

# Architekturen und Entwurf von Rechnersystemen

## Zusammenfassung

Fabian Damken

8. November 2023



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

---

# Inhaltsverzeichnis

---

<b>1 Bluespec</b>	<b>4</b>
1.1 Grundlagen . . . . .	4
1.1.1 Zerlegungshierarchie . . . . .	4
1.1.2 Methodenarten . . . . .	4
1.1.3 Zuweisungsoperatoren . . . . .	5
1.2 Ausführungssemantik . . . . .	5
1.2.1 WILL_FIRE . . . . .	5
1.2.2 Parallelität . . . . .	5
1.2.3 Nebenläufigkeit . . . . .	5
1.2.4 Pipelines . . . . .	6
1.3 Synthetisierung . . . . .	6
1.4 Komponenten . . . . .	6
1.4.1 FIFO . . . . .	6
1.4.2 Concurrent Registers . . . . .	7
1.4.3 Attribute . . . . .	7
<b>2 Hardware-Entwurfstechniken</b>	<b>8</b>
2.1 Entwurfsebenen (Abstraktionsgrad) . . . . .	8
2.2 Ablauf der Synthese . . . . .	8
2.3 Design Constraints (Einschränkungen) . . . . .	8
2.4 Verifikation . . . . .	9
2.4.1 Prä-Synthese-Simulation . . . . .	9
2.4.2 Post-Synthese-Simulation . . . . .	9
2.4.3 Post-Layout-Simulation . . . . .	10
<b>3 Rekonfigurierbare System-on-Chips (SoCs)</b>	<b>11</b>
3.1 ARM Cortes-A9 Prozessorkern . . . . .	11
3.2 SIMD Rechnungen mit NEON . . . . .	11
3.3 Speichersystem . . . . .	11
3.4 Programmierbare Logik . . . . .	12
3.4.1 Konfigurierbarer Logikblock . . . . .	12
3.4.2 Hard Blocks . . . . .	12
3.5 Hard Core vs. Soft Core . . . . .	12
3.6 Schnittstelle PS ↔ PL . . . . .	13
3.7 AXI4 . . . . .	13
3.7.1 Protokollfamilie . . . . .	13
3.7.2 Grundkonzepte AXI4 . . . . .	14
3.7.3 Grundlagen der Signalisierung . . . . .	14
3.8 IP-Blöcke . . . . .	15

---

<b>4</b>	<b>Entwurfsverfahren und Werkzeuge für Hardware-Beschleuniger</b>	<b>16</b>
4.1	Hardware-Klassifikation . . . . .	16
4.1.1	Hardware-Arten . . . . .	16
4.1.2	Klassifikation . . . . .	16
4.1.3	Hardware-Anforderungen . . . . .	17
4.2	Moore's Law . . . . .	17
4.3	Patterson's Walls . . . . .	17
4.4	FPGA Entwicklungsablauf . . . . .	17
<b>5</b>	<b>TaPaSCo</b>	<b>18</b>
<b>6</b>	<b>Abkürzungen</b>	<b>19</b>

---

# 1 Bluespec

---

---

## 1.1 Grundlagen

---

---

### 1.1.1 Zerlegungshierarchie

---

- Bluespec zerlegt ein großes Programm in kleinere Module
- Blätter der Hierarchie sind primitive Zustandselemente
- Auch Register sind Module!
- Module stellen Schnittstellen durch Methoden bereit
- Module enthalten Regeln, welche Methoden *anderer* Module aufrufen
- Methoden können auch Methoden *anderer* Module aufrufen

---

### 1.1.2 Methodenarten

---

Bluespec kennt die folgenden Methodenarten:

#### Value

- Können den Zustand der Schaltung nicht verändern
- Können lokale Zwischenwerte berechnen
- Geben Daten zurück

#### Action

- Können den Zustand der Schaltung ändern
- Können keine Daten zurück geben

#### Action-Value

- Können den Zustand der Schaltung ändern
- Geben Daten zurück

---

### 1.1.3 Zuweisungsoperatoren

---

Bluespec kennt die folgenden Zuweisungsoperatoren:

- = Legt den Wert einer Variable fest auf den Rückgabewert einer Methode/Funktion/...  
(Der linke Teil wird dem rechten gleichgesetzt.)
- <- Führt eine Action-Value-Methode aus und setzt die Variable auf den Rückgabewert.  
Außerdem muss dieser Operator genutzt werden, wenn Instanzen eines Moduls erstellt werden. (Der rechte Teil wird ausgeführt und danach dem linken gleichgesetzt.)
- <= Verändert den Wert eines Registers (impliziter Aufruf von `_write`).

---

## 1.2 Ausführungssemantik

---

### 1.2.1 WILL\_FIRE

---

Das `WILL_FIRE` einer Methode setzt sich aus dem Guard der Methode/Regel und den Guards der aufgerufenen Methoden.

---

### 1.2.2 Parallelität

---

Die Aktionen innerhalb einer Regel werden Echtparallel ausgeführt und *nach* der Auswertung der Regel den Registern zugewiesen. Zum Zeitpunkt des Lesens von Registern wird immer der alte Wert zurück gegeben.

Hierdurch sind die folgenden Codebeispiele identisch und tauschen die Werte der Register `x` und `y`:

```
1 rule swap;  
2   x <= y;  
3   y <= x;  
4 endrule
```

(a) Vertausche `x` und `y`

```
1 rule swap;  
2   y <= x;  
3   x <= y;  
4 endrule
```

(b) Vertausche `y` und `x`

---

### 1.2.3 Nebenläufigkeit

---

---

#### Ablaufplan

---

**Ausführungsreihenfolge** Es werden möglichst viele Regeln *innerhalb eines Taktes*, aber *nicht zeitgleich* ausgeführt, solange dies nicht zu Konflikten führt.

**Beeinflussung** Der Ablaufplan kann mit einigen Attributen beeinflusst werden, siehe 1.4.

---

#### Konflikte

---

Bei der der Erstellung des Ablaufplans kann es zu Konflikten kommen, da es (bei manchem Methoden) Einschränkungen gibt, in welcher Reihenfolge diese ausgeführt werden:

---

Einschränkung	Bedeutung: Regeln mit Aufrufen von mA und mB können ...
$mA \text{ konfliktfrei } mB$	...nebenläufig feuern (beliebige Reihenfolge).
$mA < mB$	...nebenläufig feuern, solange mA vor mB ausgeführt wird.
$mB < mA$	...nebenläufig feuern, solange mB vor mA ausgeführt wird.
$mA \text{ konflikt } mB$	...nicht nebenläufig feuern.

## Häufige Ursachen

**Rule Ordering Conflict** Zustandselemente können nur einmal je Takt umschalten

- Lesen eines geänderten Zustandes im selben Takt ist nicht (ohne weiteres) möglich

**Rule Resource Conflict** Hardware-Ressourcen (z.B. Drähte) können nur einmal je Takt genutzt werden

---

### 1.2.4 Pipelines

#### Dynamische Pipeline

- Latenz ist nur datenabhängig variabel
- Gegenteil: statische Pipeline

#### Elastische Pipeline

- Daten bewegen sich mit unterschiedlichem Fortschritt durch die Pipeline
- Gegenteil: inelastische/starre Pipeline

---

## 1.3 Synthetisierung

Die Synthetisierung kann mit einigen Attributen beeinflusst werden, siehe 1.4.

---

## 1.4 Komponenten

---

### 1.4.1 FIFO

#### Normale FIFO

- Ist die FIFO leer, ist kein deq möglich (auch wenn zeitgleich ein enq stattfindet).
- Ist die FIFO voll, ist kein enq möglich (auch wenn zeitgleich ein deq stattfindet).

#### Pipeline FIFO

- Ist die FIFO leer, ist kein deq möglich.
- Ist die FIFO voll, ist enq möglich, wenn zeitgleich ein deq stattfindet. `first` liefert in dem Falle den alten Wert (vor enq).

---

## Bypass FIFO

- Ist die FIFO leer, ist `deq` möglich, wenn zeitgleich in `enq` stattfindet. `first` liefert in dem Falle den neuen Wert (nach `enq`).
- Ist die FIFO voll, ist kein `enq` möglich.

---

## 1.4.2 Concurrent Registers

---

- Halten eine Historie von Werten innerhalb eines Taktes
- Hierdurch ist mehrmaliges Schreiben und Lesen nach Schreiben möglich.
- Präzedenzrelation

```
1 Reg#(Bool) ports[N] <- mkCReg(N, False);
2
3 ports[0]._read < ports[0]._write <
4 ports[1]._read < ports[1]._write <
5 (* $ \cdots $ *)
6 ports[N - 1]._read < ports[N - 1].write
```

---

## 1.4.3 Attribute

---

Attribute werden in Runden Klammern mit Sternchen angegeben. Beispiel: `(* <attribute> *)`

---

### Scheduling Attribute

---

**descending\_urgency** Reihenfolge/Priorität der Berechnung der `WILL_FIRE` Bedingungen (Dringlichkeit).

**execution\_order** Reihenfolge/Priorität der Ausführungsreihenfolge des Regelkörpers (Frühzeitigkeit).

**preempts** Erlaubt einer gefeuerten Regel, das Ausführen anderer Regeln zu unterdrücken (auch wenn kein Konflikt vorliegt).

**mutually\_exclusive** Zusicherung an dem Compiler, dass zwei Regelbedingungen *niemals zeitgleich* wahr sind (bspw. bei One-Hot-Kodierungen).

---

### Synthesisierung Attribute

---

**synthesize** Verhindert das Inlining eines Moduls. Kann nur angewandt werden auf Module, welche ausschließlich Schnittstellen mit Bits, Skalaren und Bit-Vektoren hat (da Verilog weniger mächtig ist als Bluespec).

---

## 2 Hardware-Entwurfstechniken

---

---

### 2.1 Entwurfsebenen (Abstraktionsgrad)

---

1. Verhaltensebene
  - Was soll passieren? Die Realisierung bleibt offen.
2. Systemebene
  - Grobe Aufteilung von Struktur, Zeit, Daten und Kommunikation (CPU, FPGA, DRAM, ...)
3. Register-Transfer-Ebene (RTL, Register-Transfer-Logic)
  - Synchron, getaktet
4. Logik- oder Gatterebene
  - Netze aus Gattern, Flip-Flops, ...
5. Transistorebene
  - Elektrischer Schaltplan
6. Layoutebene
  - Maßstabsgetreue geometrische Anordnung des Chips mit verschiedenen Schichten (3D)

---

### 2.2 Ablauf der Synthese

---

Siehe 2.1.

---

### 2.3 Design Constraints (Einschränkungen)

---

- Zeit
  - Timing-Analyse
  - Geschätzt nach Synthese (*ohne* Verdrahtungsverzögerung)
  - Exakt nach Platzieren und Verdrahten
  - → Layoutebene
- Fläche
  - Geschätzt nach Synthese (*ohne* Verdrahtungsfläche)
  - Exakt nach Platzieren und Verdrahten



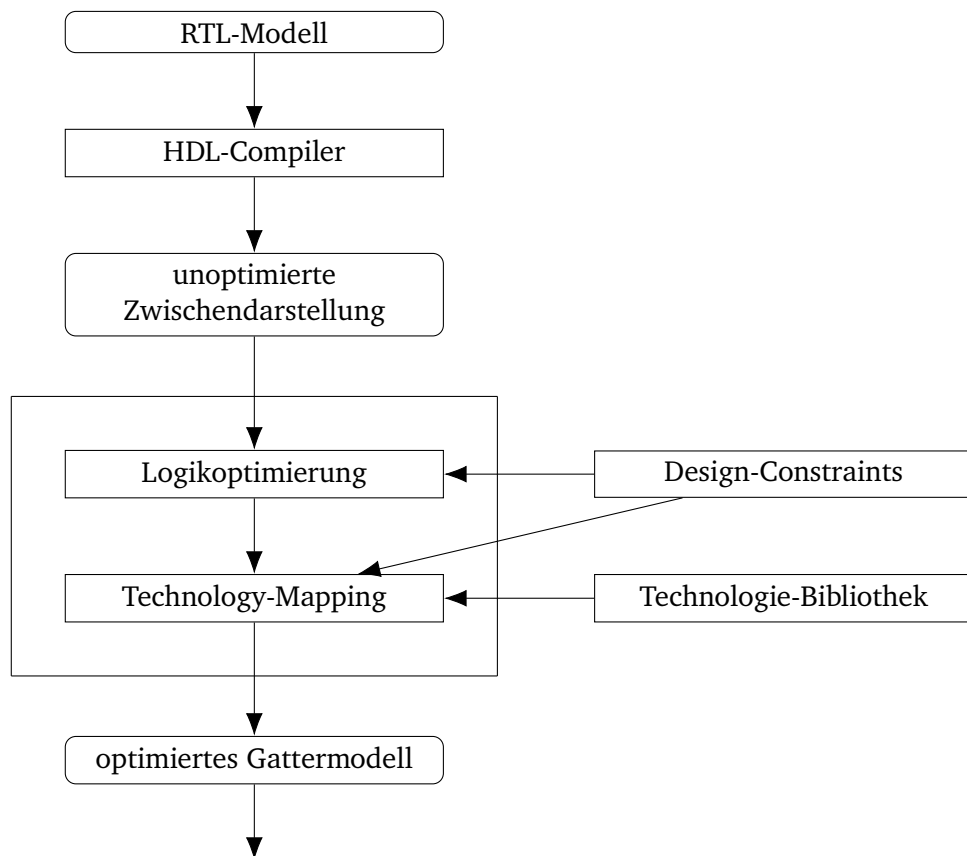


Abbildung 2.1: Ablauf der Synthese

- Elektrischer Leistungsaufnahme
  - Simulation auf Layout-Ebene
  - Bestimmung von Umschaltfrequenzen (*toggle frequencies*) von Signalen

---

## 2.4 Verifikation

---

### 2.4.1 Prä-Synthese-Simulation

---

- Testen der Implementierung in der HDL (Simulation)

### 2.4.2 Post-Synthese-Simulation

---

- Gleiche Testdaten wie bei der Prä-Synthese-Simulation
- Vergleichen der Ergebnisse
- Vergleiche sind Aufgrund des Zeitverhaltens nicht trivial
  - Unterschiedlich viele Ergebnisse
  - Verschiedene Werte

- 
- Unterschiedliche Zeiten
  - Interpretation nötig

---

### **2.4.3 Post-Layout-Simulation**

---

- Gleiche Testdaten wie bei der Prä-/Post-Synthese-Simulation
- Gleiches Vorgehen wie bei der Post-Synthese-Simulation
- Schließt Gatter- und Leitungsverzögerungen mit ein
- Kann auch Widerstände, Kapazitäten, Induktivitäten, ... enthalten

---

## 3 Rekonfigurierbare System-on-Chips (SoCs)

---

---

### 3.1 ARM Cortes-A9 Prozessorkern

---

- Superskalar Out-of-Order
- Holt zwei Instruktionen je Takt
- Kann je Takt bis zu vier Instruktionen ausführen
  - ALU/MUL
  - ALU
  - FPU/SIMD
  - Load-Store
- Mehr ILP (Instruction Level Parallelism) durch:
  - Umbenennung von Registern (*renaming*)
  - Dynamisches Vorziehen von unabhängigen Instruktionen (*out-of-order execution*)

---

### 3.2 SIMD Rechnungen mit NEON

---

- *Single Instruction Multiple Data* (die gleiche Operation wird auf mehreren Daten ausgeführt)
- Wird auch als normale FPU (auf skalaren Daten) genutzt

---

### 3.3 Speichersystem

---

- Prefetching
  - Beobachte Speicherzugriffe des Prozessors
  - Holt die Daten schon mal „vorweg“ in L1-D\$
  - Beobachtet dabei bis zu acht unabhängige Datenströme
- Cache-Kohärenz
  - Problem
    - \* Prozessorkerne haben eigenen L1-Cache
    - \* Greifen aber auf den gemeinsamen Hauptspeicher und L2-Cache zu
  - Prozessorkerne müssen austauschen, wo welcher Wert liegt
  - Aufgabe wird von Snoop-Control-Unit übernommen

- 
- \* Überwacht Speicherzugriffe
  - \* Gibt aktuellen Wert weiter
  - \* Protokoll: MESI (Modified/Exclusive/Shared/Invalid)
  - On-Chip-Memory
    - 256 KB SRAM direkt auf dem Chip
    - Nicht sonderlich schnell (1.400 MB/s) (Vergleich: Externes 32b DDR3-SDRAM 1066 liefert 3.700 MB/s)
    - Aber sehr geringe Latenz (32 bis 34 Taktzyklen) (Vergleich: Externer 32b DDR3-SDRAM hat 37 bis 200 Zyklen)

---

## 3.4 Programmierbare Logik

---

### 3.4.1 Konfigurierbarer Logikblock

---

- Je zwei Slices mit je 4 LUTs und 8 Flip-Flops
- Verbunden mit der Switch Matrix

---

### 3.4.2 Hard Blocks

---

- Integrierter Speicher (BRAM)
  - Je 36 Kb Speicher
  - Wortbreite konfigurierbar
  - Alle Blöcke parallel zugreifbar
- Integrierter Multiplizierer (DSP)

---

## 3.5 Hard Core vs. Soft Core

---

**Hard Core** In ASIC erstellter Prozessor.

**Soft Core** Auf einem FPGA realisierter Prozessor.

### Vergleich Hard/Soft Core

- Soft Cores in rekonfigurierbarer Logik
  - brauchen deutlich mehr Chip-Fläche als Hard Cores
  - laufen wesentlich langsamer.
- Einsatz von Soft Cores i.d.R. nicht mehr effizient, Trend geht zu Kombination von
  - Hard Core CPUs
  - Rekonfigurierbarer Logik

---

## 3.6 Schnittstelle PS ↔ PL

---

### General Purpose Interface (GP)

- Generelles AXI-Interface für (fast) alle Anforderungen
- Direkter Zugriff auf den Speicher
- Anschluss für Peripherie
- Hat 2 Master und 2 Slave-Ports (bidirektional)
- Cache-inkohärent zu der CPU

### High Performance Interface (HP)

- Hochperformantes AXI-Interface für schnelle Datenübertragung
- Direkter Zugriff auf den Speicher
- Kein Peripherie-Anschluss, daher geringe Belastung
- Cache-inkohärent zu der CPU
- Zugriff immer direkt auf den RAM, belastet nicht den L2-Cache der CPU

### Amber Coherence Protocol Interface (ACP)

- Zugriff auf die SCU (Snoop-Control-Unit)
- Daher Beibehaltung der Cache-Kohärenz
- Liefert schnell Daten, solange diese im Cache liegen (sonst sehr langsam)
- Teilt sich L2-Cache mit der CPU

---

## 3.7 AXI4

---

### 3.7.1 Protokollfamilie

---

#### AXI4

- Mächtigste Implementierung
- Benötigt am meisten Chipfläche
- Unterstützt memory-mapped I/O
- Erlaubt effiziente Übertragung von Datengruppen (*burst transfer*)

---

### AXI3-Lite

- Einfacher als AXI4
- Benötigt weniger Chipfläche als AXI4
- Unterstützt memory-mapped I/O
- Erlaubt keine Burst-Transfers, sondern nur einzelne Daten

### AXI4-Stream

- Spezialisierte Realisierung (z.B. für Sensoren)
- Überträgt nur reine Datenströme (keine Adressen und somit kein memory-mapped I/O)
- Unbegrenzt lange Bursts
- Unidirektionale Übertragung von Master zu Slave

---

### 3.7.2 Grundkonzepte AXI4

- Master löst Übertragung aus, Slave reagiert auf die Übertragung
- Slave liefert Daten an Master bei Lesezugriff
- Slave nimmt Daten vom Master entgegen bei Schreibzugriff
- Slave liefert Status an Master (über Lesekanal oder extra Rückkanal für Antworten bei Schreibzugriffen)

Bei einer Burst-Übertragung setzt der Master nur die Start-Adresse, der Slave muss die Folgeadressen selbst bestimmen (bspw. durch inkrementieren).

Die maximale Länge eines Bursts ist bei AXI4 256 Datensätze (genannt *beats*). Ein Beat kann 1 bis 128 Bytes umfassen.

---

### 3.7.3 Grundlagen der Signalisierung

- Handshake zwischen Quelle und Ziel von Daten, ausgewertet bei @posedge
  - Quelle setzt VALID, wenn gültige Daten anliegen
  - Ziel setzt READY, wenn Daten übernommen werden können
  - Sind VALID und READY zeitgleich gesetzt, wurden die Daten übernommen → READY fällt.
- Es existieren die folgenden Signale (Kanal → Handshake Paar):

**Write Address Channel** AWVALID, AWREADY

**Write Data Channel** WVALID, WREADY

**Write Response Channel** BVALID, BREADY

**Read Address Channel** ARVALID, ARREADY

**Read Data Channel** RVALID, RREADY

---

## 3.8 IP-Blöcke

---

- „Intellectual Property“
- Vordefinierte Hardware-Funktionen
- Meist stark konfigurierbar

---

## 4 Entwurfungsverfahren und Werkzeuge für Hardware-Beschleuniger

---

---

### 4.1 Hardware-Klassifikation

---

---

#### 4.1.1 Hardware-Arten

---

**ASIC** Application Specific Integrated Circuit

**microController** Instruction Set Architecture (ISA), limitierter Einsatzbereich

**System-on-Chip** Kleinskalierte Architekturen

**Low-Power-CPU** Desktop-CPUs mit Fokus auf Energieeffizienz

**Multi-Core CPU/SoC** Desktop und Server CPUs, SoCs

**GPGPU** General Purpose GPU

**Many-Core CPU** Massivparallele CPUs

**DSP** Digital Signal Processor, Massivparallele Arithmetikeinheiten

**FPGA** Field Programmable Logic Gate Array

---

#### 4.1.2 Klassifikation

---

**Commodity ISAs** microController, LPCPU, Multi-Core CPUs, SoCs

- Standardisiertes Instruction Set
- Starke Unterstützung durch Tools (Compiler, Debugger, ...)
- Programmierbar in Standardsprachen (C, C++, ...)

**Specialized ISAs** GPGPUs, DSPs, Many-Cores

- Nicht-Standardisiertes Instruction Set
- Limitierter Unterstützung durch Tools (Herstellertools)
- Programmierbar in spezialisierten Sprachen (OpenMP, CUDA, OpenCL, ...)

**Reconfigurable Technology** PLDs, FPGAs

- Eigenes Design, kein Standard
- Limitierte Unterstützung durch Tools (Herstellertools)



- Setzt HDLs voraus
- Limitierte Unterstützung von Standardsprachen und spezialisierten Sprachen (C, C++, OpenCL, ...)

**ASICs** Applikationsspezifische Hardware (bspw. Bitcoin Mining, Verschlüsselung)

- Eigenes Design, keine Standards
- Kein Compilersupport (OS-Integration, APIs, ... sind selber zu entwickeln)
- Setzt HDLs voraus
- Nur Programmierbar in Low-Level Sprachen, bzw. Hardware-Schnittstellen

---

### 4.1.3 Hardware-Anforderungen

---

## 4.2 Moore's Law

---

**Aussage** Die Anzahl der Transistoren pro Fläche verdoppelt sich alle zwei Jahre.

Hat sich 10 Jahre gehalten!

### Ende von Moore's Law

- 2014 wurden 14nm nur knapp erreicht
- 2017 wurden 10nm nicht erreicht
- 2020 sollen 5nm erreicht werden, sehr fraglich

---

## 4.3 Patterson's Walls

---

**Aussage** Power Wall + Memory Wall + ILP Wall = Brick Wall

Wenn man zwei der Mauern durchbricht, scheitert man an der dritten.

Des hat sich für alle bisherigen Computersysteme bewahrheitet (Energieeffizienz vs. Performanz, Performanz vs. Speicher, ...).

---

## 4.4 FPGA Entwicklungsablauf

---

1. Auswahl oder Bau des FPGAs
2. Erstellung oder Anpassung des Basisdesigns
3. Implementierung der Logik
4. Simulation des Designs
5. Platzieren und Verdrahten
6. Testen auf echter Hardware

Dabei werden die Schritte 1 und 2 als „Hardware Bring-Up“, die Schritte 3 und 4 als „Logikdesign“ und die Schritte 5 und 6 als „Hardware Synthese“ bezeichnet.



---

## 5 TaPaSCo

---

---

## 6 Abkürzungen

---

<b>ACP</b>	Amber Coherence Protocol
<b>ALU</b>	Arithmetic Logical Unit
<b>AMBA</b>	Advanced Microcontroller Bus Architecture
<b>API</b>	Application Programming Interface
<b>APU</b>	Application Processing Unit
<b>ASIC</b>	Application Specific Integrated Circuit
<b>AXI</b>	Advanced eXtensible Interface
<b>BRAM</b>	Block RAM
<b>BSV</b>	Bluespec System Verilog
<b>CE</b>	Clock Enable
<b>CF</b>	Conflict Free
<b>CLB</b>	Configurable Logic Blocks
<b>CLK</b>	Clock
<b>COTS</b>	Commercial Off The Shelf
<b>CPU</b>	Central Processing Unit
<b>DDR</b>	Double Data Read
<b>DMA</b>	Direct Memory Access
<b>DRAM</b>	Direct RAM
<b>DSP</b>	Digital Signal Processors
<b>FIFO</b>	First In - First Out Queue
<b>FPGA</b>	Field Programmable Gate Array
<b>FPU</b>	Floating Point Unit
<b>GALS</b>	Global Asynchronous, Locally Synchronous
<b>GPGPU</b>	General Purpose GPU

---

**GPIO** General Purpose IO

**GPU** Graphics Processing Unit

**GUI** Graphical User Interface

**HDL** Hardware Description Language

**HLS** High-Level Synthesis

**HP** High Performance Ports

**HPC** High Performance Computing

**HW** Hardware

**IC** Integrated Circuit

**IDE** Integrated Development Environment

**ILP** Instruction Level Parallelism

**IO** Input/Output

**IP** Intellectual Property

**IPC** Instructions Per Cycle

**ISA** Instruction Set Architecture

**L1D** Level 1 Data Cache

**LCA** Lumped Circuit Abstraction

**LPCPU** Low Power CPU

**LUT** Look Up Table

**MESI** Modified/Exclusive/Shared/Invalid Protocol

**MIPS** Microprocessor without Interlocked Pipeline Staged

**MMU** Memory Management Unit

**MUX** Multiplexer

**NRC** Non-Recurring Costs

**OCM** On-Chip Memory

**OoO** Out of Order (Execution)

**OS** Operating System

**PC** Program Counter

**PCB** Process Control Block

---

**PCI** Peripheral Component Interconnect

**PCIe** PCI Express

**PE** Processing Elements

**PL** Programmable Logic

**PLD** Programmable Logic Device

**PS** Processing System

**QFN** Quad Flat No Leads Package

**QFP** Quad Flat Package

**RAM** Random Access Memory

**RAW** Read after Write

**RDY** Ready

**RISC** Reduced Instruction Set Computer

**RLDRAM** Reduced Latency DRAM

**RTL** Register-Transfer-Ebene

**SAMD** Shared Address Multiple Data

**SCU** Snoop Control Unit

**SD** Secure Digital Memory Card

**SIMD** Single Instruction Multiple Data

**SoC** System on Chip

**SOP** Small Outline Package

**SSE** Streaming SIMD Extension

**SSH** Secure Shell

**SW** Software

**TaPaSCo** Task Parallel System Composer

**TLB** Translation Lookaside Buffer

**TTL** Transistor-Transistor Logik

**TTM** Time-to-Market

**USB** Universal Serial Bus

**VGA** Video Graphics Array

**VHDL** Very High Speed Integrated Circuit Hardware Description Language