TECHNISCHE UNIVERSITÄT DRESDEN

Analyse verschiedener HLS-Systeme in Hinblick auf ihren Umgang mit der Hochsprachenabstraktion Speicher

Sascha Kath

Dresden, 13.08.2015

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Gliederung

1. Motivation & Zielstellung
2. HLS-Systeme
3. Benchmark-Funktionen
4. Auswertung
5. Zusammenfassung & Ausblick

1. MOTIVATION & ZIELSTELLUNG

1. Motivation

- Erhöhung des Abstraktionsniveaus
  - Entwicklungszeit/Kosten senken
  - Bessere Wartbar- und Erweiterbarkeit
  - Weniger spezielles Wissen nötig
  - Leichterer Einstieg

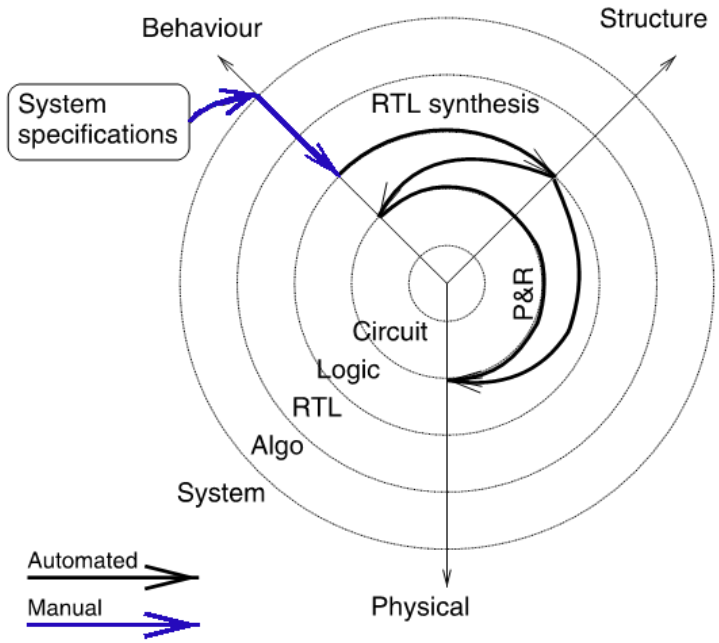
  → Entwicklung stark vereinfacht

1. Zielstellung

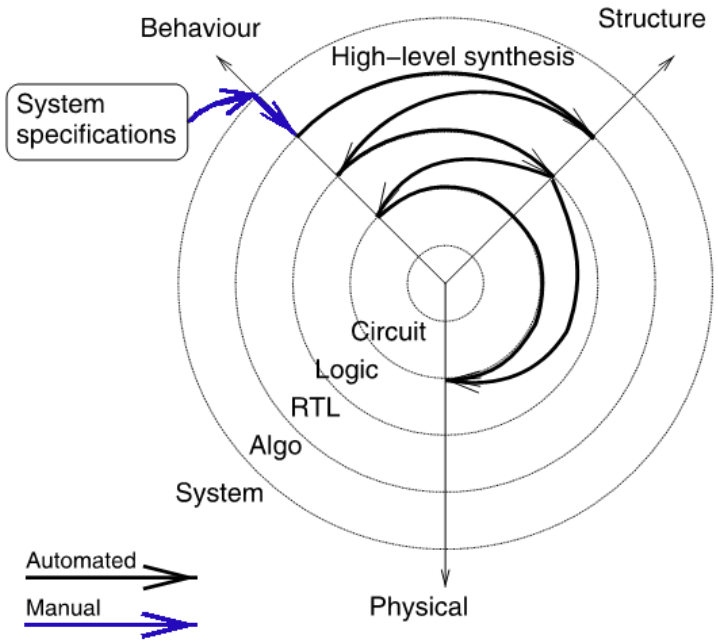
- Variablen → Speicherzuordnung
- Speicherverwaltung und -anbindung
- Benchmark-Funktionen erstellen und implementieren
- Ergebnisvergleich

2. HLS-SYSTEME

2.0 HLS-Systeme - Überblick



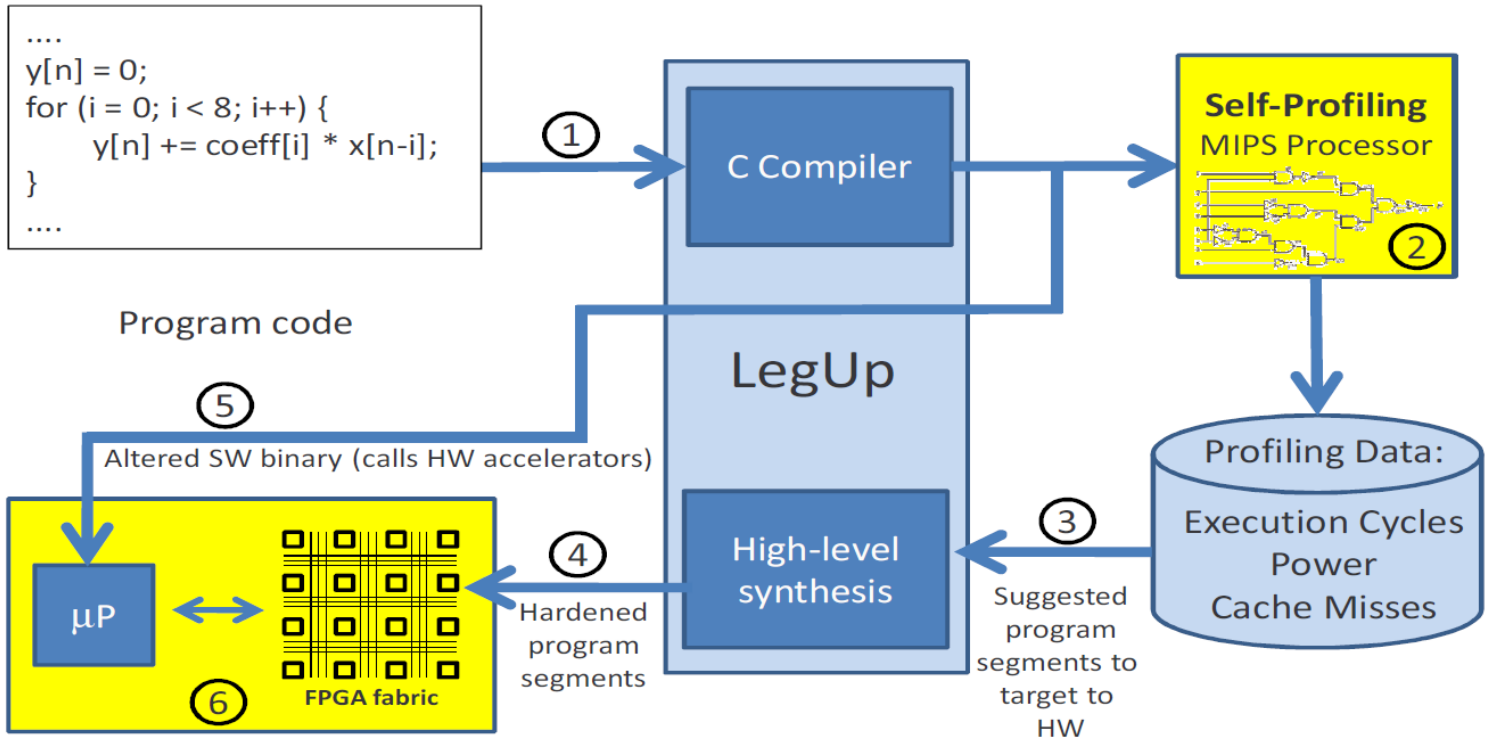
(a) RTL design flow



(b) HLS design flow

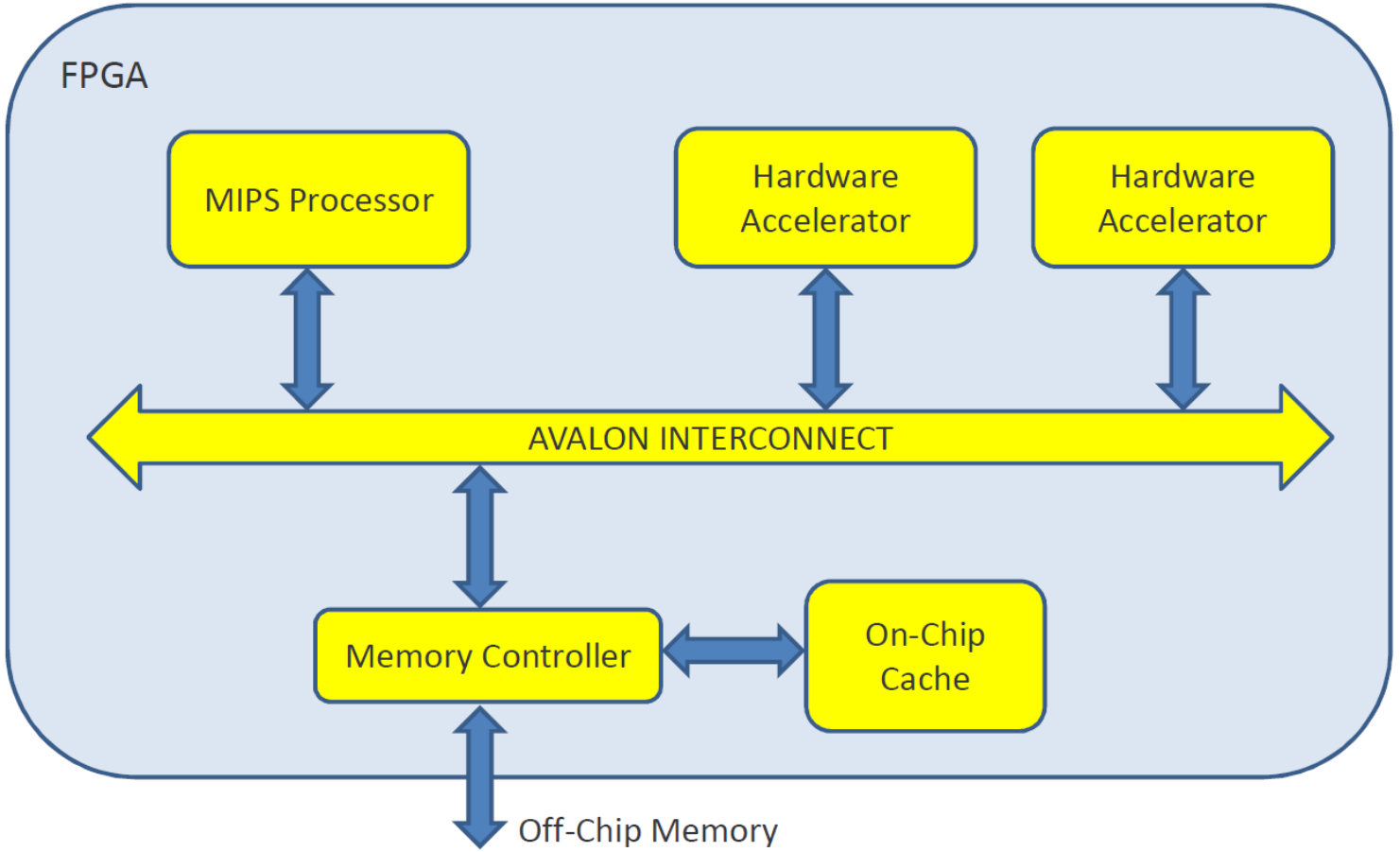
[1]

2.1 LegUp


....
y[n] = 0;
for (i = 0; i < 8; i++) {
y[n] += coeff[i] * x[n-i];
}
....
1
C Compiler
Self-Profiling
MIPS Processor
2
Program code
LegUp
5
Altered SW binary (calls HW accelerators)
Profiling Data:
Execution Cycles
Power
Cache Misses
3
High-level synthesis
4
Hardened program segments
Suggested program segments to target to HW
µP
6
FPGA fabric

[2]

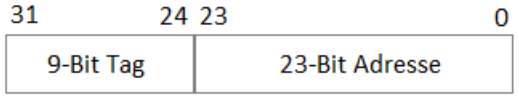
2.1 LegUp - Systemarchitektur



[2]

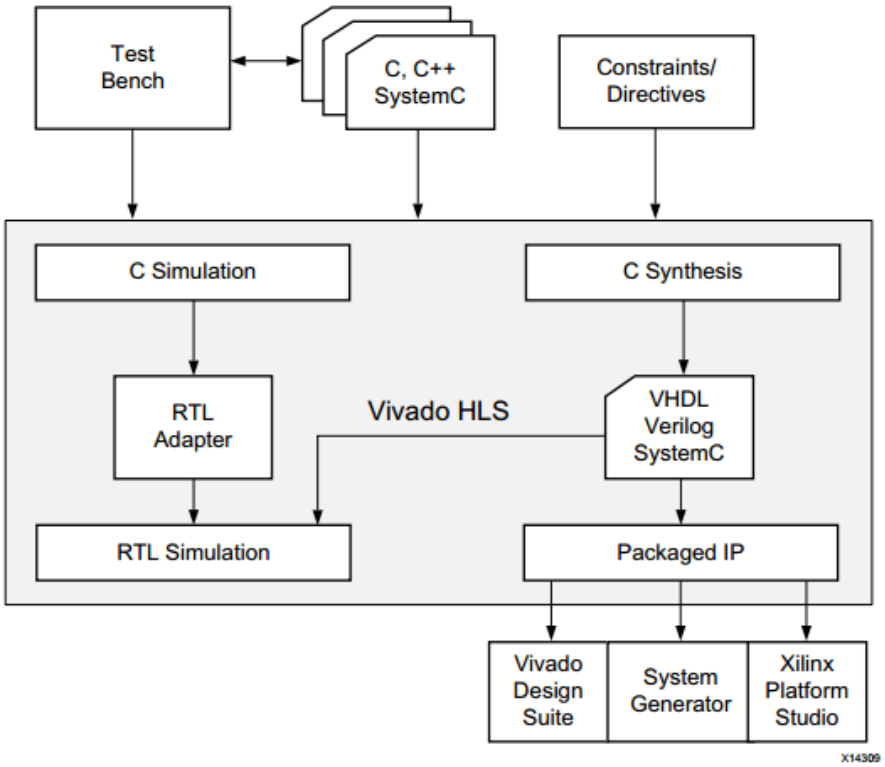
2.1 LegUp - Speicherverwaltung

- Speicherkontroller:
  - Gepiplined (3 Takte)
  - Zwei Ports
  - gemeinsame Speichernutzung
  - Pointerverwaltung
  - 32-Bit Adressformat:



- Ein Altsync RAM pro Variable

2.2 Vivado HLS



[3]

2.2 Vivado HLS - Speicherarchitektur

- Skalare Variablen:
  - in Registern gespeichert
  - Zugriff mittels dedizierter Kanäle
- Arrays:
  - Funktionsargument der Top-Level-Funktion:
    - Annahme: Array liegt außerhalb des Designs → erzeugt Datenport, Adressport und Chip-Enable, bzw. Write-Enable Signale
  - Register, Schieberegister, FIFO oder BRAM
  - Speicherverwaltung durch spezielles Speichermodul
  - Zugriff mittels dedizierte Kanäle

2.3 Leap

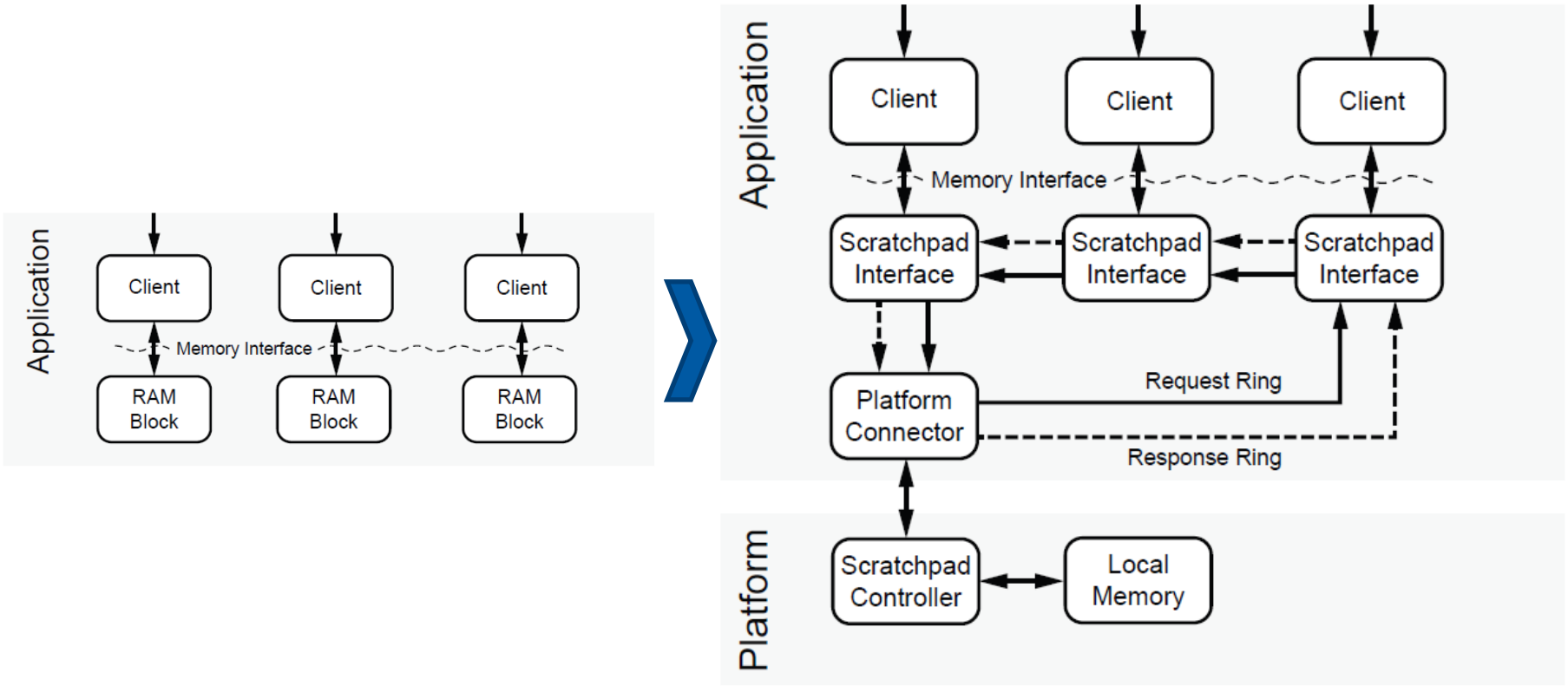
- LEAP = LINC-based Environment for Application Programming

- Grundlegende Konzepte:

1. Flexibles Intermodul-Kommunikations-Paradigma
   - Programm = Kollektion von latency-insensitive Modulen
   - Intermodul-Kommunikation über latency-insensitive Channels (FIFOs)

2. Allgemeines Speicher-Paradigma
   - Speicher willkürlicher Größe (private oder shared)

2.3 Leap - Scratchpads



a) Private RAM-Blöcke

b) Scratchpads mit Ringverbindung [4]

2.4 Vergleich der HLS-Systeme

Systeme / Kriterien	LegUp	Vivado HLS	LEAP
Eingabesprache	ANSI C	C, C++, SystemC	BSV
Einschränkungen	Keine dyn. Konstrukte im HW-Teil	Keine dyn. Konstrukte	niedrigeres Abstraktionsniveau als in C
Ausgabeformate	Verilog	Verilog, VHDL, SystemC, exportierbar als IP-Core	Verilog
Zielplatform	Alteras Cyclone II	Alle Xilinx Boards	Xilinx: XUPV5, ML605, VC707 Altera: Arria II Dev. Board, DE2, DE3, DE4

Systeme / Kriterien	LegUp	Vivado HLS	LEAP
Steuerbarkeit	➢ volatile-Typqualifikator ➢ Compiler-Flags ➢ Eingriff in LegUp-Implementierung (Open-Source)	➢ Direktiven und Constraints	➢ Durch Verwendung spezieller Module
Architektur des synthetisierten Programms	➢ Soft-Prozessor und Accelerator-Funktionen ➢ über einen Bus verbunden ➢ Speicherhierarchie (eigener und geteilter Speicher)	➢ Top Funktion wird als Modul umgesetzt ➢ FSM zur Ablaufsteuerung ➢ Speichermodule für Zugriffe auf Speicher	➢ Latency-Insensitive (LI) Module ➢ LI- Channels zur Verbindung

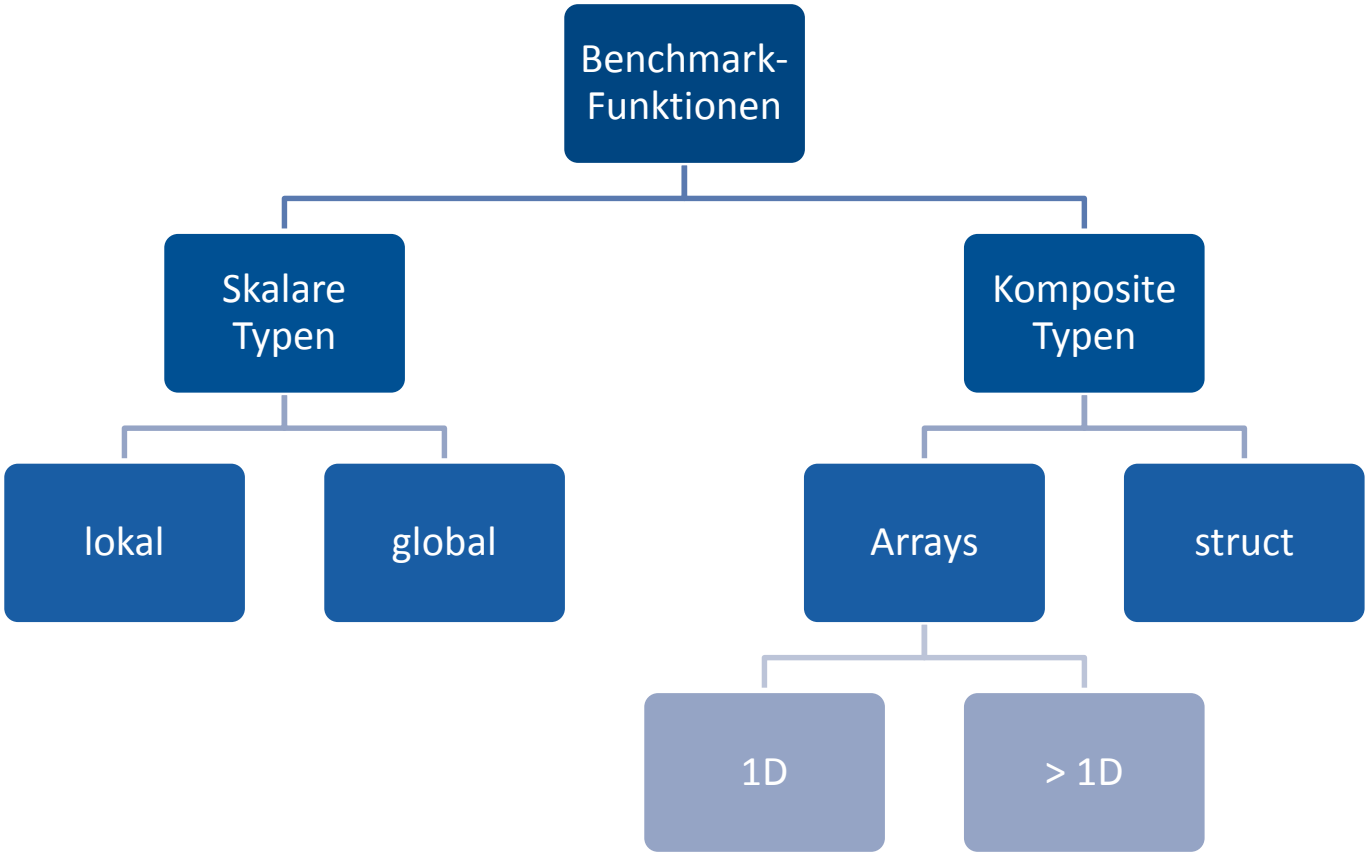
3. BENCHMARK-FUNKTIONEN

3.1 Vorbetrachtungen

- LEAP nicht weiter betrachtet
  - Speicherverwendung durch Entwickler entschieden
  - Eher Hardware-OS oder Framework für FPGAs als HLS-System

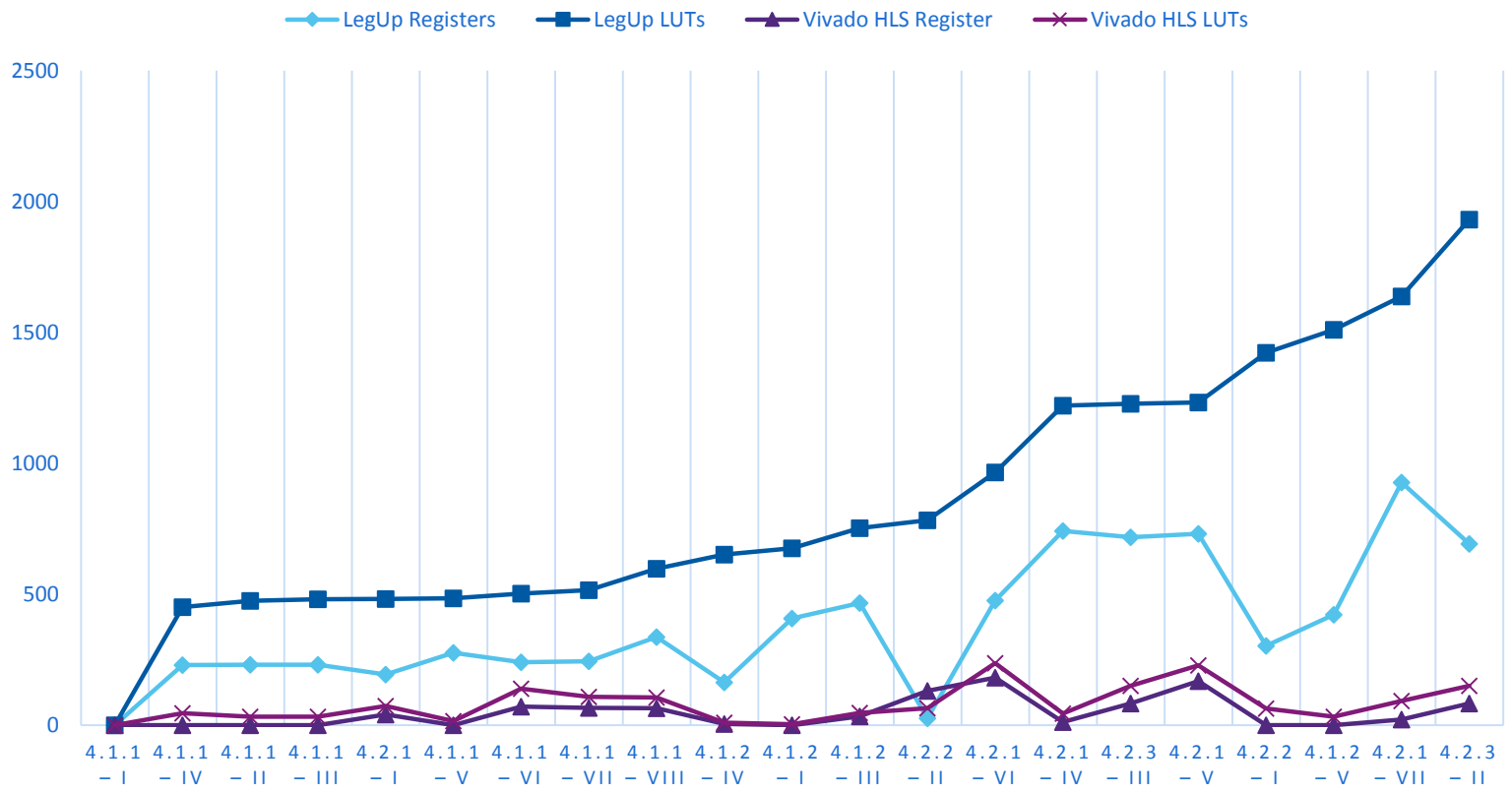
- Vergleichbarkeit Vivado HLS ⟷ LegUp
  - Verwendung Hybrid-Flow von LegUp
    - Nur beschleunigte Funktion in Hardware
    - Main Funktion durch Softprozessor abgearbeitet

3.1 Funktionskategorien

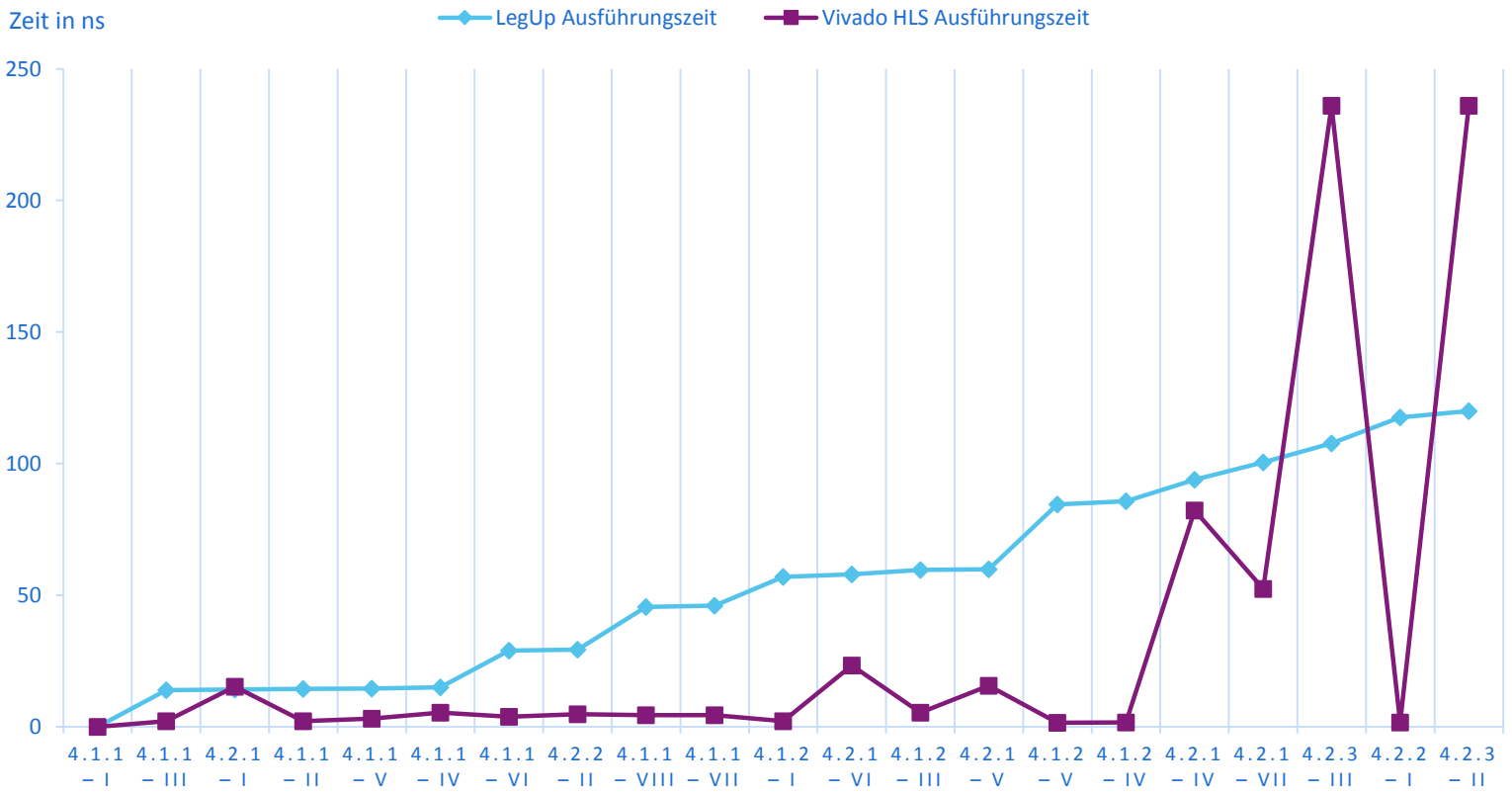


4. AUSWERTUNG

4.1 Ressourcenverbrauch

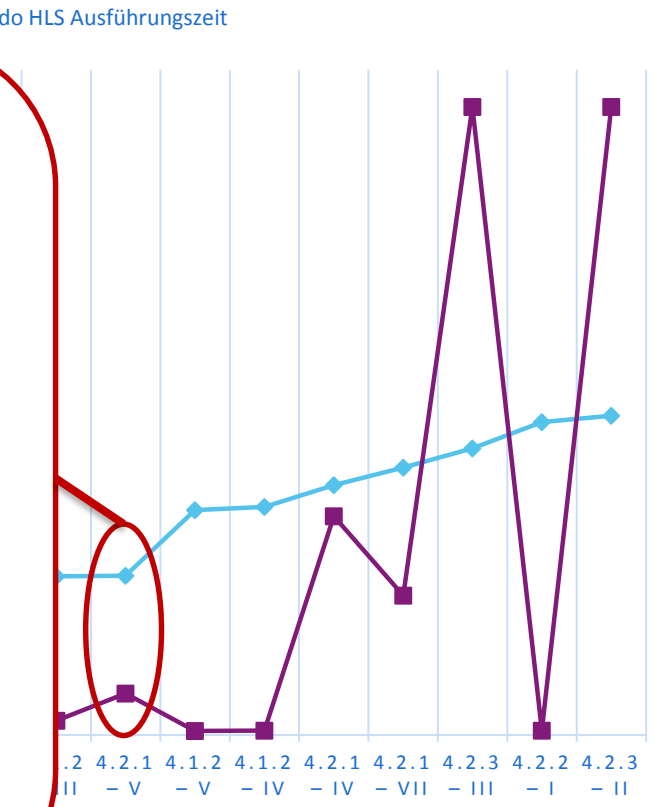


4.1 Ausführungszeit

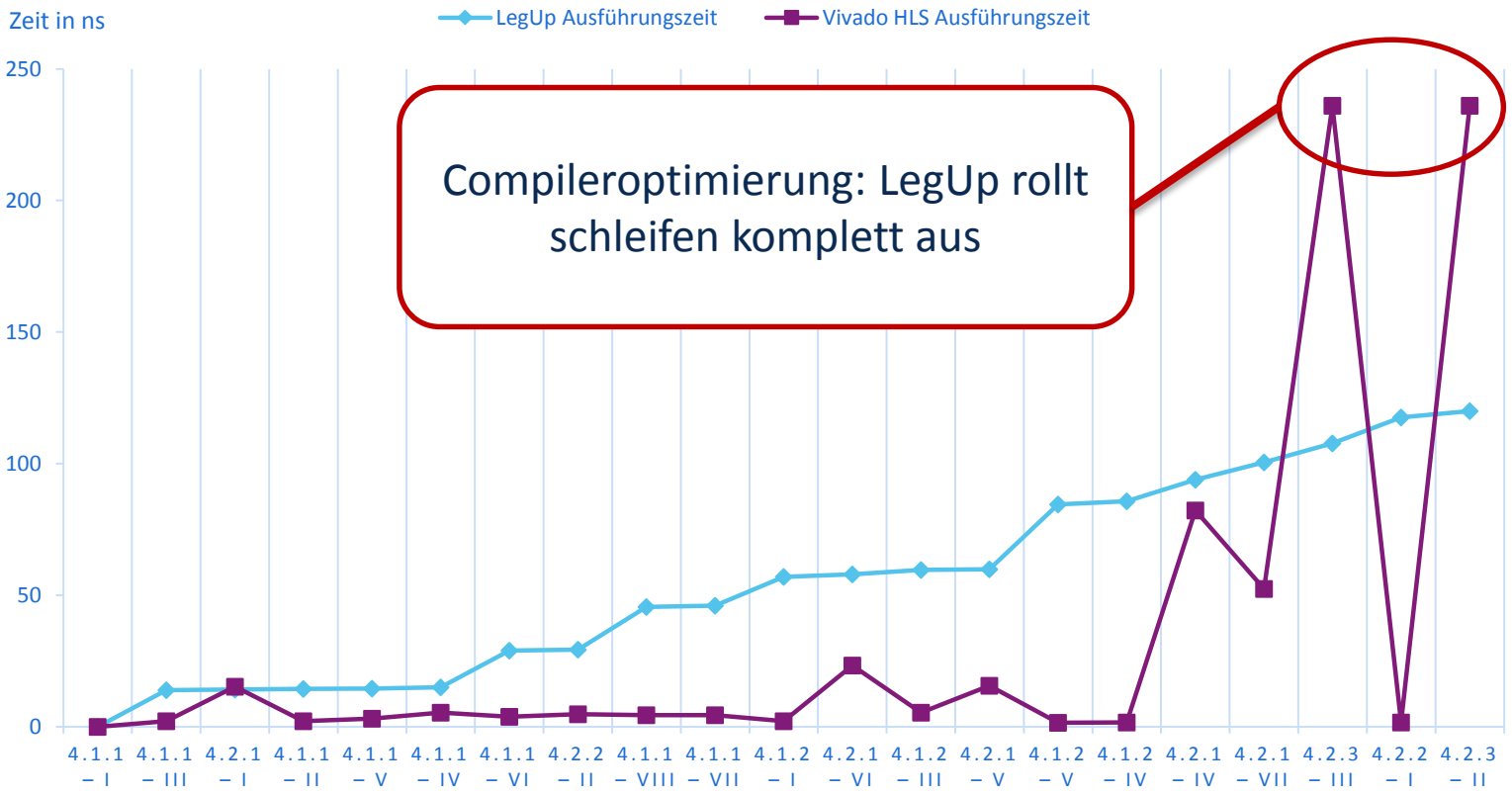


4.1 Ausführungszeit

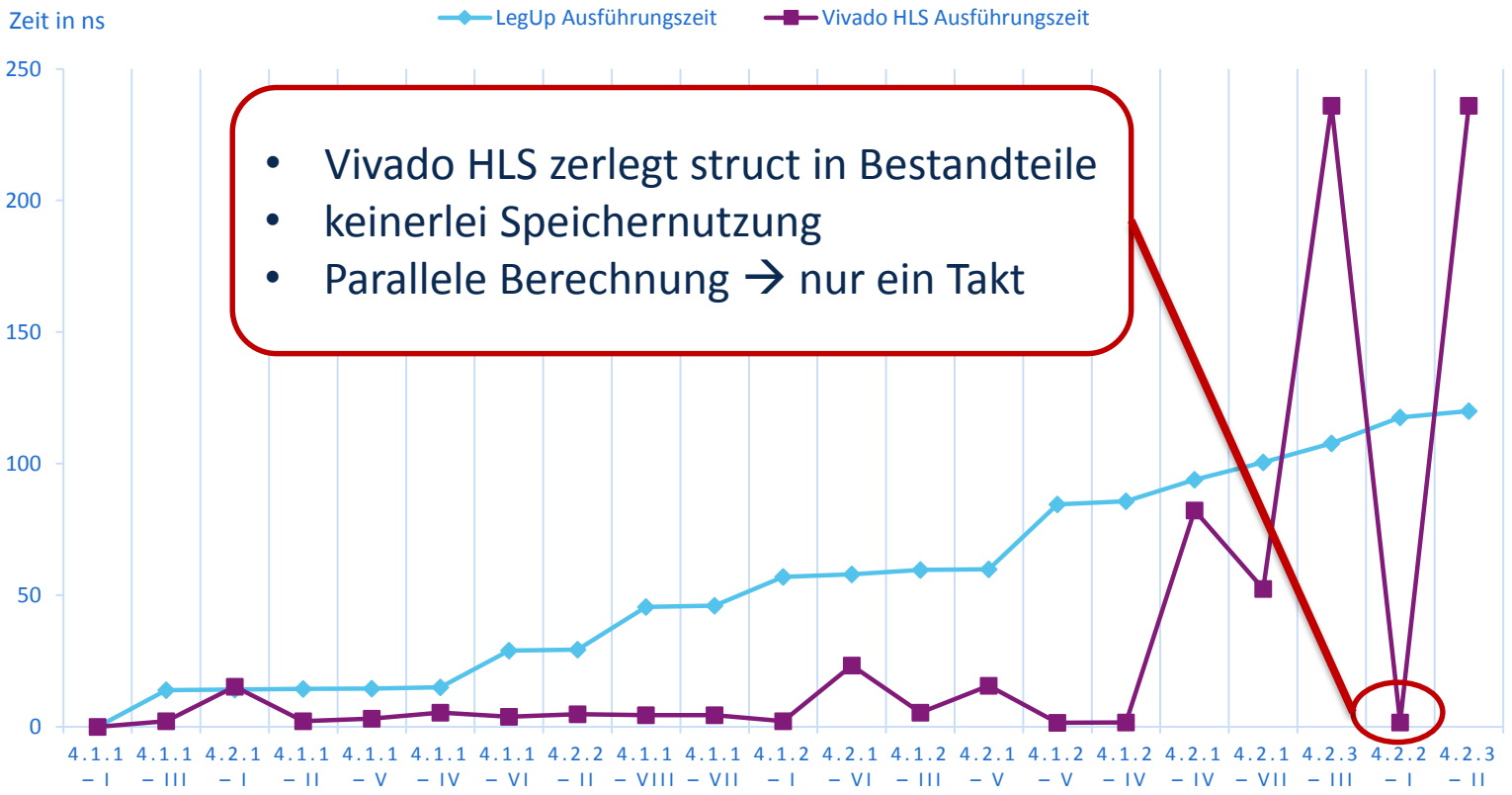
```
#define SIZE 5

int a[SIZE] = {10, 11, 12, 13, 14};
int b[SIZE] = {5, 6, 7, 8, 9};
int c[SIZE] = {0, 1, 2, 3, 4};

int testFunction( int inA ) {
   int result=0;
   result += f1(a, inA) + f1(b, inA+1) + f1(c, inA+2);
   return result;
}

int f1(int array[5],int indx){
   return array[indx]*2;
}
```



4.1 Ausführungszeit



4.1 Ausführungszeit



- Vivado HLS zerlegt struct in Bestandteile
- keinerlei Speichernutzung
- Parallele Berechnung → nur ein Takt

5. ZUSAMMENFASSUNG & AUSBLICK

5. Zusammenfassung

- Handgeschriebener Code effizienter
  - Aber Eingriffe in HLS möglich durch Direktiven oder im System selbst
- Nicht komplette C-Funktionalität unterstützt → Kenntnisse über HW helfen
  - Aber effiziente CPU Programmierung auch mit Architekturkenntnissen (ausnutzen der Caches)
- Compileroptimierungen haben großen Einfluss

5. Zusammenfassung

- LegUp:
  - Uneingeschränktes C durch Soft-Prozessor
  - Speicherkontroller: gut für Pointer, aber Flaschenhals

- Vivado HLS:
  - Keine feste Speicherhierarchie → flexibler
  - Ein Speichermodul je Speicher und wenn möglich vervielfältigt → mehr Parallelität
  - Mittels Direktiven gute Ergebnisse möglich, aber Hardwarekenntnisse nötig

5. Ausblick

- LegUp:
  - eigener MIPS (schneller, effizienter und selbst messend/beschleunigend)
  - Speicherhierarchie feiner untergliedern
- Vivado HLS:
  - besser Strukturen erkennen
  - Pointer auf Arrays ermöglichen
- Heap
  - Aber HW für dyn. Speicher nicht gut geeignet
  - Mehr Abstraktion → HW-Vorteile verschenkt?

Quellen

[1] W. Meeus, et al., "An overview of today's high-level synthesis tools.," Design Automation for Embedded Systems, no. 16. Jg., Nr. 3, pp. 31-51, 2012.

[2] A. Canis, et al., "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems.," ACM Transactions on Embedded Computing Systems (TECS), no. 13. Jg., Nr. 2, p. 24, 2013.

[3] Xilinx, "Vivado Design Suite User Guide, High-level Synthesis," UG902 (v2014.3), 2014.

[4] M. Adler, et al., "LEAP Scratchpads: Automatic Memory and Cache Management for Reconfigurable Logic [Extended Version]," 2010.