibliotheken
- „*Mixed Language*“ Simulation möglich (VHDL, Verilog, usw.)
- Durchgängiger Design-Flow: *TLM* $\rightarrow$ *RTL* $\rightarrow$ *Gatter*
  z.B. C-To-Silicon Compiler von Synopsys, Vivado HLS von Xilinx, ...

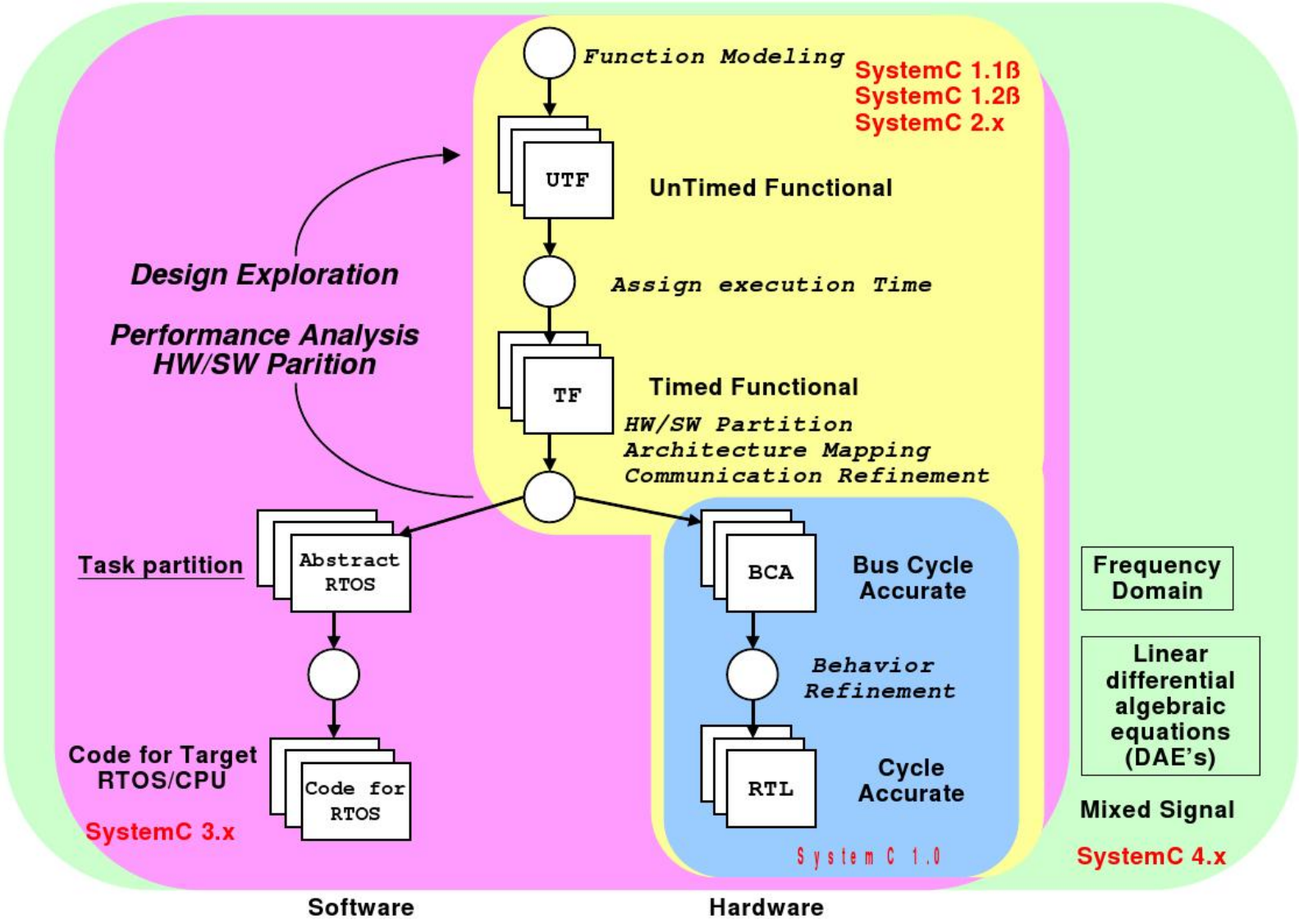
- Plattformunabhängig (Win32, Linux, ...)
- Source Code ist frei erhältlich
- ...

Was ist SystemC

Nachteile:

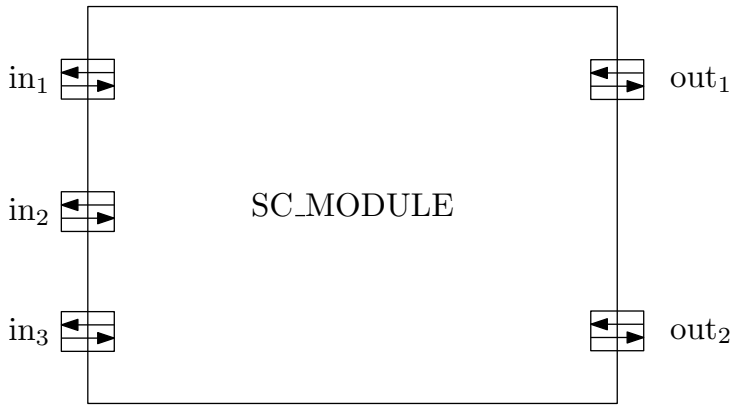
- Es gibt noch keinen einheitlichen Modellierungsansatz
- SystemC ist nur sehr eingeschränkt Synthetisierbar
- Kein durchgängiger Designflow (im Sinne von Beschreibungssprachen)
- Weniger Unterstützung von HDL Spezifika (z. B. Functional Coverage, Constrained Random, ...) $\rightarrow$ Abhilfe durch SCV bzw. OVL

SystemC Versionen



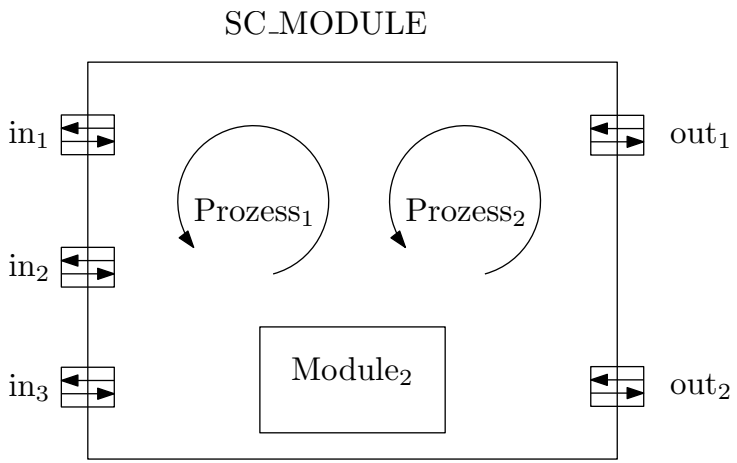
SystemC Module

- Module sind Grundblöcke (C++ Klasse)
- Module enthalten andere Module oder Prozesse
- Kommunikation mit anderen Modulen mittels Ports



SystemC Prozesse

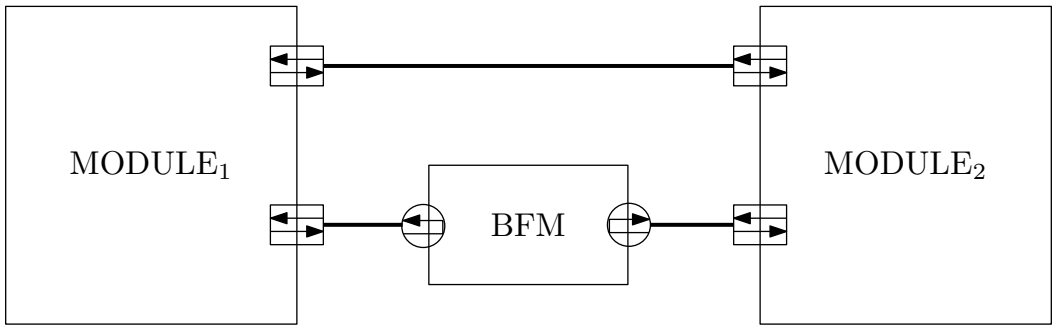
- Beschreibt die Funktionalität eines Moduls
- Funktion oder Methode welche vom Scheduler aufgerufen wird
- Prozesse laufen „parallel“
- Prozesse können sensitive auf Signals und Events sein



SystemC Ports und Kanäle

- Sind die Schnittstelle zwischen Modulen und dienen dem Austausch von Daten
- Ports haben eine Richtung (in, out, inout)
- Ports haben einen Typ: C/C++, SystemC, Benutzerspezifisch (Busse, ...)

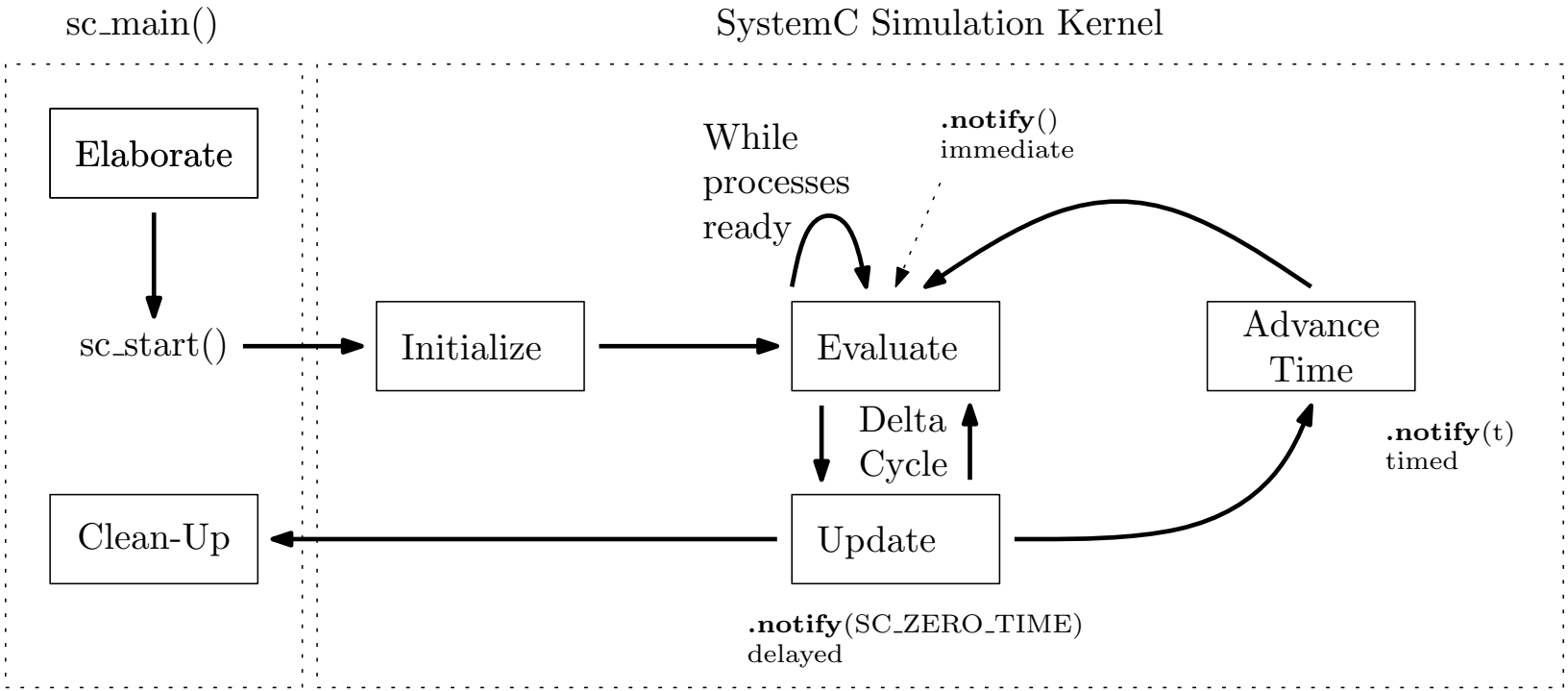
- Kanäle verbinden Ports von Modulen
- Kanäle haben ebenfalls einen Typ: C/C++, SystemC, Benutzerspezifisch (Busmodelle, Prozesse, ...)



Der Begriff „Zeit“

- Eigener Datentyp für die Darstellung der Zeit *sc_time*
  - quantitative Größe (intern 64-bit Integer)
  - Einheit (z.B. SC_SEC, SC_MS, . . .)
- Interne Auflösung der Zeit kann angepasst werden
- Dauer der Simulation festlegen.
  - *sc_start()* $\rightarrow$ Simulationszeit ∞
  - *sc_start(sc_time)* $\rightarrow$ Simulationszeit maximal *sc_time*
- Spezielles Objekt *sc_clock* für die Darstellung Taktsignalen
  - periodischer Pegelwechsel
  - Form des Signals einstellbar
  - Clock kann auch als *boolean* modelliert werden

SystemC Simulation Kernel



Inhalt

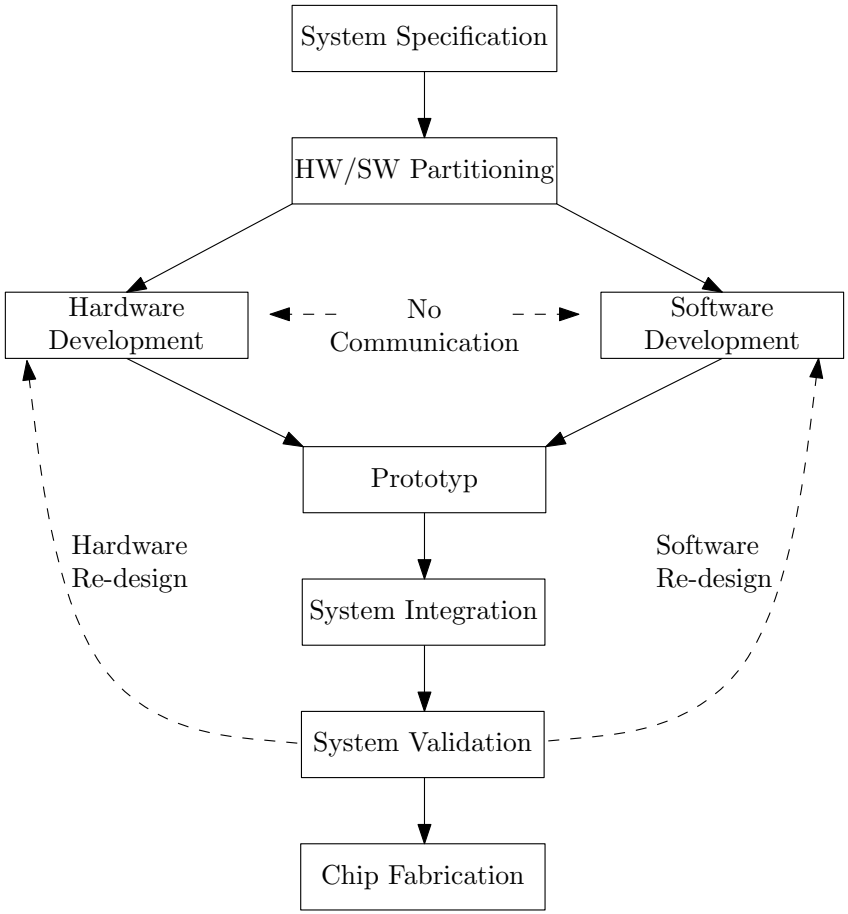
Probleme/Forderungen beim Entwurf komplexer Systeme

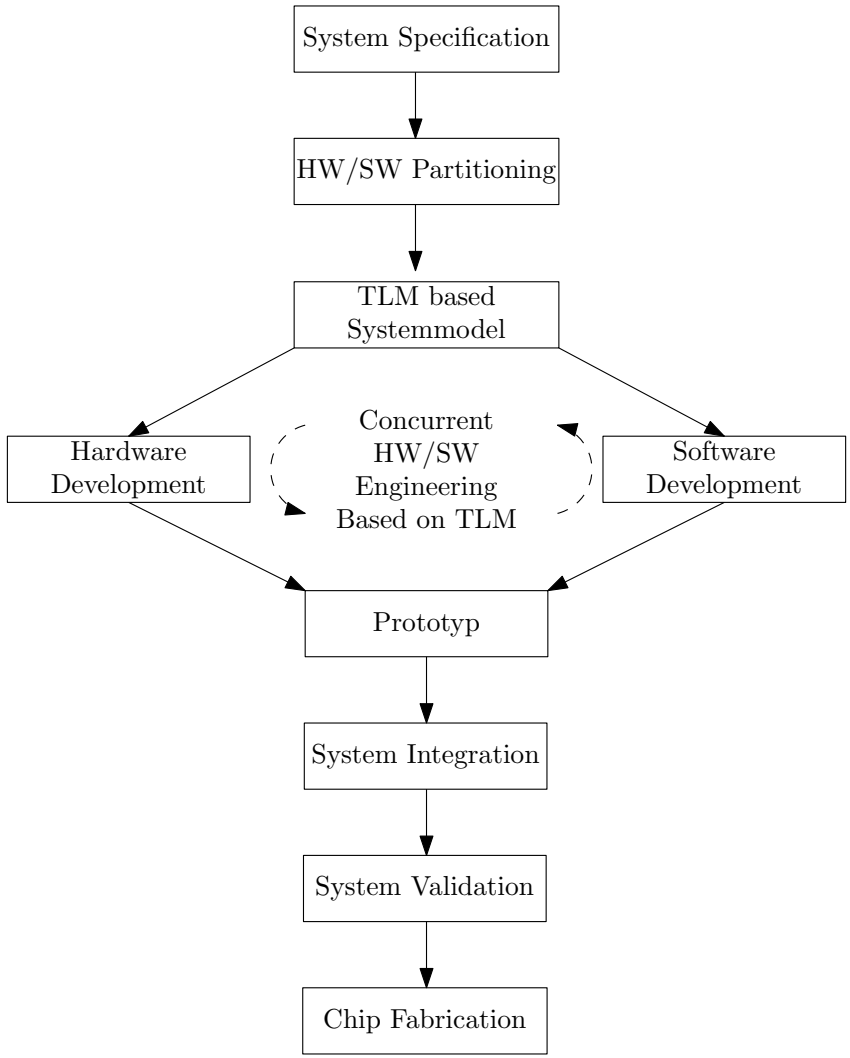
- Sind alle Anforderungen an das System bekannt?
- Spezifikation auf Systemebene - System-on-a-Chip
- Entwurfsraumexploration - Wo liegen die Designgrenzen?
- Verfeinerung von Algorithmen - Von Matlab nach RTL in einem Schritt?
- Paralleles entwickeln von Firmware, Hardwarekomponenten und Bussystemen - Time-to-Market!
- Systemsimulation
  - Simulationszeit
  - Verifikation von Hardware und Software früh im Designzyklus

TLM - eine Entwurfsmethodik

- Top-down Ansatz
  - Hardware- und Softwareentwicklung auf Basis des selben Modells
  - Modelliere nur jenen Detailgrad, der für Entwicklung einer Komponente erforderlich ist.
- Auftrennung in Kommunikation und Berechnung
- TLM ist Abstraktion von
  - Zeit
  - Daten
  - Funktion
- Vorteile:
  - Unabhängig von der Entwicklungssprache
  - Erhöhen der Simulationsgeschwindigkeit
  - Entwurfsraumexploration und Verifikation auf hoher Abstraktionsebene
  - Frühe Softwareentwicklung

Classic vs. TLM based Designflow





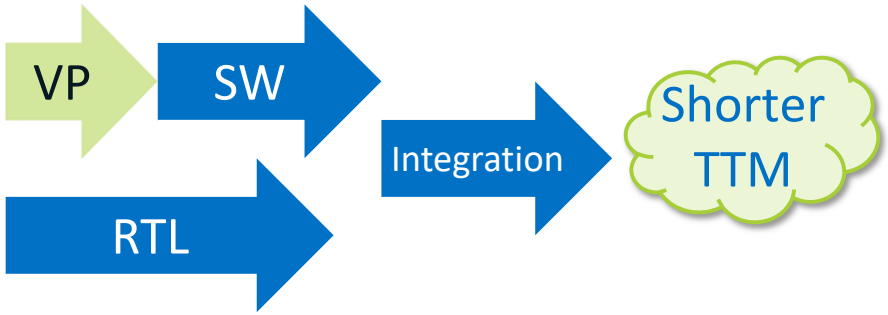
Classic vs. TLM based Designflow

- Klassisch, serieller Design Flow

Classic vs. TLM based Designflow

- Klassisch, serieller Design Flow

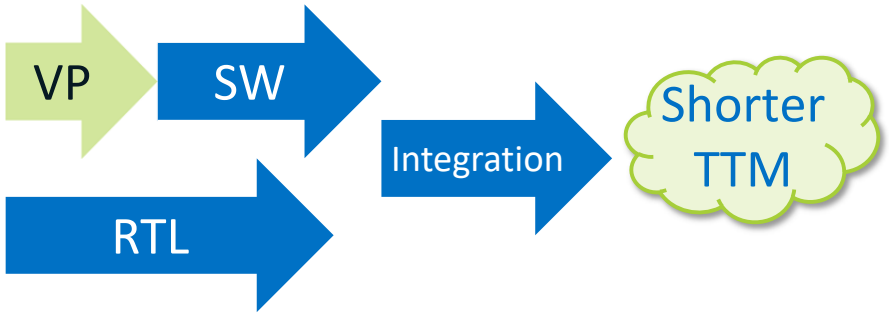
- VP: left-shift im Design Flow

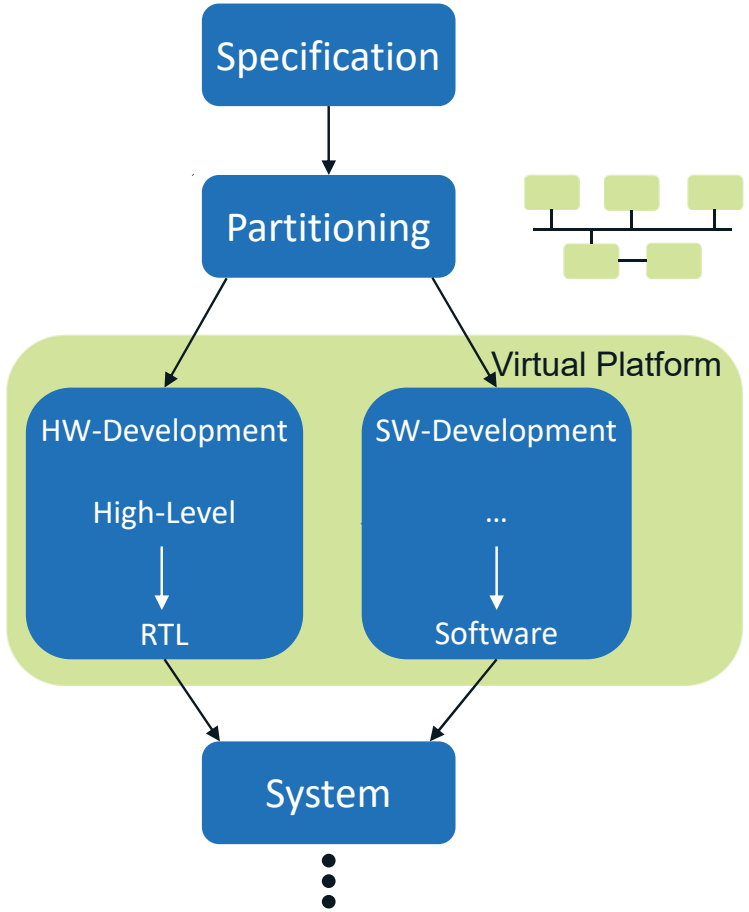


Classic vs. TLM based Designflow

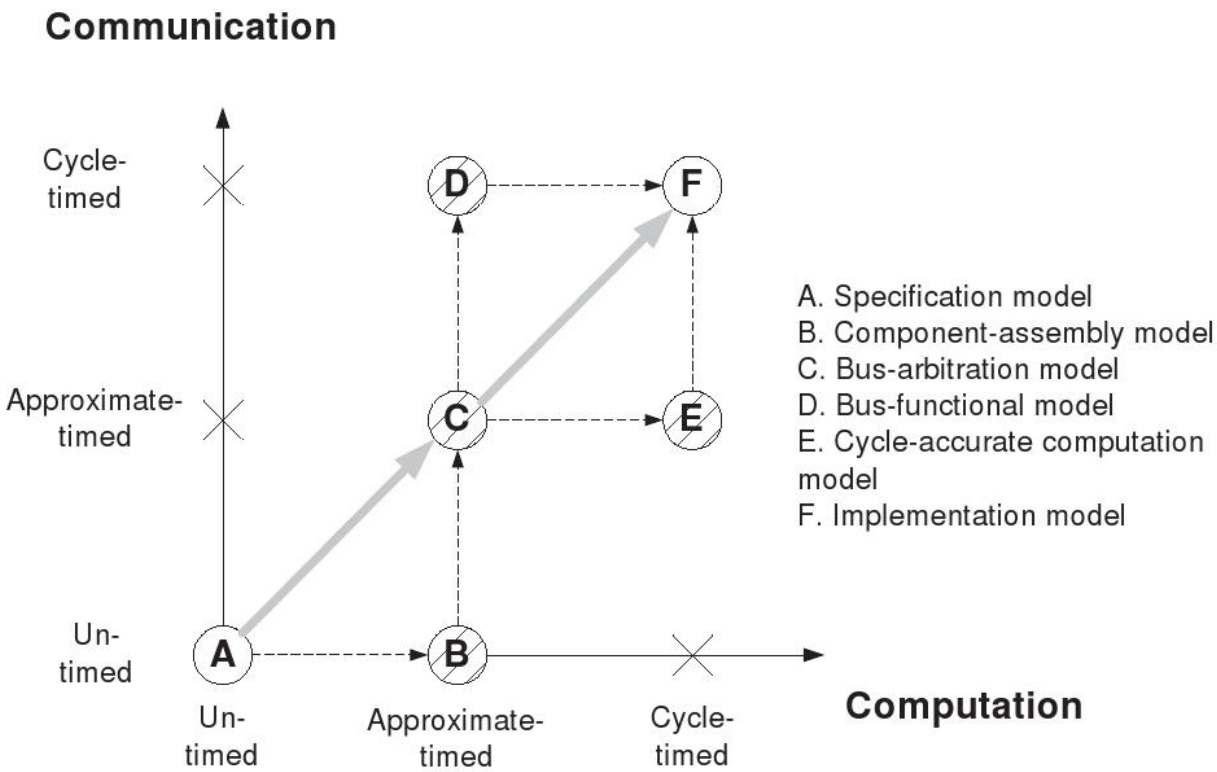
- Klassisch, serieller Design Flow

- VP: left-shift im Design Flow

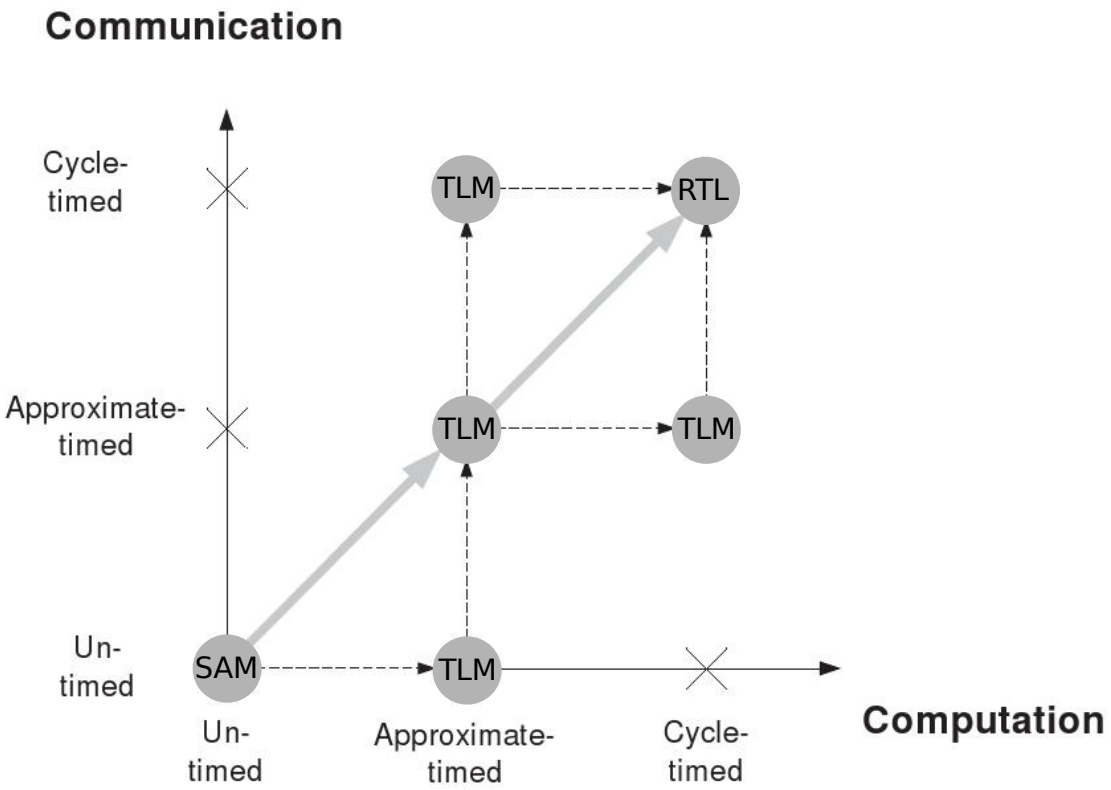




System Modeling Graph

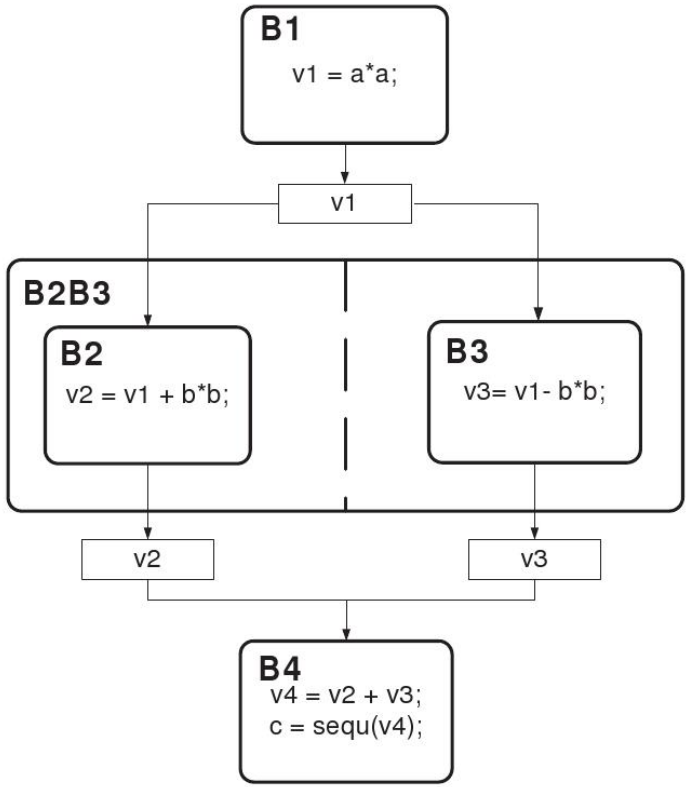


System Modeling Graph



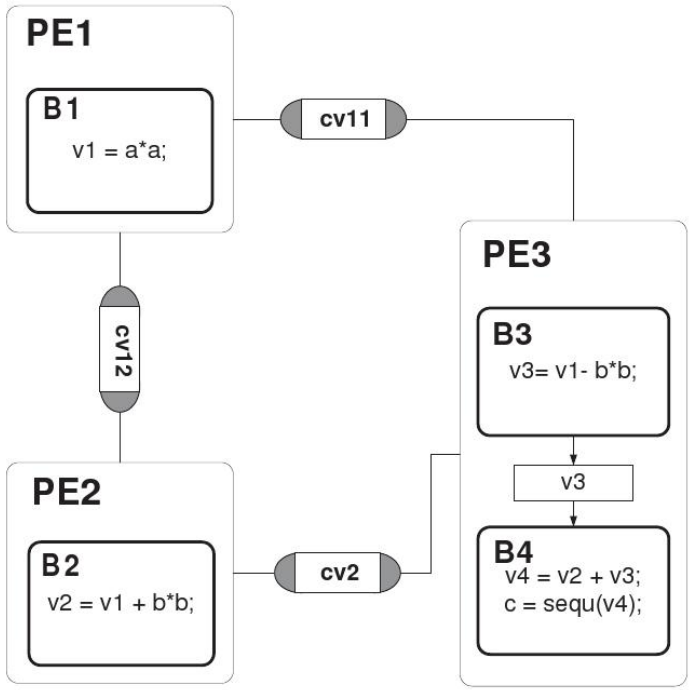
Specification Model

- Beschreibt die Funktionalität des Systems
- Modellierung der Kommunikation mittels Variablen (keine Kanäle)
- Keine Implementierungsdetails
- Übersetzung der Spezifikation von Matlab o.ä. nach C++/SystemC



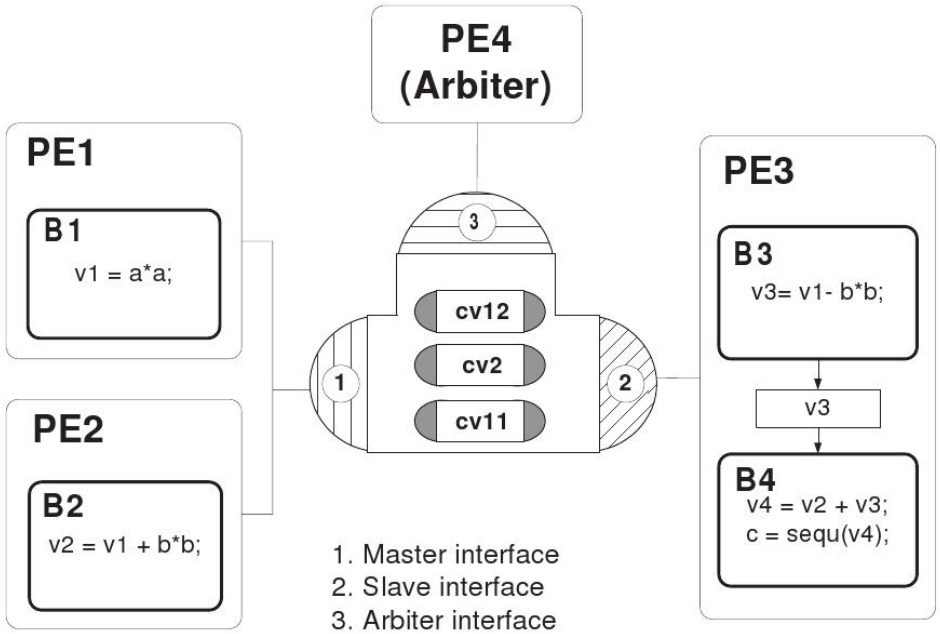
Component-Assembly

- Aufteilen in parallel arbeitende Prozesse
- Funktionale Blöcke werden auf PEs (Processing Elements) abgebildet
- Kommunikation mittels „Message Passing Channels“ (untimed)
- Berechnung wird mit Timing (geschätzt) modelliert
  Wait-Statement pro PE oder Code-Block



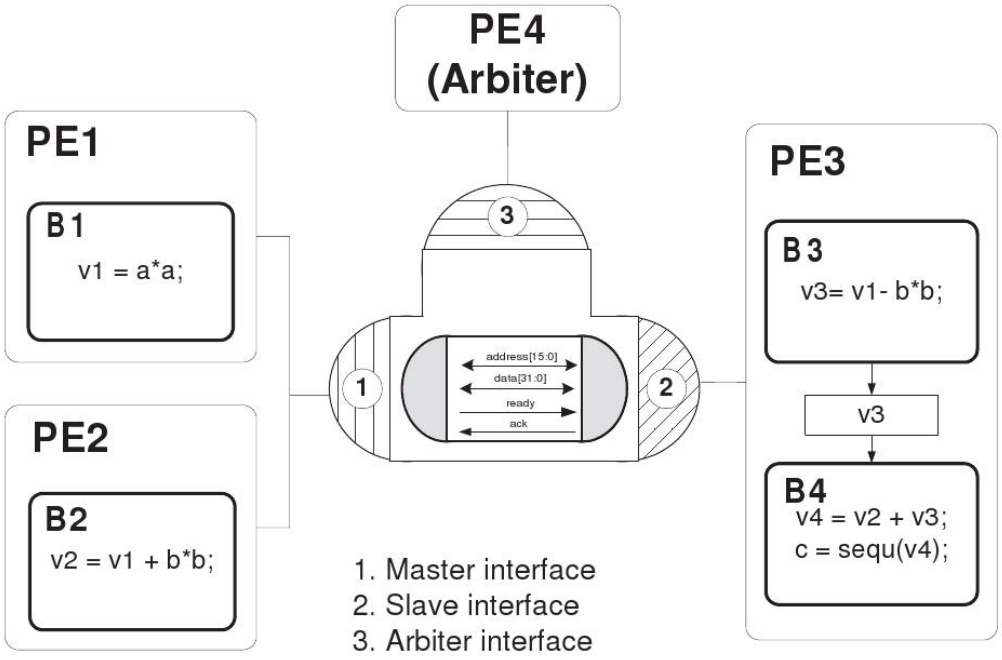
Bus-Abitration Model

- Betrachten der Kanäle als abstrakte Busse
- Protokoll → blocking und non-blocking Zugriffe
- Schnittstelle um Adresse und Priorität erweitert
- Pro Transaction wird ein Wait-Statement ausgeführt
- Ermöglicht Exploration der Busarchitektur



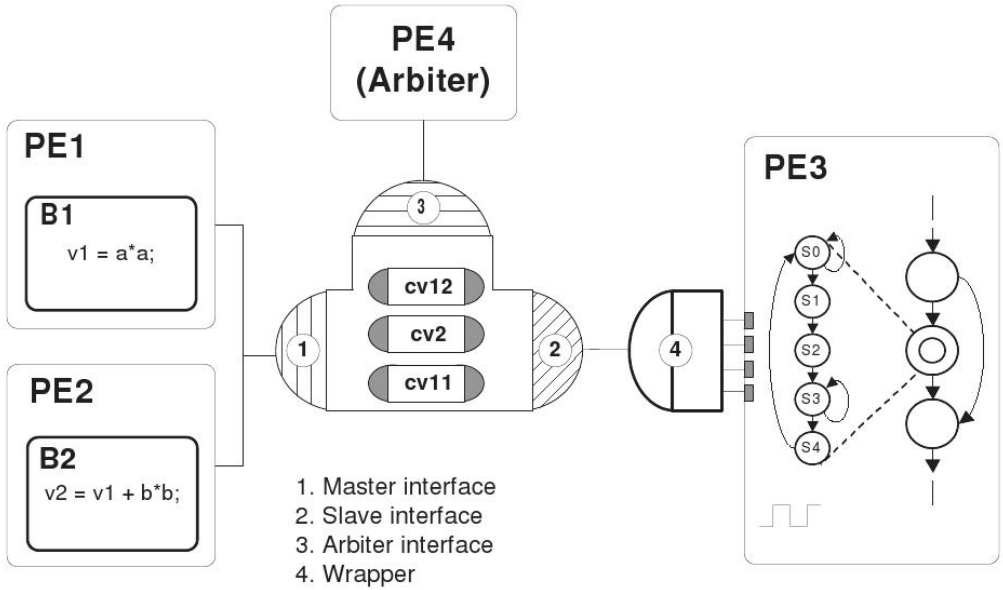
Bus-Functional Model

- „Protocol Channels“ → Zeit/Takt und Pin genau
  - Time accurate
  - Cycle accurate
- Schnittstelle kann weiterhin Abstrakt sein (Bus bildet intern BFM ab)
- PEs arbeiten „approximate-timed“



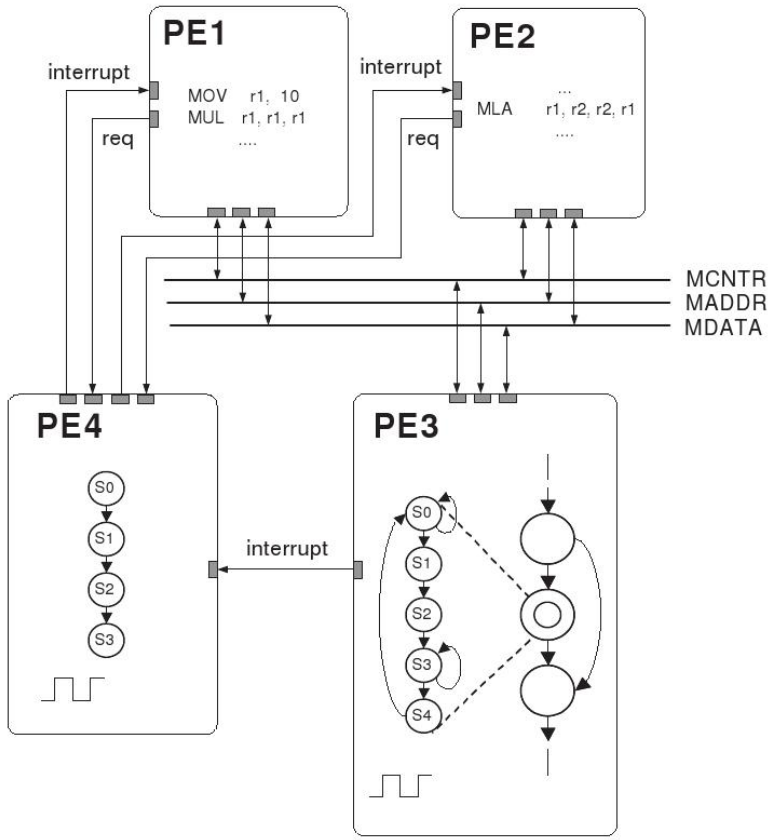
Cycle-Accurate Computation Model

- Teile der Berechnung erfolgt zyklengenau
  - Spezifische Hardware: RTL
  - CPU, DSP: Instruction Set Architecture (ISA)
- Schnittstelle ist pingenau
- Kommunikation „approximate-timed“

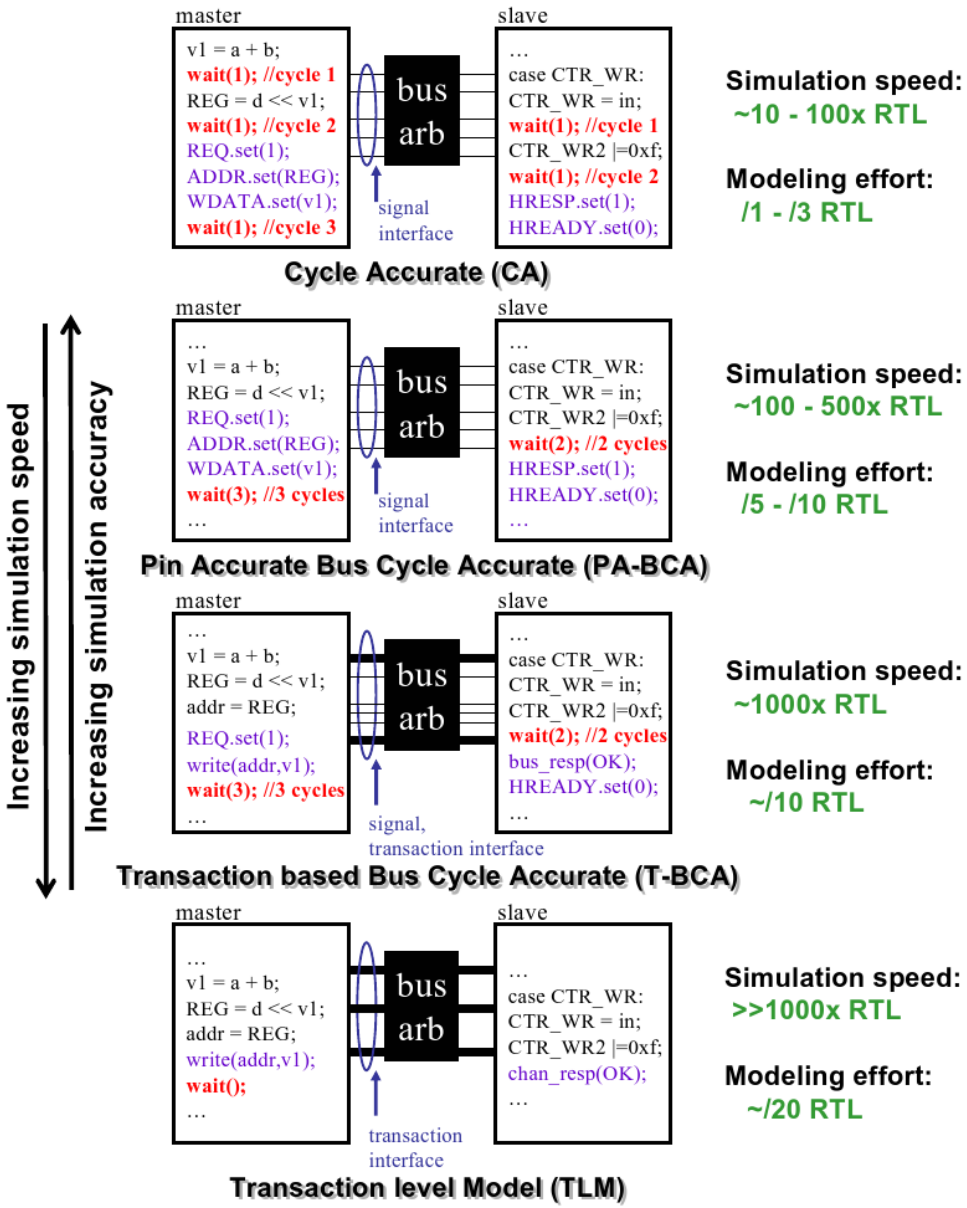


Implementation Model

- Alles zyklengenau, RTL und ISA

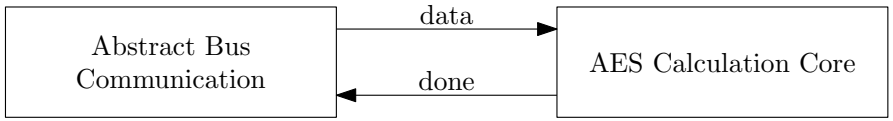


TLM - Aufwand und Speed-Up



Case Study: AES Core

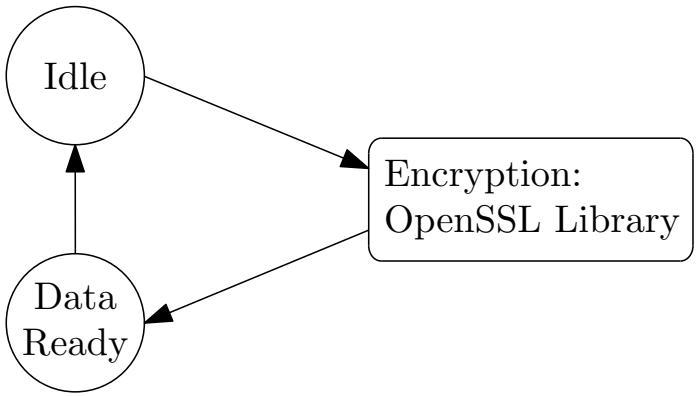
- Implementierung eines einfachen AES Cores auf verschiedenen Abstraktionsstufen
  - UML Modell als Grundlage für Codegenerator
  - Modelle A1 - A5 generierte SystemC Modelle
  - Abstraktionsstufen von untimed bis zyklengenau



- Vergleich Implementierungsaufwand, Komplexität und Simulationsgeschwindigkeit

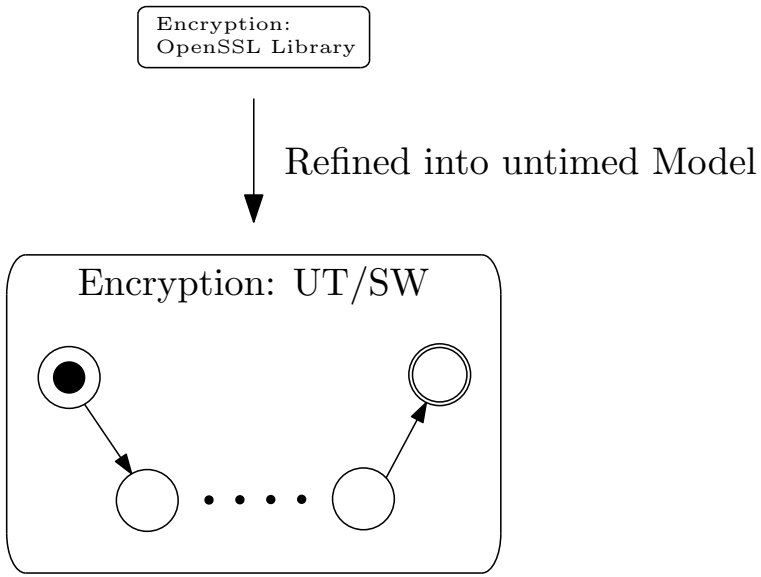
Case Study: AES Core – Levels of Abstraction

- A1 → Verschlüsselung durch Verwendung der OpenSSL Bibliothek



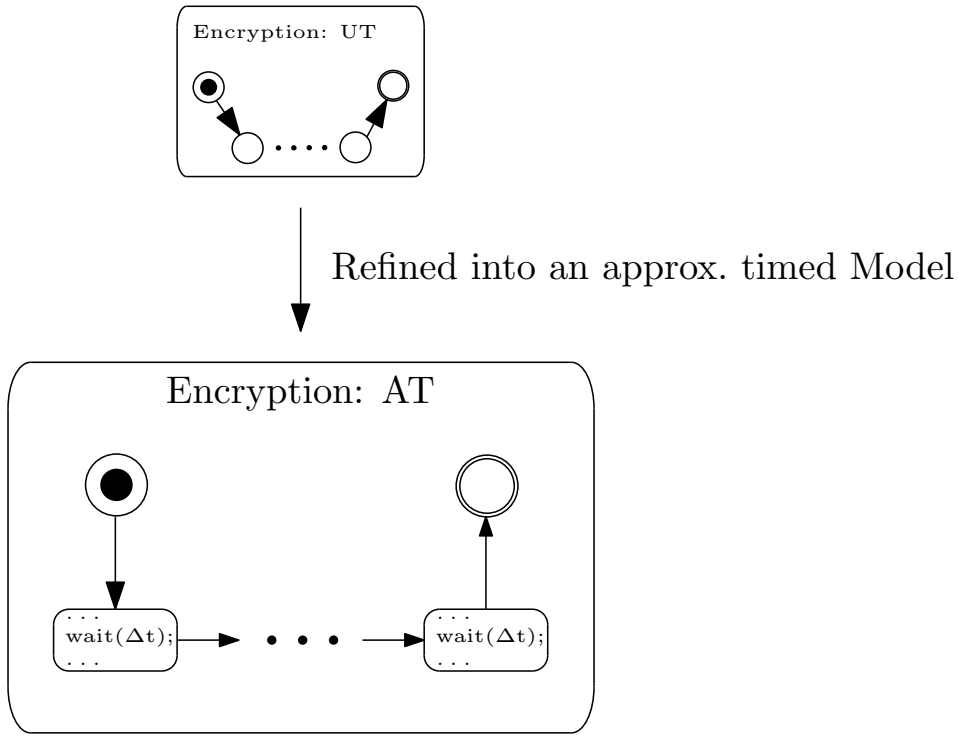
Case Study: AES Core – Levels of Abstraction

- A2 $\rightarrow$ Ersetzen der Bibliothek durch eigene Implementierung der Verschlüsselung
- Modell arbeitet weiterhin ohne Timing



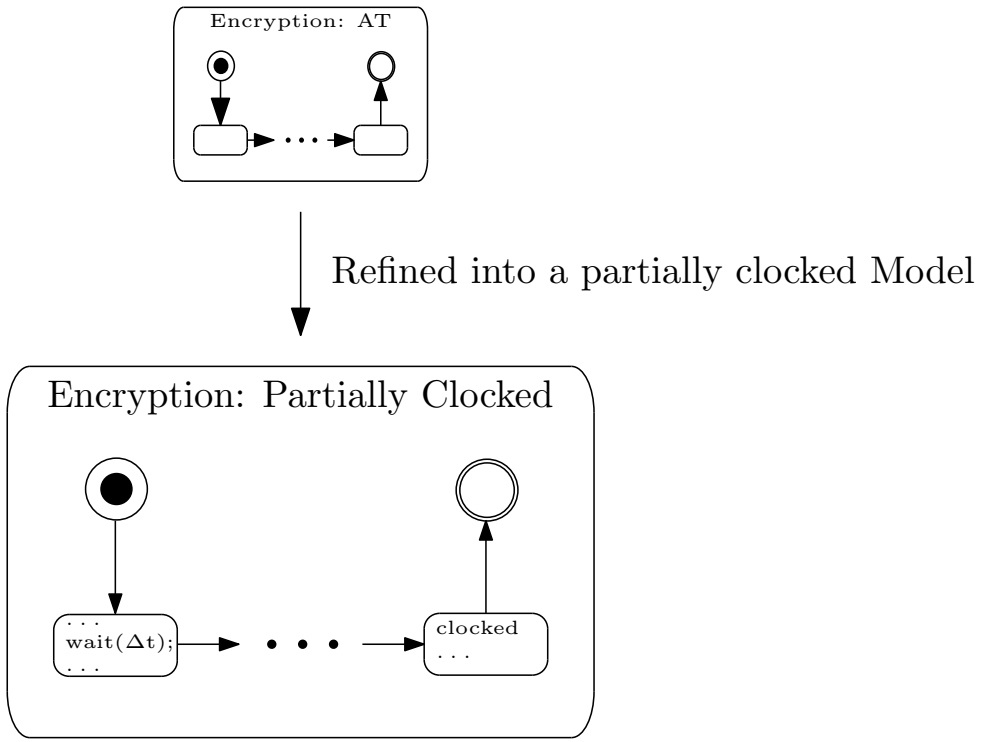
Case Study: AES Core – Levels of Abstraction

- A3 $\rightarrow$ Ersetzen der Berechnung aus *A2* durch strukturierte Implementierung mit groben Zeitverhalten
- Zeitverhalten einzelner Operationen bzw. Funktionsblöcke geschätzt



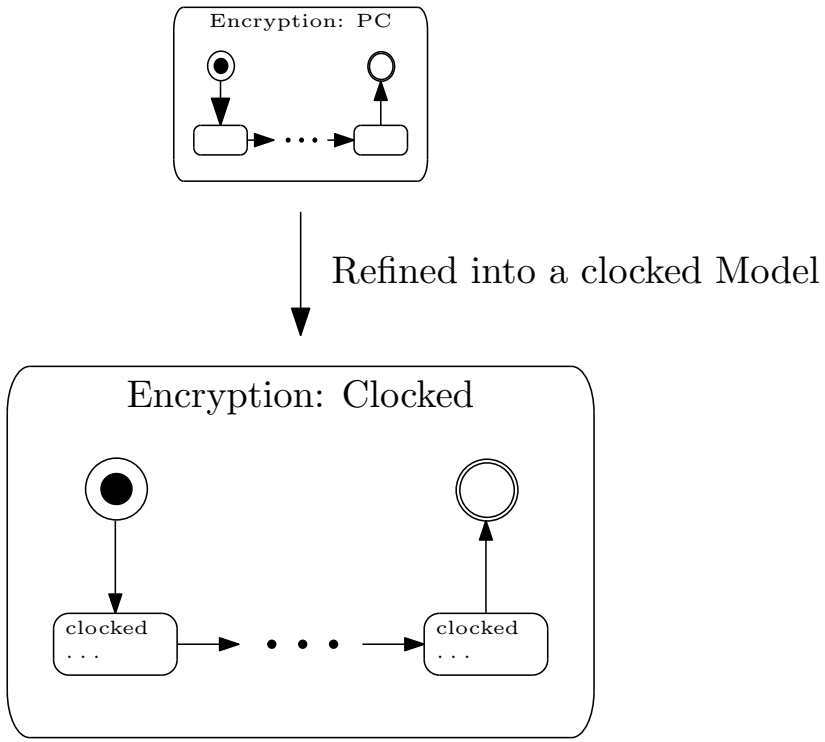
Case Study: AES Core – Levels of Abstraction

- A4 → Weitere Verfeinerung der Funktionsblöcke und Operationen aus *A3*
- Gemischtes Zeitverhalten → getaktete Funktionsblöcke und Funktionsblöcke mit geschätztem Timing



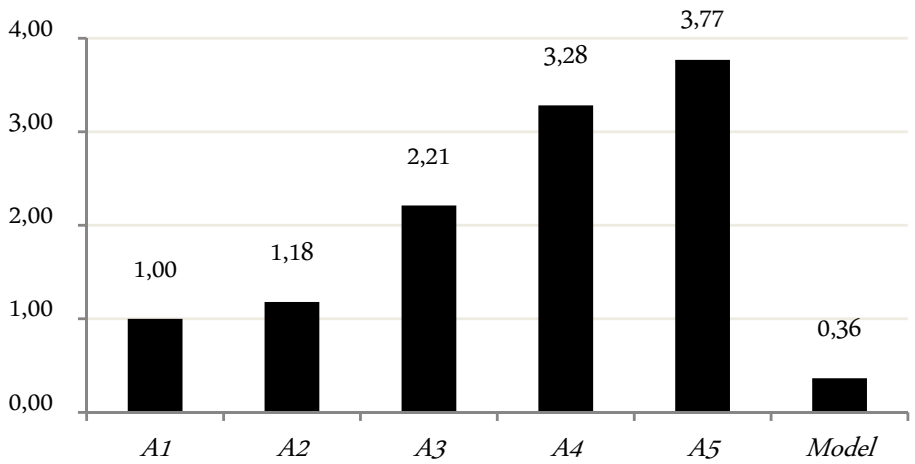
Case Study: AES Core – Levels of Abstraction

- A5 $\rightarrow$ Weitere Verfeinerung der Funktionsblöcke und Operationen aus *A4* in zyklengenaues Modell
- Alle Funktionsblöcke werden getaktet
- Synthesefähiger VHDL-Code kann aus diesem Modell generiert werden



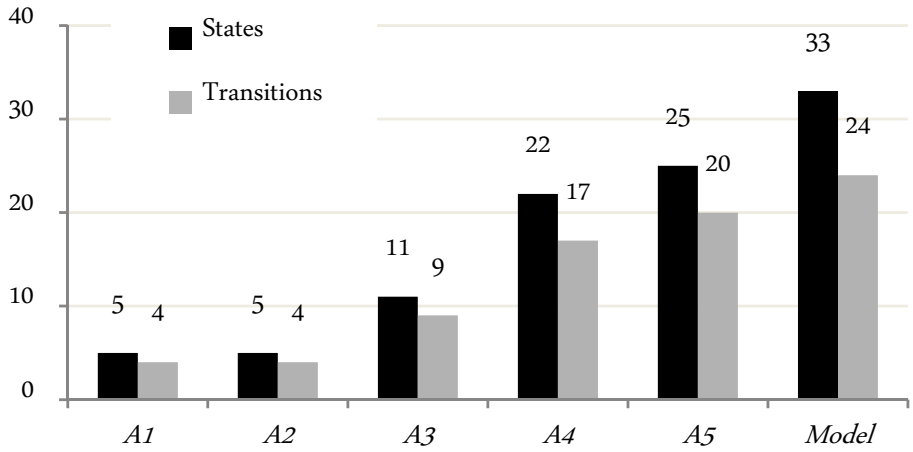
Case Study: AES Core – Results

- Source Lines of Code (SLOC) normiert auf Modell *A1* als Abschätzung für den Implementierungsaufwand
- Aufwand steigt mit dem Detaillierungsgrad
- Produktivitätssteigerung durch abstraktere Modelle → generieren der zyklengenauen Modelle



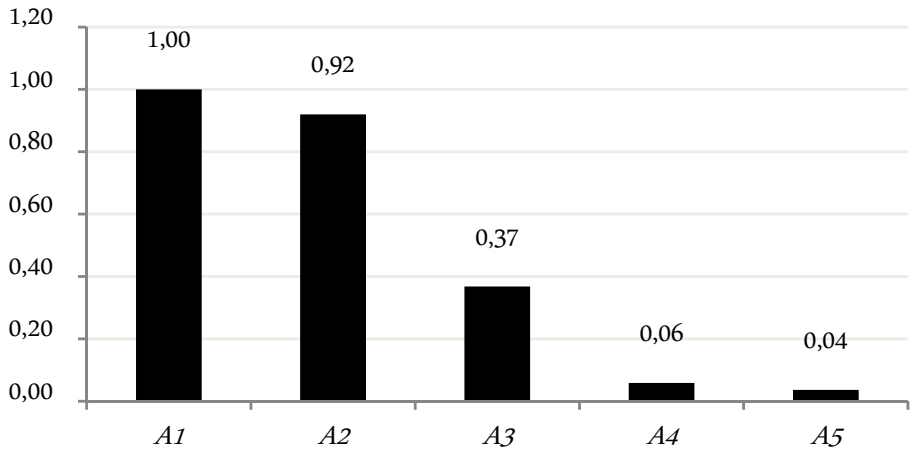
Case Study: AES Core – Results

- Zustände und Zustandsüberführungen der einzelnen Modelle



Case Study: AES Core – Results

- Verschlüsselungsdurchläufe pro Sekunde Simulationszeit normiert auf Modell *A1*
- Performance sinkt mit Detaillierungsgrad massiv
  - Algorithmische Komplexität und Optimierungen wirken sich heute kaum auf Simulationsgeschwindigkeit aus
  - Wichtig ist die Granularität der Modelle → Abbildung der Wirklichkeit vs. Simulationsgeschwindigkeit



TLM – Erkenntnisse

- Grad der Abstraktion hängt vom Blickwinkel und Designstufe ab
- Gewisse Beschränkungen sinnvoll, z.B. SoC design:
  - Bit genaues Verhalten $\rightarrow$ Schnittstellen auf Registerebene
  - Ermöglicht paralleles entwickeln der Firmware und Hardwareblöcke
- Architektur- und Performanceanalyse sehr früh im Entwicklungsprozess
- Simulationszeit verkürzen
  - Modellierung $\rightarrow$ je genauer bzw. hoher der Detaillierungsgrad desto schlechtere Laufzeit
- Verifikation:
  - Systemverifikation sehr effizient, wenn Teile abstrahiert werden können.
  - „Golden-Model“