Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Anwendungen 1 WiSe 2011

Steffen Rempp

Parallelisierung auf MPSoC Plattformen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Steffen Rempp

Parallelisierung auf MPSoC Plattformen

Ausarbeitung eingereicht im Rahmen der Veranstaltung Anwendungen 1
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Prüfer : Prof. Dr. Kai von Luck
Prüfer : Prof. Dr. Gunther Klemke

Betreuer : Prof. Dr. Bernd Schwarz

Abgegeben am 29. Februar 2012

Inhaltsverzeichnis

1 Einführung

Embedded Systems sind eine Querschnitttechnologie, in der Computersysteme in einen technischen Prozess eines Produktes eingebettet sind und seine Funktion ausführen und steuern. Nach [Bar12] werden Embedded Systems als eine Kombination von Computer Hardware und Software sowie zusätzlichen mechanischen oder digitalen Komponenten definiert, um eine spezielle Anwendung auszuführen, für welche sie entwickelt wurden. In vielen Fällen sind Embedded Systems Teil eines größeren Systems oder Produkts, wie z.B. im Falle eines Antiblockiersystems (ABS) in einem Automobil. Sie erstrecken sich über verschiedene Anwendungsbereiche und Märkte, z.B. in der Automobil- und Flugzeugindustrie, Industrie- und Automatisierungstechnik, Verbraucherelektronik oder in der Medizintechnik. Nach [Sta10] unterscheiden sich Embedded Systems von konventionellen Computersystemen wie Desktop PCs oder Notebooks dadurch, dass sie viel enger an ihre Umgebung gekoppelt sind, mit der sie über Interfaces, Sensorik und Aktorik interagieren. Oft muss das Embedded System einer Reihe von strikten Echtzeitanforderungen und Sicherheitskriterien genügen. Um Performancesteigerungen zu erreichen werden im Embedded Bereich oftmals FPGAs (Field Programmable Gate Arrays) und ASICs (Application Specific Integrated Curcuits) eingesetzt. Die verwendete Software implementiert meist eine fixe und applikationsspezifische Funktion.

Als eine Hardware- und Systemarchitektur haben sich *Systems on Chip* (SoC) im Embedded Bereich schon seit längerer Zeit etabliert [JW05]. Ein SoC ist ein integrierter Schaltkreis (Chip), in dem alle Komponenten und Funktionen eines elektronischen Systems integriert sind. Das System enthält z.B. Speicherblöcke, Prozessoren, Busse oder reprogrammierbare, parallele Arithmetik- und Logik-Ressourcen. Hierbei ist die SoC Architektur ebenfalls auf die jeweilige Applikation zugeschnitten. Ein SoC wird als *Multiprozessor System on Chip* (MPSoC) bezeichnet, wenn es mehrere Instruction-Set Prozessoren (CPUs) beinhaltet. Aufgrund der steigenden Komplexität in den Anwendungen werden SoCs immer mehr durch MPSoCs abgelöst [ZVE10]. Diese Entwicklung geht einher mit der Weiterentwicklung universeller Prozessorarchitekturen, die sich durch eine steigende Zahl von Prozessoren oder Prozessorkernen beobachten lässt, z.B. Intels Core i7 mit vier Prozessorkernen. Auch im Embedded Bereich steigt die Zahl der Prozessoren, wie z.B. der ARM11 MPCore mit bis zu vier Prozessorkernen.

Das Ziel des Forschungsschwerpunkts *High Performance Embedded Computing* (HPEC) im Projekt *FAUST* an der HAW Hamburg ist es, FPGA-basierte Embedded Plattformen zur Steuerung eines autonomen Fahrzeugs zu entwickeln. Hierfür werden u.a. Spartan3 oder Virtex5 FPGAs verwendet. Damit ist sowohl die Integration von Software Lösungen auf einem Softcore MicroBlaze Prozessor als auch die Implementierung von Fahrspurfüh-

rungsalgorithmen in Hardware auf dem FPGA möglich. Die Modellierung von Algorithmen in Hardware eignet sich insbesondere für die Echtzeit-Signalverarbeitung, z.B. in der digitalen Bildverarbeitung zur Fahrspurerkennung und -führung. Die bisher verwendeten SoC Plattformen bestehen aus einer Vielzahl von *Intellectual Properties* (IPs), die über ein PLB-Bussystem mit dem MicroBlaze auf dem FPGA verbunden sind. Weiterhin werden die IPs auf dem FPGA nach ihrer Echtzeitfähigkeit in Realtime IPs und Non-Realtime IPs unterteilt. Die Realtime IPs werden für die Erfassung der Kamera- und Sensordaten verwendet, während die Non-Realtime IPs Debug- Monitoring- und Kontrollfunktionen implementieren. Die Fahrerassistenzsysteme sollen in den nächsten Projektschritten auf FPGA-basierte MPSoC Plattformen portiert werden.

Das Ziel meiner Masterarbeit ist es, eine MPSoC Plattform auf Basis des ARM Dualcore Cortex A9 Prozessors zu entwickeln und das Fahrspurführungssystem des HPEC Projektes darauf zu implementieren. Diese Anwendung ist bereits auf den bisher verwendeten, FPGA-basierten SoC Plattformen implementiert und soll nun auf die neue MPSoC Plattform migriert werden. In Zukunft sollen die Fahrerassistenzsystem auf der *Zynq 7000 Extensible Processing Platform* (Zync EEP) von XILINX entwickelt werden, deren Kernkomponenten ebenfalls ein ARM Cortex A9 DualCore Prozessor auf einem XILINX 28 nm FPGA sind. Da diese Plattform zu diesem Zeitpunkt von XILINX für das Jahr 2012 zwar angekündigt wurde jedoch auf dem Markt noch nicht frei erhältlich ist, wird in meiner Arbeit solange die die *OMAP 4430 Plattform* von Texas Instruments zur Entwicklung eingesetzt, die ebenfalls auf dem ARM Cortex A9 DualCore Prozessor basiert, was den gemeinsamen Nenner zur Zync EEP Plattform darstellt.

Für die Entwicklung auf der MPSoC Plattform muss die bisherige Hardware-Software Partitionierung vor allem aus zwei Gründen verschoben werden. Erstens ist auf dem PandaBoard, dem Entwicklungsboard für die OMAP Plattform kein FPGA integriert, zweitens wird mit dem ARM Cortex A9 DualCore Prozessor neue Rechenleistung hinzugewonnen, die voll ausgenutzt werden soll. Darin begründet sich auch die Motivation für diese Arbeit, da der Anteil an Software, die auf dem Prozessor ausgeführt werden soll, steigen wird. Um den DualCore Prozessor effizient auszulasten, bedarf es zusätzlich einer Parallelisierung auf Softwareebene. Die anlässlich des 4. IT-Gipfels der Bundesregierung 2010 verabschiedete *Nationale Roadmap Embedded Systems* [ZVE10] prognostiziert ebenfalls einen Anstieg der Softwarekomponenten in Embedded Systems, was eine erhöhte Nachfrage nach Embedded Softwareentwicklern nach sich ziehe, sodass mit diesem MPSoC-Projekt auch eine umfangreiche Vorbereitung auf diesen Arbeitsmarkt durchgeführt wird.

Die folgenden Kapitel dieser Ausarbeitung sind folgendermaßen strukturiert. In Kapitel 2 werden verschiedene für die Arbeit relevante Multiprozessor Architekturen beschrieben und verglichen. Kapitel 3 gibt einen Überblick über SMP als Softwaredesign Prinzip für Multiprozessor-Programmierung. Kapitel 4 gibt einen Ausblick auf die zukünftigen Arbeiten auf dem Weg der MPSoC Entwicklung.

2 Multiprozessor Architekturen

Ein MPSoC ist ein Chip, auf dem mehrere programmierbare Instsruction Set Prozessoren (CPUs) enthalten sind. Diese Definition bietet Spielraum für eine Vielzahl an verschiedenen Multiprozessor Architekturen. Letztendlich gestaltet sich die Architektur eines Multiprozessorsystems zum einen an den Merkmalen der einzelnen Prozessor-Subsysteme, wie z.B. das Instruction Set, die Taktfrequenz oder Cache Anordnung und zum anderen an der On-Chip-Infrastruktur, die aussagt, wie die einzelnen Prozessor-Subsysteme miteinander kommunizieren. Hierbei unterscheidet man häufig zwischen nachrichtengekoppelten Multiprozessorsystemen, in denen die einzelnen Prozessorsysteme über Nachrichtenbusse kommunizieren, und speichergekoppelten Multiprozessorsystemen, in denen die Kommunikation über gemeinsam genutzten Speicher erfolgt. Da gerade bei Embedded Systems Plattform und Applikation gegenseitig exakt aufeinander zugeschnitten werden, fließt die Aufgabenverteilung der Funktionen auf die einzelnen Prozessor-Subsysteme ebenfalls in die Architektur als zusätzliches Vergleichskriterium mit ein. Im Folgenden werden nun drei verschiedene Multiprozessorarchitekturen nach den gerade erwähnten Kriterien untersucht und verglichen.

2.1 Heterogene Multiprozessorsysteme

Heterogene MPSoC bestehen aus mehreren verschiedenen programmierbaren Prozessorsystemen mit unterschiedlichen Befehlssätzen. In den meisten Fällen handelt es sich um nachrichtengekoppelte Multiprozessorsysteme, in denen die Kommunikation über Message Passing via eines On-Chip Bussystems stattfindet, über das die einzelnen heterogenen Prozessorsysteme miteinander verbunden sind. Als Beispiel sei hier das AMBA (Advanced Microcontroller Bus Architecture) On-Chip Bussystem von ARM erwähnt. Jedes Prozessor-Subsystem verfügt über seinen eigenen Speicherbereich und benutzt seinen eigenen Adressraum. Es gibt keinen gemeinsam genutzten Speicher oder Adressraum. Jedoch sind externe oder On-Chip Speicherblöcke in einigen Architekturen durchaus gängig.

Jedes Prozessor-Subsystem führt seinen eigenen Software Stack aus, d.h. jedem Prozessor-Subsystem wird sein eigenes Software-Subsystem zugeteilt, was die Komplexität der Softwareentwicklung aufgrund der unterschiedlichen Befehlssätze ansteigen lässt. Diese Heterogenität und vollständige Nebenläufigkeit der Hardware/Software Prozessorsysteme eignet sich vor allem bei funktionaler Parallelität oder *Task-level* Parallelität [JW05] in komponentenbasierten Applikationen, die sich in nebenläufige Funktionen un-

terteilen lassen, wobei eine Komponente jeweils aus einem Prozessor-Subsystem und seiner zugeteilten Funktion in Software besteht.

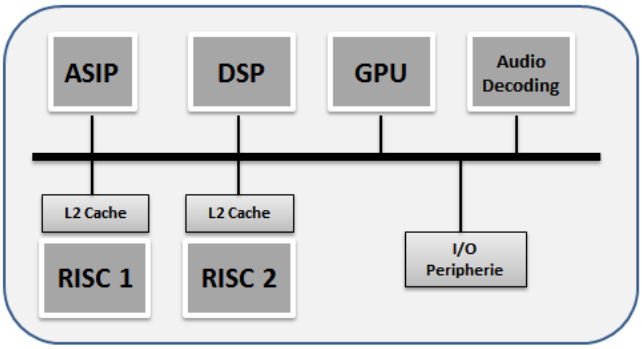

Abbildung 2.1: Modell eines heterogenen MPSoC

Moderne Smartphones weisen heutzutage vier bis acht Prozessorsysteme auf (vgl. Abbildung 2.1). Ein oder mehrere RISC Prozessoren für User Inferface, Netzwerkprotokoll-Stack oder Kontrollfunkitonen, einen DSP (*Digital Signal Processor*) für Video Encoding oder Decoding, einen Audio Prozessor für das Abspielen von Audiosequenzen oder eine GPU (*Graphical Processing Unit*) für Bildverarbeitung. Zusätzlich kann es noch weitere eingebettete Prozessoren geben, z.B. ASIPs für weitere Funktionen geben. Ein weiteres Beispiel für heterogenes Multiprocessing ist der MPEG-2 Encoding Prozess. Dieser beinhaltet u.a. die drei Funktionskomponenten *Motion Estimating, diskrete Cosinus Transformation* und *Huffman Coding*. Diese Prozesse werden in heterogenen MPSoC von drei verschiedenen Prozessor-Subsystemen vollständig nebenläufig ausgeführt [JW05].

Um optimale Performance, z.B. minimalen Energieverbrauch und Ressourceneffizienz auf dem Chip zu erreichen, beginnt während der Analyse- und Designphase der MPSoC Entwicklung die Suche nach dem am besten geeigneten Prozessor-Subsystem für jede einzelne Funktionskomponente einer Applikation, um die strikten Echtzeit- und Performance-Anforderungen im Embedded Bereich erfüllen zu können. Laut [PRJW10] führe der Trend bei heterogenen MPSoC in Zukunft zu einer Abnahme der Prozessor-Subsysteme und zu gleichzeitiger Zunahme der Prozessorkerne innerhalb eines Prozessor-Subsystems. Damit reduziert sich auch die Zahl der einzelnen Software Stacks, die jedoch an Funktionalität zunehmen werden und nach dem MIMD Prinzip (Multiple Instruction, Multiple Data Stream) auf mehreren identischen Kernen eines Prozessor-Subsystems ausgeführt werden.

2.2 Symmentrische Multiprozessoren

Der Begriff SMP (*Symmetric Multiprocessing*) wird zum einen aus Sicht der Hardware als eine Multiprozessor-Architektur und zum anderen aus Sicht der Software als Fähigkeit eines Betriebssystems bezeichnet, mehrere Threads auf mehrere Prozessoren oder Prozessorkerne symmetrisch zu verteilen. In diesem Kapitel wird zunächst nur auf SMP als Multiprozessor-Architektur eingegangen. Multicore Prozessoren, deren Ursprung die

General-Purpose Prozessorsysteme in Desktop-PCs, Notebooks und Großrechneranlagen sind, werden zunehmend im Embedded Bereich eingesetzt, was die Entwicklung von Embedded Microprozessoren widerspiegelt, wie z.B. ARM11 MPCore oder ARM Cortex A8/A9 Multicore Prozessoren.
Nach [Sta10] werden SMP Systeme als eigenständige Computersysteme mit folgenden Eigenschaften definiert:

1. Alle Prozessoren beinhalten zwei oder mehr identische Prozessoren oder Prozessorkerne.
2. Alle Prozessoren teilen sich denselben Speicherbereich und denselben Adressraum und werden deshalb auch als *Shared Memory* Architekturen bezeichnet.
3. Alle Prozessoren kommunizieren über den gemeinsamen Speicher über einen gemeinsam genutzten Speicherbus, wobei die Speicherzugriffszeit über den Speicherbus von jedem Prozessor aus identisch ist. Dieses Prinzip wird auch als *Uniform Memory Access* (UMA) bezeichnet.
4. Alle Prozessoren teilen sich den Zugriff auf gemeinsam genutzte I/O Geräte, Schnittstellen und Peripherie und können dieselben Funktionen ausführen, daher der Begriff *symmetrisch*.
5. Alle Prozessoren werden von einem SMP-fähigen Betriebssystem kontrolliert, welches Interaktion zwischen Prozessoren und ihren ausgeführten Prozessen und Datenstrukturen gewährleistet.

Die Skalierung der Performance-Steigerung zur steigenden Zahl von Prozessoren ist nicht linear proportional, da ebenso der Synchronisationsaufwand steigt. Um Performance-Einbußen durch zu viel Synchronisationsaufwand zu vermeiden legt z.B. der Linux Kernel 2.6.38 die Obergrenze von acht Prozessoren fest.
Neben der Zahl der Prozessoren sind die Anzahl der Cache Levels sowie deren Anordnung und Zugriffsart ein weiteres Vergleichskriterium von SMPs. Sobald in Shared Memory Architekturen ein Speicherbereich in den Cache eines Prozessors kopiert wird, ergibt sich somit das Problem der Cache Kohärenz. In vielen Architekturen sind zusätzliche Hardware Komponenten integriert um Cache Kohärenz zu gewährleisten, wie z.B. die *Snoop Control Unit* (SCU) im ARM Cortex A9 MPCore, welche das MESI Protokoll implementiert, um die Konsistenz der Daten in den einzelnen Caches zu erhalten.

Abbildung 2.2a zeigt eine Multicore Architektur, in der jeder Prozessor neben seinem eigenen Level 1 Daten- und Instruktion-Cache einen eigenen Level 2 Cache besitzt. Über die einzelnen Caches und einen gemeinsam genutzten Speicherbus hat jeder Prozessor gleichartigen Zugang zum gemeinsam genutzten Hauptspeicher und zur Peripherie. Ein Beispiel für diese Organisation mit keinem gemeinsam genutzten (*Shared*) Cache ist der AMD Opteron Multicore Prozessor. Abbildung 2.2b zeigt eine Multicore Organisation mit einem Shared Level 2 Cache. Beispiele hierfür sind der ARM Cortex A9 MPCore oder der Intel Core Duo Prozessor. Ein Beispiel für verteilten Level 2 Cache und zusätzlichem Shared Level 3 Cache (vgl. Abbildung 2.2c) ist der Intel Core i7 Prozessor.

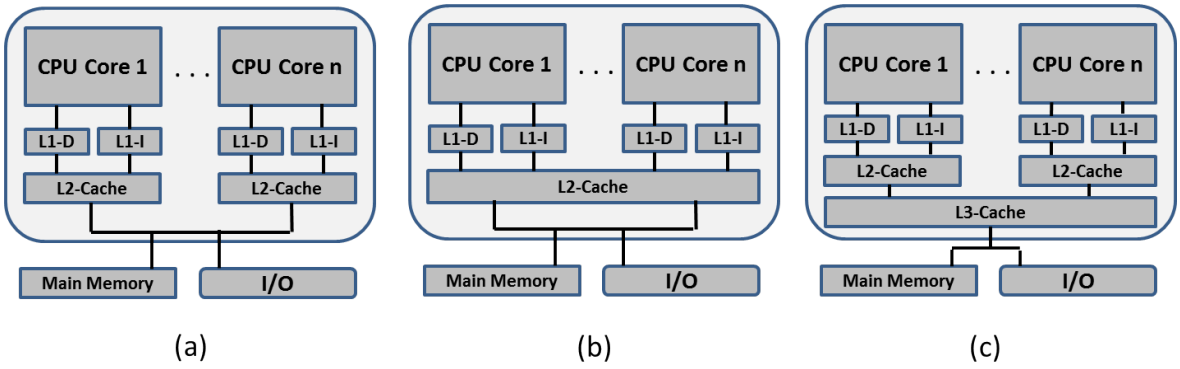

Abbildung 2.2: SMP Multicore Organisation

Die Nutzung von Shared Level 2 Cache beispielsweise hat mehrere Vorteile gegenüber getrennten Caches. Wenn ein Thread auf einem Prozessorkern einen Datenbereich aus dem Hauptspeicher in den Shared Cache holt und ein zweiter Thread auf einem anderen Kern denselben Bereich adressiert, ist dieser bereits auf dem Shared Cache vorhanden, was die Zugriffszeit reduziert. Außerdem müssen gemeinsam genutzte Daten in einem Shared Cache nicht repliziert werden. Die Latenzzeit bei Interprozess-Kommunikation über Shared Memory auf einem Shared Cache wird ebenfalls reduziert. Des Weiteren verschiebt ein Shared Level 2 Cache das Problem der Cache Kohärenz auf die getrennten Level 1 Caches. Der potentielle Vorteil bei getrennten Level 2 Caches ist, dass Prozessorkerne noch schnellere Zugriffszeiten auf ihre privaten Level 2 Caches haben, was sich vor allem bei funktionaler Parallelität von Threads mit wenig Datenabhängigkeit und Synchronisation zwischen Prozessen bemerkbar macht.

2.3 Non Uniform Memory Access

Wie SMP Systeme werden NUMA Architekturen (*Non-Uniform Memory Access*) als speichergekoppelte Multiprozessor Architekturen mit einem gemeinsamen Adressraum bezeichnet. Der Unterschied zu SMP liegt darin, dass der Adressraum physikalisch auf die lokalen Speicher der einzelnen Prozessorsysteme verteilt ist. Jedem Prozessor ist also ein Teil des gesamten Hauptspeichers zugeordnet (vgl. Abbildung 2.3), wobei die einzelnen Prozessor-Speicher-Paare systemweit verteilt sind. Ein physikalisch verteilter Speicherbereich wird als *Distributed Shared Memory* (DSM) bezeichnet. Spricht man diesem Zusammenhang von einem physikalisch verteilten aber logisch zusammenhängenden Adressraum, wird dies auch als *Virtual Shared Memory* (VSM) [Mär01] bezeichnet.

Ein Vorteil gegenüber SMP ist, dass sich der Traffic durch nichtlokale Speicherzugriffe über die gesamte verteilte Topologie verteilt, und es keinen Flaschenhals gibt, wie etwa der gemeinsam genutzte Speicherbus bei Shared Memory SMP. Ein Nachteil gegenüber SMP ist, dass die Zugriffszeiten variieren. Je weiter entfernt der adressierte Speicherbereich vom anfordernden Prozessor ist, desto länger ist die Zugriffszeit, im Gegensatz zu SMP, wo es keine zeitlichen Unterschiede im Speicherzugriff gibt (UMA).

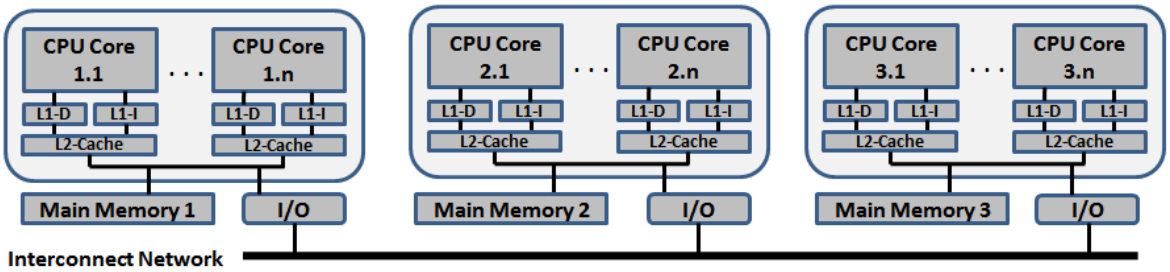

Abbildung 2.3: NUMA Multiprozessor Organisation

2.4 Software Architektur

In MPSoC besteht ein Prozessor Subsystem aus mindestens einem Prozessor inklusive dem Software Stack, welcher darauf ausführt wird, sowie allen Hardware Komponenten, die für die Funktionsausführung benötigt werden, wie z.B. Speicher, spezielle Peripheriekomponenten oder Interfaces. Abbildung 2.4 zeigt ein Modell der Hardware-Software Architektur auf einem Prozessor Subsystem nach [PRJW10]. In heterogenen MPSoC ist jede Hardware-Software Architektur verschieden, während es in reinen SMP Systemen eine Software Architektur gibt, die auf einem Prozessor Subsystem mit mehreren identischen Prozessoren ausgeführt wird. Der Software Stack ist in zwei Ebenen strukturiert.

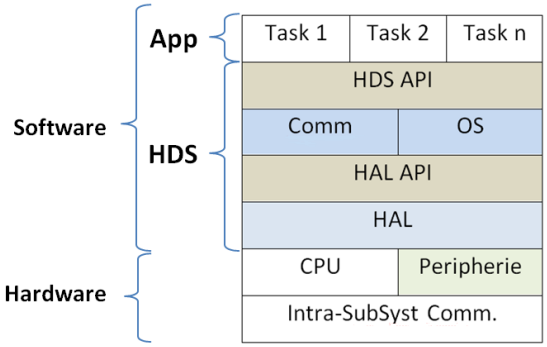

Abbildung 2.4: Hardware-Software Architektur

Die Applikationsebene und die HdS Ebene (Hardware-dependent Software). Die Applikationsebene repräsentiert das High-Level Verhalten der Hardware-Software Komponente, bestehend aus allen Funktionen, aus welchen die ausgeführte Applikation zusammengesetzt ist. Die HdS-Ebene repräsentiert das Hardware-abhängige Low-Level Verhalten, wie z.B. Interrupt Service Routinen, I/O Treiber oder Task-Scheduling. Die HdS-Ebene wird in drei verschiedene Subebenen aufgeteilt: Betriebssystem (*OS*), applikationsspezifische I/O Kommunikation (*Comm*) und die *Hardware Abstraction Layer* (HAL). Nach [YYJ+04] beansprucht der zeitliche Aufwand von HdS Debugging 80 % des zeitlichen Gesamtaufwands für Debugging in der Software Designphase. Grund hierfür seien neben der Komplexität und Fehleranfälligkeit vor allem, dass diese Ebene mehr dafür verantwortlich ist, strikte Systemanforderungen wie z.B. Echtzeit einzualten, als die Applikationsebene.

3 SMP als Multiprozessor Programmiermodell

Eine SMP Architektur wie die OMAP Plattform mit einem ARM Cortex A9 Dualcore Prozessor ermöglicht es der Software, mehrere Threads parallel auf den beiden Prozessorkernen auszuführen. Um den Dualcore Prozessor effektiv und symmetrisch auslasten zu können, bedarf es eines Multithreading Programmiermodells, das dem Softwareentwickler zusätzlichen Overhead durch Multithreading abnimmt, wie z.B. Erzeugen von Threads, Inter-Prozess Kommunikation (IPC) oder Synchronisation sowie Kontrolle von Shared Memory.

3.1 Symmetric Multiprocessing im Linux Kernel

Linux gehört laut [Yag03] mittlerweile zu den am meisten verbreiteten Standards im Embedded Bereich und ist durch seine SMP-Fähigkeit ebenfalls geeignet für MPSoC. Im Linux Kernel ist SMP eine Erweiterung um das symmetrische Verteilen von Prozessen auf mehrere identische CPUs. Diese Erweiterung ist in das ursprüngliche Scheduling integriert.

3.1.1 Kernel Komponenten

Die am meisten verwendete Datenstruktur im Linux Kernel ist der *Prozess* , z.B. ein Thread mit eigenem Adressraum. Dieser wird im Kernel durch eine Struktur vom Typ `task_struct`[1] repräsentiert und auch als *Process Deskriptor* [Bov06] bezeichnet. Er fasst mit seinen Attributen und Feldern alle Informationen des Prozesses zusammen, die für dessen Verwaltung im Kernel relevant sind, z.B. das Feld `pid` für die Prozess ID oder das Feld `state` für den aktuellen Zustand des Prozesses. Jedem Prozess in der Applikationsebene wird sein Prozess Deskriptor zugeordnet.

Linux weist jeder CPU im SMP System ihre eigene *Runqueue* zu, ein Satz von insgesamt 140 doppelt verketteten Listen mit Prozess Deskriptoren, wobei die einzelnen Runqueue-Listen nach ihrer Priorität strukturiert sind. Bei einem Context Wechsel entzieht der Scheduler dem aktuellen Prozess die CPU und weist sie gemäß der konfigurierten

[1] `task_struct`, deklariert in /include/linux/sched.h, Zeile 1193, Identifier Search: http://lxr.free-electrons.com/source/include/linux/sched.h?v=2.6.39;a=arm#L1193

Scheduling Policy dem nächsten Prozess in der Runqueue-Liste zu. Eine Runqueue wird im Linux Kernel durch die Datenstruktur `rq`[2] repräsentiert.

3.1.2 SMP Scheduling

Abbildung 3.1 zeigt die Sequenz an Funktionen, welche die integrierte SMP Erweiterung innerhalb des Scheduling Prozesses im Kernel implementieren. Der Übersicht halber werden hier nur die wichtigsten Funktionen auf oberster funktionaler Ebene beschrieben. Diese bedienen sich im Linux Kernel einer zu großen Vielzahl von Hilfsfunktionen und Makros, um sie übersichtlich darstellen zu können. Die Funktion `scheduler_tick()` wird bei

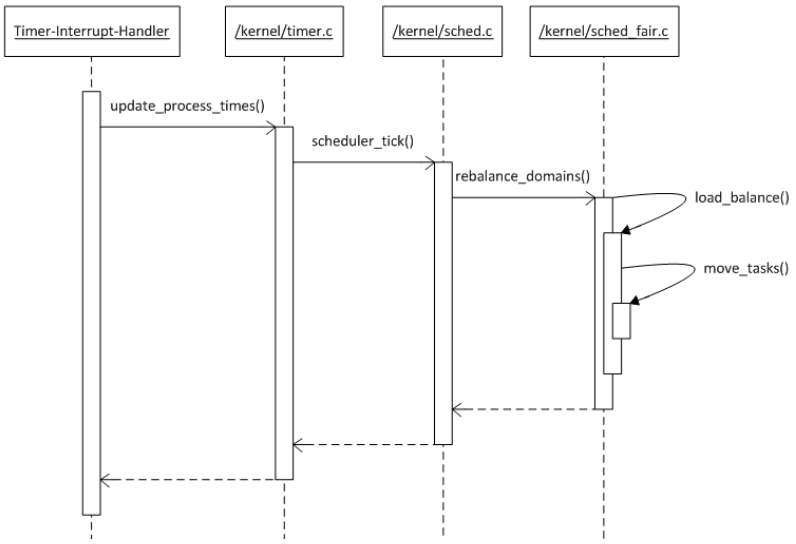

Abbildung 3.1: Sequenzdiagramm: Linux SMP Scheduling

jedem Timer Interrupt aufgerufen. Sie überprüft, ob ein Re-Scheduling nötig ist, z.B. durch Überprüfung von Zeitlimits und Aktualisieren aller zeitabhängigen Felder des aktuellen Prozess Deskriptors. Am Ende wird die Funktion `rebalance_domains()` aufgerufen. Diese bestimmt die Anzahl der Prozese in den Runqueues aller CPUs und aktualisiert deren durchschnittliche Arbeitsauslastung. Danach überprüft sie über alle Runqueues derselben Priorität ob deren Arbeitsauslastungen ausgeglichen sind. Falls nein, wird die Funktion `load_balance()` aufgerufen. Diese Funktion sucht die am stärksten ausgelastete Runqueue und ermittelt die Anzahl Prozesse, die in die Runqueue einer anderen CPU mit derselben Priorität verschoben werden soll, um eine ausgeglichene Arbeitsauslastung aller Runqueues zu erreichen. Die Migration von Prozessen zwischen Runqueues wird von der Funktion `move_tasks()` durchgeführt und durch Setzen der Felder `this_cpu` und `this_rq` in den Prozess Deskriptoren realisiert. Danach wird ein Re-Scheduling veranlasst.

[2] `rq`, deklariert in /include/linux/sched.c, Zeile 447, Identifier Search: http://lxr.free-electrons.com/source/kernel/sched.c?v=2.6.39;a=arm#L447

4 Ausblick

Der Fahrspurführungsassistent des HPEC Projektes soll zunächst als Applikation für die MPSoC Entwicklung auf der OMAP Plattform implementiert werden, solange bis die Zync 7000 Plattform verfügbar ist. Durch den FPGA Block bietet die Zync Plattform mehr Freiheitsgrade als die OMAP Plattform, was das Definieren von Hardware-Funktionsblöcken oder Prozessor-Subsystemen betrifft. Die OMAP Plattform ist ein klassisches Beispiel von einem heterogenen MPSoC, da sie neben dem ARM Cortex A9 DualCore weitere Prozessor-Subsysteme, wie z.B. einen DSP, einen Image Signal Prozessor (ISP), einen Cortex M3 Mikroprozessor sowie einen POWERVR 3D Grafikprozessor und einen IVA3 Hardwarebescleuniger enthält. Wie diese Hardware-Komponenten im weiteren Verlauf dieser Arbeit für die Entwicklung eine Rolle spielen, muss noch erörtert werden. Der Fahrspurassistent enthält eine Reihe von HW-Funktionsblöcken auf dem FPGA, die zusammen den Bildverarbeitungskanal zur Berechnung von Stell- und Kontrollgrößen bilden. Die Eingangsdaten werden zum einen von einem Kamera-Interface und zum anderen von einem Laserscanner geliefert.

Um Echtzeitanforderungen zu erfüllen, könnten einige Prozessor-Subsysteme auf der OMAP Plattform von Nutzen sein, jedoch besteht die Hauptarbeit darin, den ARM Cortex A9 DualCore Prozessor voll auszulasten und den überwiegenden Teil des Fahrspurassistenten in Software zu implementieren. Da es sich beim Cortex A9 um eine Shared Memory Architektur mit Shared Level 2 Cache und einer Snoop Control Unit für Cache Cohärenz handelt, soll in jedem Fall ein SMP-fähiges Betriebssystem darauf installiert werden. Da QNX echtzeitfähig ist und ebenfalls SMP unterstützt, könnte es für die Entwicklung verwendet werden, da aber QNX proprietär ist und der Quellcode damit nicht offen ist, wird am Anfang eine GNU Linux Kernel Version verwendet, um das in Kapitel 3.1 beschriebene SMP-Verhalten sichtbar zu machen und Aufschluss darüber zu bekommen, welcher Thread auf welcher CPU ausgeführt wird.
Als nächster Schritt wird eine Entwicklungsumgebung für die OMAP Plattform auf dem PandaBoard errichtet. Hierzu wird das ARM Developement Studio 5 verwendet, das Analyse- und Debugfunktionen bietet, um die Software auf allen HdS Ebenen, wie z.B. Kernelfunktionen oder Interrupt Handler Funktionen bis hinunter auf Ebene der ARM Prozessorregister zu debuggen. Der Linux Kernel wird durch einen von OMAP spezifizierten Bootvorgang von einer SD-Karte aus gebootet. Eine weitere Herausforderung im nächsten Schritt wird die Beherrschung der Erfassung von zwei Bilddatenströmen durch das Kamera Interface und den Laserscanner sein, welche über HW-Interrupts in Echtzeit an die Software weitergereicht und verarbeitet werden sollen.

Abbildungsverzeichnis

Literaturverzeichnis

[Bar12] BARR, Michael: *Embedded Systems Glossary*. http://www.netrino.com/Embedded-Systems/Glossary/index.php. Version: 2012

[Bov06] BOVET, D.: *Understanding the Linux Kernel*. 3. Edition. O'Reilly, 2006

[JW05] JERRAYA, A. ; WOLF, M.: *Multiprocessor Systems-on-Chip*. Morgan Kaufmann Publishers, 2005

[Mär01] MÄRTIN, C: *Rechnerarchitekturen - CPUs, Systeme, Software-Schnittstellen*. Fachbuchverlag Leipzig, 2001

[PRJW10] POPOVICI, K. ; ROUSSEAU, F. ; JERRAYA, A. ; WOLF, M.: *Embedded Software Design and Programming of Multiprocessor System-on-Chip*. Springer, 2010

[Sta10] STALLINGS, W.: *Computer Organization and Architecture - Designing for Performance*. 8. Edition. Pearson, 2010

[Yag03] YAGHMOUR, K.: *Building Embedded Linux Systems*. O'Reilly, 2003

[YYJ+04] YOUSSEF, M. ; YOO, S. ; JERRAYA, A. ; SASONGKO, A. ; PAVIOT, Y: *Debugging HW/SW Interface for MPSoC: Video Encode System Design Case Study*. DAC 2004, San Diego, 2004

[ZVE10] ZVEI: *Nationale Roadmap Embedded Systems*. http://www.bitkom.org/files/documents/NRMES_2009_einseitig.pdf. Version: 2010