Prof. Dr. Gundolf Kiefer
Michael Schäferling, M.Sc.

Hochschule Augsburg
Fakultät für Informatik

Hardware-Systeme – SS 2020

Themen für Kürprojekte

In dem Kür-Projekt arbeiten Sie in einem Team mit 3 Personen. Die folgenden Aufgaben beschreiben mögliche Projekte. Gerne können Sie auch eigene Vorschläge machen.

Grundsätzlich darf jedes Projektthema nur von einem Team ausgewählt werden. Viele Themen sind jedoch nach Absprache aufteilbar. Das ist in Moodle an den maximal vorgegebenen Teilnehmerzahlen erkennbar. Das heißt, Moodle-Gruppen mit mehr als 3 Teilnehmern werden nach Absprache intern noch entsprechend aufgeteilt.

Ein Hinweis zum Schwierigkeitsgrad: Die meisten Themenbeschreibung sind bewusst offen formuliert, sodass Sie die Projekte wahlweise in einer Standardvariante oder durch Bearbeitung von optionalen Aufgaben in einer anspruchsvollen Variante durchführen können (vgl. Baustein "Anspruch" in der Bewertung). Aufgaben, **die mit einem '*' gekennzeichnet** sind, erfordern spezielle Vorkenntnisse oder sind bereits in der Grundform überdurchschnittlich anspruchsvoll.

In jedem Fall sollten Sie die Auswahl und Gestaltung Ihres Projektthemas mit Ihrem Betreuer absprechen.

*Hinweis: Aufgaben, die mit **"[LaborHW]"** gekennzeichnet sind, erfordern Spezialhardware, die im EES-Labor ausgeliehen werden kann. In diesem Semester finden keine Präsenztermine im Labor statt. Das heißt, die Arbeiten müssen zu Hause bearbeitet werden. Themen, bei denen nur Teilaufgaben mit "[LaborHW]" markiert sind, können auch ohne Spezialhardware bearbeitet werden.*

*Bitte überprüfen Sie frühzeitig, welche **Software** Sie für Ihr Projekt benötigen und ob Ihr Rechner die nötigen Voraussetzungen erfüllt. Sämtliche bei den Themen erwähnte Software ist für Sie kostenlos erhältlich und in der Regel in der Veranstaltungs-VM lauffähig. Manchmal bestehen aber erhöhte Anforderungen an Hauptspeicher und Festplatte, die von Ihrem persönlichen Rechner evtl. nicht erfüllt werden. Sprechen Sie uns an, um zu klären, inwieweit Sie evtl. per Remote-Login auf den Labor-Rechnern arbeiten können.*

Themenfeld: VISCY-Erweiterung und Hardware-Entwicklung

1. Optimierung der VISCY-CPU

Beschleunigen Sie Ihr Multiplikationsprogramm, z. B. durch

- Optimierung der Software (schnellerer Algorithmus)
- schnellere Ausführung der CPU-Befehle (z.B. Optimierung am Steuerwerk, Pipelining)
- neue CPU-Befehle
- einen Cache

2. Schnelle Integer-Multiplikation

a) Informieren Sie sich über Hardware-Algorithmen zur Integer-Multiplikation. Welcher Algorithmus ist jeweils ideal, wenn
- die Chipfläche
- die Latenz
- der Durchsatz

optimiert werden soll?

b) Implementieren und validieren Sie einen der Algorithmen in VHDL und erzeugen Sie mit Alliance ein IC-Layout zu Ihrem Entwurf!

c) (Optional) Erweitern Sie Ihre VISCY-CPU um einen Befehl zur Integer-Multiplikation. Vergleichen Sie die Rechenzeit mit der ursprünglichen Version der CPU und der Software-Implementierung der Multiplikation!

Quellen:

- Jean-Pierre Deschamps, Géry Jean Antoine Bioul, Gustavo D. Sutter: "Synthesis of Arithmetic Circuits – FPGA, ASIC and Embedded Systems", Wiley-Interscience, 2005
- Israel Koren: "Computer Arithmetic Algorithms", 2nd Edition, A. K. Peters LTD., Natick, MA, 2002, http://www.ecs.umass.edu/ece/koren/arith/simulator

3. Schnelle Integer-Division

a) Informieren Sie sich über Hardware-Algorithmen zur Integer-Division. Welcher Algorithmus ist jeweils ideal, wenn
- die Chipfläche
- die Latenz
- der Durchsatz

optimiert werden soll?

b) Implementieren und validieren Sie einen der Algorithmen in VHDL und erzeugen Sie mit Alliance ein IC-Layout zu Ihrem Entwurf!

c) (Optional) Erweitern Sie Ihre VISCY-CPU um einen Befehl zur Integer-Division.

Quellen:

- Jean-Pierre Deschamps, Géry Jean Antoine Bioul, Gustavo D. Sutter: "Synthesis of Arithmetic Circuits – FPGA, ASIC and Embedded Systems", Wiley-Interscience, 2005
- Israel Koren: "Computer Arithmetic Algorithms", 2nd Edition, A. K. Peters LTD., Natick, MA, 2002, http://www.ecs.umass.edu/ece/koren/arith/simulator

4. Gleitkomma-Arithmetik

Die Aufgabe befasst sich mit der Darstellung von Gleitkomma-Zahlen nach dem Standard IEEE 754.

a) Erläutern Sie die Darstellung von Gleitkomma-Zahlen nach dem IEEE-754-Standard, insbesondere der Typen *half* (16 Bit), *single* (32 Bit) und *double* (64 Bit).

b) Wie werden mit 32 Bit (*single,* was in C oder Java dem Typ *float* entspricht) die Zahlen 3, 0,75 sowie 0,7 dargestellt? Überprüfen Sie Ihre Berechnung mit einem kleinen C- oder Java-Programm, in dem Sie die Zahlen zum Beispiel einer Variablen vom Typ *float* zuweisen und dann deren Codierung als Hexadezimalzahl ausgeben!

c) Warum lässt sich die Zahl 0,7 nicht genau darstellen?

d) Schreiben Sie eine C-Funktion (oder Java-Methode), die zwei Zahlen im *single*-Format miteinander addiert oder multipliziert, und dabei selber *keine Gleitkomma-Operation* verwendet.

e) (Optional) Beschreiben Sie in VHDL eine Hardware zur Addition von zwei Zahlen im *half*-Format (16 Bit). Gehen Sie dabei so vor, wie im Skript im Abschnitt "Entwurf auf Register-Transfer-Ebene" beschrieben ist. Der Code muss mit Alliance synthetisierbar sein.

f) (Optional) Entwerfen Sie zu Ihrem Gleitkomma-Addierer ein Chip-Layout mit Alliance.

g) (Optional) Beginnen Sie, Ihren VISCY-Prozessor mit einer Gleitkomma-Einheit (FPU) auszustatten und integrieren Sie Ihren Gleitkomma-Addierer in die CPU!

Quellen:

- Wikipedia: IEEE 754, https://de.wikipedia.org/w/index.php?title=IEEE_754&oldid=199813385

5. Quick-Sort

Implementieren Sie für den VISCY-Prozessor den Quick-Sort-Algorithmus! Schreiben Sie auch eine C-Implementierung und führen Sie Zeitmessungen durch, um die Geschwindigkeit des VISCY mit der C-Implementierung auf einem PC zu vergleichen.

Tipp: Schreiben und testen(!) Sie zuerst die C-Version und nehmen Sie sie dann als Vorlage für das VISCY-Programm. Wie realisieren Sie Unterprogrammaufrufe? Wie den notwendigen Stack?

6. High-Level-Synthese

Unter High-Level-Synthese (HLS) versteht man die Generierung von Hardware aus Verhaltensbeschreibungen, z.B. in VHDL oder C. Dieses Projekt beschäftigt sich mit der High-Level-Synthese mithilfe von *Xilinx Vivado HLS*.

a) Besorgen Sie sich von der Xilinx-Webseite das Tutorial "Vivado Design Suite Tutorial: High-Level Synthesis" sowie die dazugehörigen Design-Files.

b) Machen Sie sich mit den HLS-Tools vertraut, indem Sie aus dem o. g. Dokument die Übungen des Kapitels "High-Level Synthesis Introduction" durchspielen.

c) Worin besteht die Eingabe, was ist die Ausgabe von Vivado HLS? Was ist eine "Solution"? Was sind "Directives" und wo können Sie angegeben werden?

d) Nennen und erklären Sie 3-5 wesentliche Merkmale, die High-Level-Synthese von der RTL-Synthese unterscheiden! In welchen Punkten werden z.B. vom HLS-Tool Entscheidungen getroffen, die beim RTL-Entwurf manuell erfolgen?

e) Nennen und erklären Sie mehrere Techniken, um eine Design bezüglich Fläche und/oder Geschwindigkeit zu optimieren. Demonstrieren Sie die jeweiligen Techniken an einem Beispiel!

Optional:

f) Erläutern Sie die Schnittstellen, die Vivado HLS unterstützt. Gehen Sie insbesondere darauf ein, wie sich ein IP-Core in ein System integrieren lässt.

g) Nennen und erklären Sie die Datentypen, die Vivado HLS unterstützt.

Quellen:

- Xilinx: "Vivado Design Suite Tutorial: High-Level Synthesis", UG871, www.xilinx.com
- Xilinx: "Introduction to FPGA Design with Vivado High-Level Synthesis ", UG998, www.xilinx.com

7. FPGA-Design mit Open-Source-Tools

FPGA-Hersteller wie Xilinx, Intel (ehemals Altera) oder auch Lattice bieten für ihre FPGA-Bausteine eigene Design-Tools an. Diese decken die Arbeitsschritte von der Synthese über die Platzierung und Verdrahtung bis zur Generierung einer Binärdatei zum Programmieren des jeweiligen FPGA (sog. Bit-File) und letztlich die Programmierung des FPGA selbst ab. Diese Werkzeuge sind zwar sehr mächtig, haben aber den Nachteil dass sie kommerziell / proprietär sind, d.h. es handelt sich um Closed-Source-Software, deren interne Arbeitsweise nur vom Prinzip her bekannt ist, Details bleiben jedoch das Geheimnis des jeweiligen Herstellers.

Für die „iCE40"-FPGA-Familie des Herstellers Lattice gibt es jedoch mit dem Projekt „IceStorm" eine Open-Source-Toolchain, die alle oben genannten Arbeitsschritte abdeckt. In dieser Aufgabe entwickeln Sie damit Logik für das „iCEblink40HX1K Evaluation Kit" und evaluieren die Leistungsfähigkeit der Open-Source-Toolchain.

a) Besorgen Sie sich zunächst von der Lattice-Webseite die „iCEcube2 Design Software" (Sie müssen sich für die Lizenzierung der Software bei Lattice anmelden) und installieren Sie diese. Hier haben Sie noch die Wahl ob Sie unter Linux oder Windows arbeiten möchten, für die Open-Source-Tools (Teilaufgabe b) ist es aber sehr empfehlenswert Linux zu verwenden. Im TI-Wiki finden Sie Tipps zur Installation unter Linux und auch ein Demo-Design für eine einfache Binär-Counter-Demo, welche Sie nun mit iCEcube2 synthetisieren und auf das Board programmieren (gerne dürfen Sie das Design auch modifizieren). Das von Lattice angebotene „iCEcube2 Tutorial" hilft bei der Benutzung von iCEcube2.

b) Installieren Sie nun die nötigen Open-Source-Tools, eine Hilfestellung dazu finden Sie im TI-Wiki. Führen Sie für die Binär-Counter-Demo mit der Open-Source-Toolchain alle Schritte der Synthese und der Platzierung und Verdrahtung durch, bis Sie eine fertige Programmierdatei (*.bin) erhalten.

   Welches Tool wird für welchen Arbeitsschritt benötigt? Aus welchem Open-Source-Projekt stammt dieses jeweils und wie sind diese entstanden? Welche FPGAs werden unterstützt, wie sind diese im Vergleich zu FPGAs anderer Hersteller einzuordnen?

c) *[LaborHW]* (Optional) Nehmen Sie die Demo auf dem Board in Betrieb.

d) Entwerfen Sie ein eigenes Hardware-Design (das z.B. auch die Buttons verwendet) und generieren daraus jeweils ein Bit-File mit iCEcube2 und der Open-Source-Toolchain. Vergleichen Sie anhand der Log-Ausgaben den Ressourcen-Verbrauch im FPGA und die Timing-Informationen zur Schaltung. Was lässt sich über die Geschwindigkeit des kompletten Ablaufs von der Synthese bis zur Generierung des Bit-Files jeweils feststellen?

e) (optional) Synthetisieren sie Ihr Design für einen (vergleichbaren) FPGA eines anderen Herstellers mit dessen Toolchain. Was lässt sich hier bezüglich Flächenverbrauch, Taktfrequenz und Synthese-Zeit aussagen?

Quellen:

- Lattice: "iCEcube2 Tutorial", v1.2, 26. August 2014, www.latticesemi.com
- Michael Schäferling: *„HowTo zum Lattice-Board iCEblink40-HX1K", http://ti-wiki.informatik.hs-augsburg.de/doku.php?id=howto_lattice_iceblink40-hx1k*

Themenfeld:
Rechnerarchitekturen und konfigurierbare Prozessoren

8. Cache-Architekturen und -Effizienz

Für die Entwicklung effizienter Software ist die Kenntnis der Cache-Architekturen und Cache-freundliches Speicherzugriffsverhalten ein wesentliches Kriterium.

a) Stellen Sie die Funktionsweise und Entwurfsalternativen von Caches vor.

b) Sie sind Hardware-Entwickler und sollen das Cache-System eines neuen Prozessors entwerfen. Wie gehen Sie vor? Worauf ist zu achten? Welche Optimierungsoptionen gibt es?

c) Stellen Sie das Cache-System eines bekannten Prozessors im Detail vor (z.B. aktuelle *ARM Cortex A*-Serie oder *Intel Core i7*).

d) Betrachten Sie das Programm "*matmul.c*" (Material-Seite). Die Funktion *matmul(...)* führt eine Matrix-Multiplikation für quadratische Matrizen beliebiger Größe nach folgendem Schema durch:

$$\begin{pmatrix} a_{0,0} & \dots & a_{0,N-1} \\ \vdots & \ddots & \vdots \\ a_{N-1,0} & \dots & a_{N-1,N-1} \end{pmatrix} \cdot \begin{pmatrix} b_{0,0} & \dots & b_{0,N-1} \\ \vdots & \ddots & \vdots \\ b_{N-1,0} & \dots & b_{N-1,N-1} \end{pmatrix} = \begin{pmatrix} r_{0,0} & \dots & r_{0,N-1} \\ \vdots & \ddots & \vdots \\ r_{N-1,0} & \dots & r_{N-1,N-1} \end{pmatrix}$$

mit $$r_{i,j} = a_{i,0} \cdot b_{0,j} + a_{i,1} \cdot b_{1,j} + \dots + a_{N-1,0} \cdot b_{N-1,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

Warum benötigt das Programm für $N = 512$ viel länger als für $N = 513$?

e) Lassen Sie das Programm für alle N innerhalb eines sinnvollen Bereiches auf dem in c) vorgestellten Prozessor laufen und stellen Sie die Laufzeiten grafisch dar.

Hinweis: Schalten Sie die Geschwindigkeitsoptimierungen des Compilers ein! (GCC: Option "-O3")

f) Bestimmen Sie durch passende Experimente,
- wie groß der Level-1(2/3)-Cache ist.
- wie groß die Blockgröße ist.
- ob der Cache direkt abgebildet ist bzw. welche Assoziativität vorliegt.

g) Optimieren Sie das Programm bzgl. der Cache-Effizienz. Welche Beschleunigung erreichen Sie?

h) (optional) Verwenden Sie einen Cache-Simulator (z.B. *Dinero*, siehe Literatur), um die Fehlgriffsraten der ursprünglichen und der optimierten Version zu ermitteln.

i) (optional) Was versteht man unter Cache-Kohärenz? Wie arbeitet das MESI-Protokoll? Was ist bei der Parallelisierung auf Thread-Ebene hinsichtlich der Speicher-Zugriffe zu beachten?

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017
- Dinero (mit Download-Link): *https://en.wikipedia.org/wiki/Dinero_(cache_simulator)*
- Code "*matmul.c*", Material-Seite

9. Parallelität auf Befehlsebene

a) Was versteht man unter Parallelität auf Befehlsebene (ILP, Instruction-Level Parallelism)? Nennen Sie Compiler- und Hardware-basierte Techniken, die in diese Kategorie fallen!

b) Stellen Sie eine Technik im Detail vor.

c) Demonstrieren Sie diese Technik anhand eines praktischen Beispiels.

d) Stellen Sie die ILP-Umsetzung eines konkreten Prozessors im Detail vor (z.B. aktuelle *ARM Cortex A*-Serie oder *Intel Core i7*).

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017, Kapitel 3

10. Parallelität auf Datenebene

a) Was versteht man unter Parallelität auf Datenebene (DLP, Data-Level Parallelism)? Nennen Sie Hardware-Strukturen, die in diese Kategorie fallen!

b) Vergleichen Sie die Ansätze! Wie erfolgt jeweils die Nutzung aus Software-Sicht?

c) Stellen Sie eine Technik im Detail vor.

d) Demonstrieren Sie diese Technik anhand eines praktischen Beispiels.

e) Stellen Sie einen konkreten Prozessor im Detail vor.

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017, Kapitel 4

11. Parallelität auf Thread-Ebene

a) Was versteht man unter Parallelität auf Thread-Ebene (TLP, Thread-Level Parallelism)? Welche Speicher-Modelle gibt es dabei?

b) Was versteht man unter Cache-Kohärenz? Beschreiben Sie eine Technik zur Gewährleistung der Cache-Kohärenz (z.B. das MESI-Protokoll) im Detail!

c) Was ist bei der Parallelisierung von Software auf Thread-Ebene hinsichtlich der Speicher-Zugriffe zu beachten? Demonstrieren Sie dies anhand eines praktischen Beispieles!

d) Schreiben Sie eine möglichst effiziente parallele Implementierung der Matrix-Multiplikation (Datei *matmul.c*). Verwenden Sie dazu zum Beispiel *OpenMP*.

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017, Kapitel 5
- Code "matmul.c", Material-Seite

12. *Spectre, Meltdown und Co.* - Wie genau funktioniert spekulatives Rechnen und warum gibt es da Sicherheitslücken?

Diese Aufgabe überschneidet sich thematisch mit den Aufgaben 8 und 9. Bitte stimmen Sie sich deshalb ggfs. mit den Teams ab, die diese Aufgaben gewählt haben.

Seit der Jahreswende 2017/2018 sind die Sicherheitslücken *Spectre* und *Meltdown* in aller Munde. Diese Lücken nutzen Techniken von CPUs aus, die bereits seit vielen Jahren im Einsatz und für die Rechenleistung sehr bedeutsam sind. Ziel dieses Projektes ist es, diese Hardware-Techniken zu erklären und zu erläutern, warum daraus Sicherheitslücken entstehen können.

a) Gehen Sie zunächst von einer einfachen CPU mit Pipeline, aber ohne "Out-of-order Execution" aus. Welche Arten der Pipeline-Konflikte ("Pipeline Hazards") gibt es, und wie kann man ihnen ohne große Leistungseinbußen begegnen?

b) Wozu dient eine Sprungvorhersage ("branch prediction")? Welches technische Problem wird damit gelöst? Stellen Sie verschiedene Vorhersagestrategien vor und bewerten Sie diese.

c) Was versteht man unter dynamischem Scheduling ("dynamic scheduling")? Welches technische Problem kann damit gelöst werden? Stellen Sie eine Technik im Detail vor, die auch spekulative Ausführung erlaubt. Erläutern Sie daran detailliert, wie Spekulation funktioniert und wie zum Beispiel sichergestellt wird, dass keine falschen Ergebnisse entstehen.

d) Erläutern Sie einen Angriff im Detail anhand eines Beispieles, zum Beispiel:

- den Meltdown-Angriff "Rogue Data Cache Load". Welche CPU-Eigenschaften werden dabei ausgenutzt? Erläutern Sie dazu auch die "Page Table Isolation (PTI)"-Technik in Betriebssystemen. Inwieweit kann damit ein Meltdown-Angriff abgewehrt werden?
- den Spectre-Angriff "Bounds Check Bypass". Welche CPU-Eigenschaften werden dabei ausgenutzt? Wie lässt er sich unterbinden?
- den Spectre-Angriff "Branch Target Injection". Welche CPU-Eigenschaften werden dabei ausgenutzt? Wie lässt er sich unterbinden?

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017, insbes. Kap. 3, Anhänge B, C
- Olivia von Westernhagen: "Meltdown & Spectre-Angriffe im Detail", c't 18/03, S. 62ff., Heise Medien, 2018

13. *RISC-V:* Projektübersicht, Architektur und Tools

RISC-V ist eine offene Befehlssatzarchitektur, die unter anderem an der Computer Science Division der University of California Berkeley unter Mitwirkung von David Patterson entwickelt wurde und eine quelloffene Alternative zu Intel-, ARM- und MIPS-Prozessor-Designs bietet. Bemerkenswert ist, dass bereits eine große Anzahl an bedeutenden Firmen das Projekt unterstützen und bereits eine große Menge an Software zur Verfügung steht.

Die Befehlssatzarchitektur ist flexibel gestaltet, sodass eine große Bandbreite an Prozessoren von stromsparenden "Internet-of-Things"-Anwendungen bis zu 128-Bit-Serverprozessoren unterstützt werden.

a) Stellen Sie das RISC-V-Projekt vor. Was sind die Ziele? Welche Dokumente, Software und Hardware sind verfügbar? Wie ist das Projekt organisiert?

b) Stellen Sie kurz die Befehlssatzarchitektur vor. Welches sind die wichtigsten Eigenschaften und Befehlsgruppen? Welche Befehle sind immer vorhanden? Welche Erweiterungen gibt es?

c) Stellen Sie die verfügbaren Software-Werkzeuge vor.

Hinweis: Für die folgenden praktischen Aufgaben existieren gut vorbereitete Anleitungen auf den Projekt-Webseiten.

d) Schreiben Sie ein C-Program, das die ersten 10 Fibonacci-Zahlen ausgibt. Übersetzen Sie das Programm für RISC-V (*GCC* + *newlib*-Umgebung) und lassen Sie es in einem Simulator laufen (*Spike* oder *QEMU*).

Optional:

e) Bauen Sie einen RISC-V-Linux-Kernel und booten Sie ihn im Simulator. Lassen Sie Ihr C-Programm dann unter Linux laufen.

Quellen:

- *http://riscv.org*

14. *RISC-V:* Der Befehlssatz

Stellen Sie den RISC-V-Befehlssatz vor (ohne privilegierte Befehle).

a) Was sind die Grundeigenschaften jedes RISC-V-Prozessors? Welche Varianten gibt es?

b) Stellen Sie den Basis-Integer-Befehlssatz (RV32I) vor. Welche Befehlsgruppen gibt es? Wie sind die Befehle codiert?

c) Geben Sie einen Überblick über die optionalen Befehlssatz-Erweiterungen ("extensions").

d) Stellen Sie 1-2 der Befehlssatzerweiterungen im Detail vor.

e) (Optional) Präsentieren Sie Code-Beispiele zu den vorher beschriebenen Maschinenbefehlen. Lassen Sie sie in einem Simulator laufen!

Quellen:

- *http://riscv.org*
- Andrew Waterman, Krste Asanovic (Editors): "The RISC-V Instruction Set Manual -

Volume I: Unprivileged ISA", *https://riscv.org/specifications/isa-spec-pdf*

15. *RISC-V:* Privilegierte Befehle

Um auf einem Prozessor ein sicheres Betriebssystem mit Speicherschutz zu betreiben, sind privilegierte Operationen sowie Unterstützung für virtuellen Speicher notwendig.

a) Welche Privilegien-Level sind in der RISC-V-Spezifikation vorgesehen? Für welchen typischen Einsatzzweck sind sie vorgesehen?

b) Zu welchem Zweck gibt es spezielle Kontroll- und Statusregister (CSR)? Geben Sie einen Überblick über die Register und deren Bedeutungen.

c) Stellen Sie vor, wie die Address-Übersetzung bei der virtuellen Speicherverwaltung funktioniert.

d) Welche Schutzmechanismen sind für den Speicherschutz insgesamt vorhanden?

Quellen:

- *http://riscv.org*
- Andrew Waterman, Krste Asanovic (Editors): "The RISC-V Instruction Set Manual - Volume II: Privileged Architecture", *https://riscv.org/specifications/privileged-isa*

16. *ParaNut / RISC-V:* Eine freier Parallelprozessor der Hochschule Augsburg

ParaNut/RISC-V ist eine quelloffene und hochgradig konfigurierbare Prozessorarchitektur, die an der Hochschule Augsburg entwickelt wurde und stetig weiterentwickelt wird. Eine Besonderheit der *ParaNut*-Architektur ist ein spezielles Konzept für die Parallelverarbeitung, das die Vorteile von SIMD-Vektorisierung und "Simultaneous Multithreading" (SMT) in einer Architektur vereint.

Durch die RISC-V-Kompatibilität konnten schon größere Software-Projekte (z.B. der Spiele-Klassiker "Doom") auf einem *ParaNut*-System ausgeführt werden.

Theoretischer Teil:

a) Was versteht man jeweils unter Parallelität auf Daten-, Befehls-, und Thread-Ebene? Welche dieser drei Klassen der Parallelisierung werden vom *ParaNut* unterstützt und in welcher Weise?

b) Stellen Sie die Architektur des *ParaNut* vor. Aus welchen Komponenten besteht ein *ParaNut*-Prozessor und welche Funktion haben sie jeweils? Was ist eine *CePU*, was eine *CoPU*?

c) Was ist der "Linked Mode" und wie funktioniert er? Erläutern sie ihn anhand eines Beispiels. Welche Modi gibt es, und was beschreibt der "Capability"-Level eines Kerns?

d) Was ist in dem Quellcode-Paket des Projektes enthalten? In welchen Sprachen ist der Prozessor beschrieben? Welche Tools und Software-Bibliotheken gibt es noch?

e) Welche Funktionalitäten bietet die Bibliothek *libparanut*?

Praktischer Teil:

f) Nehmen Sie den Befehlssatzsimulator (engl. "Instruction Set Simulator, ISS") in Betrieb und

lassen Sie darauf ein einfaches "Hello World"-Programm laufen.

g) (Optional) Implementieren Sie ein einfaches eigenes C-Programm (zum Beispiel eines zur Berechnung der Mandelbrot-Menge). Versuchen Sie, dieses mit Hilfe des "Linked Mode" oder des "Thread Mode" für einen ParaNut mit 4 Kernen zu optimieren. Bestimmen Sie die Laufzeiten Ihrer Varianten mithilfe des Befehlssatzsimulators!

h) *[LaborHW]* (Optional) Bauen Sie ein ParaNut-System für das ZYBO und lassen Sie Ihr "Hello World"-Programm auf dem System laufen.

i) *[LaborHW]* (Optional) Erweitern Sie das ParaNut-System und binden Sie weitere Hardware an, zum Beispiel, um die Portierung des Spieles "Doom" mit einem Joystick spielbar zu machen. Sprechen Sie uns an, um genauere Hinweise zu erhalten.

Quellen:

- Projektseite: *https://ees.hs-augsburg.de/paranut*
- Alexander Bahle, Gundolf Kiefer, Anna Kerstin Pfützner, Lutz Vollbracht: "The ParaNut/RISC-V Processor - An Open, Parallel, and Highly Scalable Processor Architecture for FPGA-based Systems", embedded world Conference 2020, Nürnberg, Download: *https://ees.hs-augsburg.de/paranut*

17. *RISC-V:* Vorstellung einer freien Implementierung

Dieses Projekt kann von mehreren Teams gewählt werden, die dann verschiedene Prozessoren oder Systeme vorstellen.

Für die RISC-V-Architektur existieren zahlreiche freie Prozessor-Implementierungen. In diesem Projekt wählen Sie *eine* Implementierung aus und stellen sie im Detail vor. Vorschläge:

- Rocket-Chip-Generator (*https://github.com/chipsalliance/rocket-chip*)
- SweRV EH1 (Western Digital) (*https://github.com/chipsalliance/Cores-SweRV*)
- PULP-Plattform (ETH Zürich) (*https://pulp-platform.org*) mit den CPU-Kernen
  - RI5CY (*https://github.com/pulp-platform/riscv*),
  - Ibex (ehemals "Zero-riscy") (*https://github.com/lowRISC/ibex*),
  - Ariane (*https://github.com/pulp-platform/ariane*)
- ParaNut/RISC-V: siehe eigene Aufgabe

a) Stellen Sie das Projekt vor. Welche Lizenz wird verwendet? Für welche Zielanwendungen ist das Design optimiert? Welche Konfigurationsoptionen gibt es? Welche Hardwarekomponenten sind in dem Paket enthalten? Welche Softwarekomponenten (z.B. Bibliotheken), welche -Tools? Welche Sprachen wurden verwendet?

b) Stellen Sie die Prozessorstruktur im Detail vor.

c) Stellen Sie (soweit vorhanden) die Software-Bibliotheken und -Tools vor.

d) *[LaborHW]* (Optional) Nehmen Sie ein Beispielsystem auf einem im Labor verfügbaren FPGA-

Board (z.B. ZedBoard, ZYBO) in Betrieb. Schreiben Sie dazu ein kurzes C-Programm, das die ersten 10 Fibonacci-Zahlen ausgibt und lassen Sie es auf dem System laufen.

Hinweis: Für einige Projekte (z.B. RocketChip) existieren dazu sehr gut vorbereitete Anleitungen auf den Projekt-Webseiten.

Quellen:

- *http://riscv.org/risc-v-cores*

18. *RISC-V:* Eigene Implementierung *

RISC-V ist eine offene Befehlssatzarchitektur, die eine quelloffene Alternative zu Intel-, ARM- und MIPS-Prozessor-Designs bietet. Die Befehlssatzarchitektur ist flexibel gestaltet, sodass eine große Bandbreite an Prozessoren von stromsparenden "Internet-of-Things"-Anwendungen bis zu 128-Bit-Serverprozessoren unterstützt werden.

a) Entwickeln Sie auf der Grundlage Ihres VISCY-Prozessors einen Prozessor, der eine *kleine Teilmenge* des RISC-V-Befehlssatzes realisiert. Diese Teilmenge muss nur so gewählt sein, dass Sie ein selber in Assembler geschriebenes Programm zur Berechnung der ersten 10 Fibonacci-Zahlen damit ausführen können.

Quellen:

- *http://riscv.org*

Themenfeld:
Heterogene Systeme und Hardwarebeschleunigung

19. System-On-Chip-Entwurf mit dem *ZYBO* *[LaborHW]*

Die Chips der Zynq-Serie von Xilinx enthalten sowohl FPGA-Logik für eigene Hardware als auch performante ARM-Cortex-A9-Kerne. In diesem Projekt untersuchen und demonstrieren Sie die Eigenschaften eines Zynq-Chips mithilfe des ZYBO.

a) Besorgen und lesen Sie die Handbücher des ZYBO sowie des darauf verbauten Zynq-Chips. Stellen Sie beides im Abschluss-Kolloquium vor!

b) Entwerfen Sie mit Vivado ein einfaches System mit ARM-CPU (PS) und FPGA-Logik (PL) und lassen Sie darauf ein "Hello World"-Programm laufen. Hilfestellung hierzu finden Sie in den Versuchsbeschreibungen zur Veranstaltung "Entwurf digitaler Systeme 2" (siehe unten).

c) Entwerfen Sie in VHDL einen eigenen IP-Core (Vivado: "Tools" -> "Create and Package IP..." -> "Create a new AXI4 peripheral") und integrieren Sie ihn in das System!

   Vorschläge für IP-Cores:

   - Dimmbare LEDs: Die 4 LEDs des Boards sind mit digitalen FPGA-Ausgängen verbunden. Durch zyklisches Ein- und Ausschalten (Pulsweiten-Modulation) lassen sie sich dimmen. Implementieren Sie ein Hardware-Modul, das über E/A-Register Helligkeitswerte entgegennimmt und passend dazu PW-modulierte Signale erzeugt.
   - Coprozessor: Implementieren Sie eine Hardware für eine mathematische Funktion, z.B. $f(x) = e^x \bmod n$ für kryptografische Anwendungen.
   - Timer: Implementieren Sie eine hochpräzise Uhr, die sich per Software auslesen lässt.

Quellen:

- *https://reference.digilentinc.com/reference/programmable-logic/zybo/start*
- Versuchsbeschreibung zur Veranstaltung "Entwurf digitaler Systeme 2", *http://www.hs-augsburg.de/~kiefer/eds_ii*, insbesondere:
  - Gesamt-Foliensatz: Kapitel 4.5 (Systementwurf mit der Xilinx Zynq-Plattform)
  - Praktikum: Praktikum 3, Teil 1, Aufgaben "Erstellen eines minimalen Systems mit Vivado" sowie "Erstellen einer 'Hallo Welt'-Software und Inbetriebnahme"

20. Embedded Linux auf der *Zynq*-Plattform *[LaborHW]* *

Hinweis: Das Bauen eines Embedded-Linux-Systems benötigt viel Festplatten-Speicher. Zudem müssen mehrere Tools und die Xilinx-Entwicklungsumgebung installiert werden. Stellen Sie deshalb sicher, dass Sie mindestens 60GB freien Speicherplatz zur Verfügung haben!

a) Die *Zynq*-Plattform

   Machen Sie sich mit der *Zynq*-Plattform vertraut:
   - Wie ist die Architektur der *Zynq*-Plattform beschaffen?

- Welche Komponenten enthält sie, wie kann sie erweitert werden?
- Welche Möglichkeiten gibt es, die *Zynq*-Plattform zu betreiben?

Hinweis: Falls Aufgabe 19 von einer anderen Gruppe bearbeitet wird, so sprechen Sie sich bitte ab.

b) Erste Schritte mit Embedded-Linux auf dem ZYBO

Booten Sie auf dem Board ein fertiges Linux, ohne vorher das FPGA programmiert zu haben (verwenden Sie hier das Referenz-System „*Zybo Base System Design*“). Inspizieren Sie von der Linux-Konsole aus das System: Was können Sie über das gestartete Linux herausfinden? Welche Eigenschaften hat der Prozessor? Wie viel Speicher und wie viel Cache stehen zu Verfügung? Welche Peripherie-Komponenten können wie angesprochen werden?

c) Erstellen eines neuen Embedded-Linux

Erstellen Sie entsprechend der Anleitung in den Quellen ein Linux-Image (statt dem „MicroZed“ verwenden Sie hier das „ZYBO“, passen Sie also die entsprechenden Arbeitsschritte an, aus „zed“/„micozed“ wird „zybo“). Welche Arbeitsschritte werden dabei vollzogen? Aus welchen Teilen besteht das „Endprodukt“?

d) Inbetriebnahme des Systems

Nehmen Sie das neu gebaute System entsprechend der Anleitung in Betrieb. Untersuchen Sie den Boot-Vorgang: in welche Stufen lässt sich dieser generell einteilen? Lassen Sie 'xosview' auf dem ZYBO laufen, so dass man es auf dem Entwicklungs-PC bedienen kann.

e) Erstellen Sie ein kleines Demo-Programm welche die MIOs anspricht (z.B. Taster lesen und LED schreiben).

f) Entwerfen Sie Hardware in VHDL, welche eigenständig die Taster und LEDs des Boards anspricht, z.B. einen 4-Bit-Zähler (eine Steuerung von Linux aus wird nicht verlangt). Die Hardware kann in den PL-Teil des Zynq-Bausteins von Linux aus über das Gerät „/dev/xdevcfg“ programmiert werden.

Optional *:

g) Erweitern Sie das System um ein GPIO-Modul, welches die Board-LEDs von Linux aus ansteuert. Passen Sie das Embedded Linux an den passenden Stellen (Stichwort „Device Tree“) an, so dass Sie von Linux aus die LEDs mit einem eigenen Programm steuern können.

Quellen:

- *https://reference.digilentinc.com/reference/programmable-logic/zybo/start*
- Korbinian Schalkhammer: Anleitung *„Linux auf der Zynq-Plattform“, http://ti-wiki.informatik.hs-augsburg.de/doku.php?id=linux_microzed*
- Xilinx: Wiki-Seiten zu „Xilinx Linux“, *http://www.wiki.xilinx.com/Linux*

21. Heterogenes Rechnen mit der *Zynq UltraScale+*-Plattform *[LaborHW]*

Die aktuellen Chips der *Zynq UltraScale+*-Serie von Xilinx enthalten neben FPGA-Logik auch verschiedenartige ARM-Kerne, eine GPU sowie Spezialhardwarekomponenten. Sie stellen damit eine heterogene Rechnerplattform dar, die für jede Aufgabe eine passende Hardware bereitstellen kann. In dieser Aufgabe befassen Sie sich mit der Architektur.

a) Stellen Sie die *Zynq-UltraScale+*-Architektur im Detail vor. Worin unterscheiden sich die ARM-Kerne der Cortex-A- sowie Cortex-R-Serie? Welche Eigenschaften hat die GPU? Welche übrigen Module enthält der Chip des *UltraZed-EG*?

b) Stellen Sie die Architektur der ARM-Cortex-A53-Kerne im Detail vor.

c) Stellen Sie die Architektur der ARM-Cortex-R5-Kerne im Detail vor.

d) Stellen Sie die Architektur der Mali-400-GPU im Detail vor.

Optional:

e) Nehmen Sie eines der im Labor verfügbaren Boards mit einem geeigneten Demo-Design in Betrieb. Erläutern Sie das Design und erklären Sie, welcher Teil der Demo von welcher der genannten Chip-Komponenten ausgeführt wird.

   Vorschlag für das *UltraZed*-Board:

   - "Out-of-Box-Design" für das ZU3EG mit IOCC (SD-Karten-Image)

Quellen:

- Xilinx: "Zynq UltraScale+ Device - Technical Reference Manual" (UG 1085), *http://www.xilinx.com*
- Xilinx: "Zynq UltraScale+ MPSoC Base Targeted Reference Design" (UG 1221), *http://www.xilinx.com*
- *http://www.ultrazed.org/product/ultrazed-EG*
- Avnet: "UltraZed IO Carrier Card Reference Designs and Tutorials", *http://www.ultrazed.org/support/design/17596/131*

22. Computer Vision mit ASTERICS

Bildverarbeitung stellt insbesondere für eingebettete Systeme eine Herausforderung dar, da diese meist nur beschränkt Rechenleistung zur Verfügung stellen. Deshalb werden hier in der Praxis besonders häufig FPGAs zur Beschleunigung eingesetzt. Das *ASTERICS*-Framework ist ein modularer Baukasten zur Entwicklung FPGA-basierter und Echtzeit-fähiger eingebetteter Bildverarbeitungssysteme.

a) Geben Sie einen Überblick über geläufige Operationen in der Bildverarbeitung:
   - Was sind Punkt-Operatoren ("point operators")? Nennen Sie Beispiele und Anwendungen!
   - Was sind Fenster-Operatoren (oder auch "neighborhood operators")? Nennen Sie Beispiele und Anwendungen!
   - Welche Arten von Merkmalen ("Features") gibt es? Nennen Sie Beispiele!

b) Erläutern Sie Methoden, um in Bildern Kanten zu erkennen.

c) Was ist ein "Sliding Window Buffer"? Wie funktioniert er und welche Vorteile hat er?

d) Machen Sie sich mit dem *ASTERICS*-Framework vertraut. Stellen Sie das Konzept vor und geben Sie einen kurzen Überblick über die vorhandenen Module.

e) *[LaborHW]* (Optional) Nehmen Sie ein einfaches *ASTERICS*-System auf dem ZYBO in Betrieb und erläutern Sie es. Dies kann das Basis-System aus dem Praktikum zur Veranstaltung "Entwurf digitaler Systeme 2" sein oder ein ähnliches System, das Sie bequem mit *ASTERICS Automatics* generieren.

Optional:

f) *[LaborHW]* (Optional) Erweitern Sie Ihr *ASTERICS*-System um eigene Funktionalität, zum Beispiel:
   - Histogramm-Berechung
   - Bewegungserkennung anhand von Differenzbildern

Quellen:

- Richard Szeliski: "Computer Vision: Algorithms and Applications", Springer, 2010, *http://link.springer.com/book/10.1007/978-1-84882-935-0*
- Bailey, D. G.: "Design for embedded image processing on FPGAs", Wiley, Singapore, New York, N.Y, 2011, *http://onlinelibrary.wiley.com/book/10.1002/9780470828519*
- Projekt-Webseite zu ASTERICS: *http://ees.informatik.hs-augsburg.de/asterics*
- ASTERICS-Wiki: *http://asterics-wiki.informatik.hs-augsburg.de*
- Versuchsbeschreibung zur Veranstaltung "Entwurf digitaler Systeme 2", *http://ww.hs-augsburg.de/~kiefer/eds_ii*, insbesondere:
  - Gesamt-Foliensatz: Kapitel 4.5 (Systementwurf mit der Xilinx Zynq-Plattform) und Kapitel 4.6 (Computer Vision und ASTERICS)
  - Praktikum 3 und 4

23. Künstliche Intelligenz mit dem *Intel Neural Compute Stick 2* *[LaborHW]* *

Durch den *Neural Compute Stick 2 (NCS2)* von Intel ist es möglich mit geringer Stromaufnahme neuronale Netze zu simulieren.

a) Informieren Sie sich über den *Myriad-X*-Prozessor, das Kernstück des *Neural Compute Stick 2*. Welche Hauptkomponenten enthält der Chip?

b) Erläutern Sie die *VLIW*-Technik ("*very long instruction word*"). Welche Vor- und Nachteile hat VLIW gegenüber anderen Techniken zur Parallelisierung auf Befehlsebene (ILP)? Wo wird VLIW sonst noch eingesetzt?

c) Was ist ein *Convolutional Neural Network*? Wie ist es aufgebaut und wie funktioniert es? Nennen Sie Anwendungsbeispiele!

d) Nehmen Sie mit dem *Neural Compute Stick 2* ein (bereits vorhandenes) künstliches neuronales Netz in Betrieb!

   Eine Schritt-für-Schritt-Anleitung zur Inbetriebnahme des *NCS2* auf den Laborrechnern finden Sie im TI-Wiki.

Optional:

e) Entwickeln Sie ein eigenes Tool zur Personenerkennung in Echtzeit!

   Die folgenden Modelle dienen zum Erkennen von Personen anhand der "Statur" (nicht klassisch Gesichter). Dabei gibt das Netz einen Featurevektor zurück, der per Cosinusdistanz verglichen werden kann:

   - *http://docs.openvinotoolkit.org/latest/_person_detection_retail_0013_description_person_detection_retail_0013.html*
   - *http://docs.openvinotoolkit.org/latest/_person_reidentification_retail_0079_description_person_reidentification_retail_0079.html*
   - Modell-Download: *https://cloud.hs-augsburg.de/index.php/s/7YR6F6DFr9ingaJ*

   Optional können die Modelle auch mit dem „model downloader“ heruntergeladen werden: *https://github.com/opencv/open_model_zoo/tree/master/model_downloader*

   Nehmen Sie diese beiden Netze in Betrieb und testen Sie sie mit dem folgenden Datensatz:

   - *http://www.robots.ox.ac.uk/~vgg/data/celebrity_together/*

   Zusätzlich kann auch eine Anwendung mit einer Webcam implementiert werden, die "live" Personen erkennt. Bei hoher Motivation kann sogar ein kleines System geschaffen werden, das unbekannte Personen erkennt, ähnliche sammelt und nach einer Zuweisung zu einem Namen fragt (Stichwort: Clustering).

Quellen:

- J. L. Hennessy, D. A. Patterson: "Computer Architecture – A Quantitative Approach", 6th Edition, Morgan Kaufmann Publishers, 2017

- *https://de.wikipedia.org/wiki/Convolutional_Neural_Network*
- *https://software.intel.com/en-us/neural-compute-stick*
- *https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_TensorFlow.html*
- *http://ti-wiki.informatik.hs-augsburg.de*

Themenfeld: Paralleles und verteiltes Rechnen

24. Parallelität auf Thread-Ebene mit *OpenMP*

OpenMP ist eine Spezifikation für Compiler-Direktiven und Bibliotheken zur einfachen Programmierung von parallelen Anwendungen in C/C++ oder Fortran.

a) Geben Sie eine Einführung in die Programmierung mit *OpenMP*. Welche Arten von parallelen Regionen gibt es? Welche Synchronisationsmechanismen gibt es? Welche Besonderheiten gibt es hinsichtlich der Datenhaltung? Illustrieren Sie die wesentlichen Konzepte anhand kleiner, praktischer Beispiele.

b) Entwickeln Sie mithilfe von *OpenMP* eine effiziente parallele Implementierung der Matrix-Multiplikation (Datei "*matmul.c*", siehe auch andere Aufgabe zur Cache-Effizienz) und/oder einer weiteren – von Ihnen frei wählbaren – Applikation. Demonstrieren Sie anhand dieser Beispiele verschiedene Konzepte von *OpenMP*.

Quellen:

- *http://openmp.org/*
- Joel Yliluoma: "Guide into OpenMP: Easy multithreading programming for C++", *http://bisqwit.iki.fi/story/howto/openmp/*

25. Parallelität auf Datenebene mit *OpenCL*

OpenCL (Open Computing Language) ist eine offene Schnittstelle für Rechner, die mit verschiedenartigen, parallelen Recheneinheiten - z .B. CPUs, Grafikprozessoren, DSPs - ausgestattet sein können.

a) Stellen Sie *OpenCL* vor. Was sind die grundlegenden Konzepte, z.B. im Hinblick auf den Datenaustausch (Speicher-Modell) oder Synchronisation?

b) Präsentieren Sie ein praktisches Beispiel (z.B. einen paralleles Sortier-Programm).

Quellen:

- *http://www.khronos.org/opencl/*

26. Verteiltes Rechnen mit *MPI*

Message Passing Interface (MPI) ist ein Standard, der den Nachrichtenaustausch bei parallelen Berechnungen auf verteilten Computersystemen beschreibt.

a) Stellen Sie *MPI* vor. Was sind die grundlegenden Konzepte?

b) Präsentieren Sie ein praktisches Beispiel (z.B. einen paralleles Sortier-Programm).

Quellen:

- *http://de.wikipedia.org/wiki/Message_Passing_Interface*
- Projekt-Webseite: *http://www.mcs.anl.gov/research/projects/mpi*

Themenfeld: Compiler-Hardware-Schnittstelle

27. Die *GNU Compiler Collection (GCC)* *

Der GCC-Compiler wird auf vielen Systemen als Standardcompiler verwenden. Durch seine Frontend/Middleend/Backend-Struktur ist er speziell dafür ausgelegt, schnell an neue Prozessor-Architekturen angepasst zu werden. Das soll hier anhand des *RISC-V*-Backends untersucht werden.

a) Beschreiben Sie den Weg von einem C-Quellcode bis zu einer ausführbaren Binärdatei. Welche Werkzeuge sind involviert? Demonstrieren Sie dies anhand eines praktischen Beispiels auf einem Linux-PC (alternativ: für ein *RISC-V*-System).

b) Welche Aufgaben haben jeweils das Frontend, das Middleend und das Backend des Compilers?

c) Was ist notwendig, um ein neues Backend zu erstellen? Erläutern Sie dies anhand einiger Beispiele aus dem *RISC-V*-Backend (Verzeichnis *gcc/config/riscv*)

Optional:

d) Erstellen Sie ein einfaches Backend für den VISCY-Prozessor (es müssen nicht alle Sprachmerkmale unterstützt werden).

Quellen:

- *http://de.wikipedia.org/wiki/GNU_Compiler_Collection*
- GCC Internals: *http://gcc.gnu.org/onlinedocs/gccint*
- Jeffrey D. Ullman, Monica S. Lam, Ravi Sethi, Alfred V. Aho: "Compiler – Prinzipien, Techniken und Werkzeuge", Pearson Studium, München, 2008
- Quellcode und Anleitung zur RISCV-GCC-Toolkette
  - *https://riscv.org/software-tools*
  - *https://github.com/riscv/riscv-gcc*
- Quellcode und Anleitung zur OpenRISC-GCC-Toolkette:
  - *http://opencores.org/or1k/OpenRISC_GNU_tool_chain*
  - *https://github.com/openrisc/or1k-gcc*

28. Die *Java Virtual Machine (JVM)*

Java-Programme werden bei ihrer Compilierung nicht in realen Maschinencode übersetzt, sondern in Code der "Java Virtual Machine" (JVM). Zur Ausführung muss dieser sog. Bytecode erneut übersetzt oder interpretiert werden.

a) Stellen Sie die "Java Virtual Machine" im Detail vor. Welche grundsätzlichen Eigenschaften hat sie? Welche Befehlsgruppen, welche Befehle gibt es? Erläutern Sie den Befehlssatz im Detail und präsentieren Sie auch Programm-Beispiele zusammen mit dem vom Compiler erzeugten Code.

Hinweis: Mit dem Kommando "*javap -c <Klassenname>*" können Sie eine .class-Datei disassemblieren.

b) Welches sind die Unterschiede zu einem realen Maschinencode? Wo gibt es Ähnlichkeiten?

c) Wie ist eine .class-Datei genau aufgebaut?

Optional:

d) Untersuchen Sie mit "javap -c ...", welche Bytecode-Befehle das unten gelistete Programm "*Blinky.java*" verwendet und implementieren Sie in C/C++ eine (vereinfachte) JVM, die in der Lage ist, dieses Programm auszuführen!

Beim Aufruf der nativen Methode '*setLeds ()*' sollen die unteren 8 Bit des Argumentes '*state*' als Kette von ein- oder ausgeschalteten "LEDs" auf der Konsole ausgegeben werden, z.B.:

Aufruf: `setLeds (14);`
=> Ausgabe von: `.***....`

e) Implementieren Sie für den VISCY-Prozessore eine einfache JVM, die in der Lage ist, das Programm '*Blinky.java'* auszuführen! Beim Aufruf der nativen Methode '*setLeds ()*' sollen die LEDs auf dem FPGA-Board entsprechend dem Argument '*state*' geschaltet werden.

Quellen:

- Orcacle: "Java Virtual Machine Specification", *http://docs.oracle.com/javase/specs/jvms/se7/html/*
- Wikipedia: "Squawk virtual machine", *http://en.wikipedia.org/wiki/Squawk_virtual_machine*

Beispiel-Code (Datei "*Blinky.java*"):

```
public class Blinky {
  static final int delay = 4;

  native static void setLeds (int state);

  public static void main(String[] args) {
    int n, k;

    for (n = 0; n < 8; n++)
      for (k = 0; k < delay; k++) setLeds (1 << n);
    for (n = 7; n >= 0; n--)
      for (k = 0; k < delay; k++) setLeds (1 << n);
  }
}
```

Themenfeld: Betriebssystem-Hardware-Schnittstelle

29. Atomare Operationen und Synchronisation

Zur Implementierung von Synchronisationsprimitiven sind in der Praxis sog. atomare Operationen notwendig.

a) Erläutern Sie anhand von Beispielen die Funktionsweise und Anwendung der folgenden Synchronisationsprimitive.

- Mutex (Spinlock + blockierender Mutex)
- Semaphor
- Monitor
- Bedingungsvariable (condition variable)

Geben Sie zu jedem Mechanismus jeweils ein Code-Beispiel in Java (Java-Laufzeitumgebung) und in C (mit *PThread*-Bibliothek) an.

b) Stellen Sie die folgenden Spezialinstruktionstypen vor. Zeigen Sie, wie damit Spinlocks implementiert werden können und diskutieren Sie die Unterschiede.

- Test-and-set
- Compare-and-swap
- Load-link / Store-conditional

c) Erläutern und dokumentieren Sie, wie im Linux-Kernel die Spinlock-Funktionen für die folgenden Architekturen implementiert sind (Startpunkt im Kernel-Quellcode: *arch/<Architektur>/include/asm/spinlock.h*).

- ARM (arm)
- Intel x86 (x86)

Quellen:

- Jürgen Quade, Eva-Katharina Kunst: "Linux-Treiber entwickeln – Eine systematische Einführung in die Gerätetreiber- und Kernelprogrammierung", 4. Auflage, dpunkt.verlag, Heidelberg, 2016

30. Virtuelle Speicherverwaltung in der Praxis

a) Welches sind die Aufgaben einer virtuellen Speicherverwaltung?

b) Wie funktioniert eine ein- bzw. mehrstufige Seitenverwaltung? Welche Maßnahmen sind im Allgemeinen notwendig, um Sicherheit und Effizienz zu gewährleisten?

c) Stellen Sie die MMUs der folgenden Prozessorarchitekturen vor und vergleichen Sie sie miteinander:

- RISC-V
- OpenRISC
- ARM (32 Bit)
- ARM (64 Bit)
- Intel x86

d) Erläutern Sie das Speichermodell von Linux auf Kernel-Ebene. Wie wird freier physikalischer Speicher verwaltet? Wie kann ein Treiber Speicher an festen Adressen (z.B. den Framebuffer einer Grafikkarte) zugänglich machen?

Quellen:

- Jürgen Quade, Eva-Katharina Kunst: "Linux-Treiber entwickeln – Eine systematische Einführung in die Gerätetreiber- und Kernelprogrammierung", 4. Auflage, dpunkt.verlag, Heidelberg, 2016