Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW2 - SoSe 2011

Andre Jestel

Software Partitionierung auf MPSoC-Plattformen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andre Jestel

Software Partitionierung auf MPSoC-Plattformen

Ausarbeitung eingereicht im Rahmen von Anwendungen 2
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Bernd Schwarz
Gutachter : Prof. Dr.-Ing. Birgit Wendholt
Gutachter : Prof. Dr. Franz Korf

Abgegeben am 31. August 2011

Inhaltsverzeichnis

1 Einleitung

Spielekonsolen als auch Smartphones bestehen aus eingebetteten "System on a Chip" (SoC) Plattformen, die aktuell von "Multiprocessor System on a Chip" (MPSoC) Plattformen abgelöst werden [16] und deren Bedeutung im Hinblick auf Arbeitsplätze und Wertschöpfung für den Industriestandort Deutschland weiter ansteigt [4]. Auf diesen kann die Software des Systems parallel auf mehreren Prozessorkernen ausgeführt werden. Die Taktfrequenz aktueller Prozessoren stagniert auf Grund der Tatsache, dass die Ausbreitungsgeschwindigkeit der elektrischen Impulse in einem Prozessor nicht höher als die Vakuumlichtgeschwindigkeit ($c \approx 300 \cdot 10^6 \frac{m}{s}$) sein kann. Zum Zurücklegen der durchschnittlichen Flächendiagonale eines aktuellen Prozessors ist eine Taktperiode von rund 0,2 Nanosekunden erforderlich, so dass eine physikalisch maximale Taktfrequenz von 5 GHz erreichbar ist. Thermodynamische Effekte aufgrund der hohen Integrationsdichte der Halbleiter in den Chips führen dazu, dass diese Grenze nicht erreicht wird und verursachen einen steigenden Energieverlust durch Abwärme bei zunehmender Taktfrequenz.

Die parallele Verarbeitung der Software führt bei gleichbleibender Taktfrequenz zur Steigerung der Rechenleistung (Instruktionsdurchsatz). Zusätzlich zur HW-SW Partitionierungsentscheidung (Abbildung 1), muss eine Strategie zur Verteilung der Software-Threads auf die verfügbaren Prozessorkerne (*Scheduling*) entwickelt werden. Eine Prozessor-Architektur zur Parallelverarbeitung ist das "Symmetrische Multiprozessorsystem" (SMP), bei der zwei oder mehrere identische Prozessoren einen gemeinsamen Adressraum besitzen und bei der es sich um die Standard-Architektur aktueller Mehrkernprozessoren handelt. Diese werden im Gegensatz zu einem "Asymmetrisches Multiprozessorsystem" (AMP) eingesetzt, wenn keine klare Aufgabenpartitionierung durchführbar ist. Verfügt jeder einzelne Prozessor über einen Cache ist, zusätzlich zur Erhaltung der Datenkonsistenz, die Cache-Kohärenz beim Zugriff auf den gemeinsamen Speicher sicherzustellen. Dabei ist insbesondere zu berücksichtigen, ob es sich bei der Topologie des zugrundeliegenden Kommunikationsnetzwerkes um einen gemeinsamer Bus oder Punkt-zu-Punkt Verbindung etc. [2] handelt.

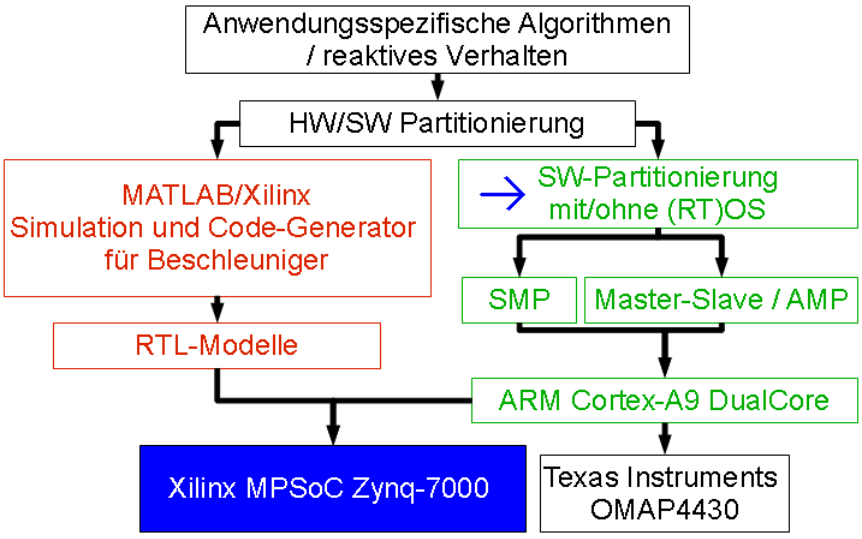

Abbildung 1: Entwicklungsstränge der HW-SW Partitionierung auf rekonfigurierbaren MPSoCs

Ziel dieser Ausarbeitung ist es, einen Überblick über aktuelle MPSoC - Forschungsschwerpunkte zu geben. Weiterhin sollen Technologie-Kompetenzen sowie Entwicklungs-KnowHow für eingebettete MPSoC-Systeme gesammelt werden. Als Anwendung werden im begleitenden Projekt-Kurs die Partitionierungsvarianten für Komponenten der laserscannerbasierte Objekterkennung [19] aus dem FAUST-SCV [5] Projekt für die "Open Multimedia Application Platform" untersucht, welche über einen ARM Cortex™-A9 mit zwei Prozessorkernen verfügt [10].

2 Aktuelle und vergleichbare Arbeiten zum Thema

In diesem Abschnitt werden fünf Konzepte zusammengefasst, die sich mit der optimalen Nutzung der Ressourcen von Multiprozessorsystemen sowie der Erhaltung der Datenkonsistenz befassen. Die Erkenntnisse und Konzepte sind darüber hinaus auch auf nicht-eingebettete Systeme übertragbar.

2.1 Symmetric Multiprocessor Systems on FPGA

Prozessoren können als "Intellectual property" (IP), also als Quellcode für Hardwarebeschreibungssprachen auf einem "Field Programmable Gate Array" (FPGA) synthetisiert werden. Dadurch lassen sich auf einem FPGA vollständige SMP-Architekturen implementieren. Vier SMP-Architekturen wurden mit *MicroBlaze*-IP Prozessoren [20] implementiert und ihre Rechenleistungen mit zwei Testanwendungen verglichen [9]. Das dazu verwendete *Xilkernel*-Betriebssystem [21], wurde zur Unterstützung des Thread-Scheduling auf mehrere μProzessoren modifiziert. Dabei identifiziert das Betriebssystem die Prozessoren, jeweils über eine ID in einem Hardware-Register und zur Synchronisation wurde ein Hardware-Mutex für den exklusiven Zugirff auf den "On-chip Peripheral Bus" (OPB) verwendet, über den die Prozessoren gemeinsam auf Peripherie und Speicher zugreifen.

Die vier getesteten Systeme charakterisieren sich durch folgende Eigenschaften:

- System 1: 2 Prozessoren, von denen jeder über 8 Kilobyte lokalen Speicher verfügt und die sich beide jeweils 64 Kilobyte Block-RAM mit direktem Zugriff als Instruktions- und Datenspeicher teilen.
- System 2: Bis zu 8 Prozessoren mit jeweils 4 Kilobyte Instruktions-Cache, ebenfalls 8 Kilobyte lokalen Speicher und 128 Kilobyte Block-RAM als gemeinsamen Speicher über den OPB-Bus.
- System 3: Bis zu 4 Prozessoren, die jeweils über 4 Kilobyte Instruktions-Cache, 4 Kilobyte lokalen Speicher und 1 Megabyte S-RAM als gemeinsamen Speicher über den OPB-Bus.
- System 4: Bis zu 4 Prozessoren, die jeweils über 4 Kilobyte Instruktions-Cache, 4 Kilobyte lokalen Speicher und 64 Megabyte DDR-RAM als gemeinsamen Speicher über den OPB-Bus.

Als Anwendungen wurden Matrizenmultiplikationen sowie ein auf AES basierender Ver- und Entschlüsselungsalgothimus verwendet. Als Metrik für die Messung wurde 'Speedup' als die Zeit definiert, die für das Abschließen einer Anwendung mit einem Prozessor benötigt wurde, geteilt durch die Zeit, die der gleiche Vorgang mit *N* Prozessoren gedauert hat. In System 1 mit 2 Prozessoren und direktem Speicherzugriff konnte ein optimaler 'Speedup' von nahezu 2 erreicht werden (Abbildung 2). Der begrenzte 'Speedup' bei steigender Prozessoranzahl in den anderen Systemen ist auf die Verwendung eines gemeinsamen Busses ohne Datencache zurückzuführen. Je mehr Prozessoren parallel rechnen und dabei auf den gemeinsamen Speicher zugreifen, je häufiger ist der Bus exklusiv von einem Prozessor belegt belegt, sodass die anderen μProzessoren auf den Bus-Zugriff warten müssen Die Ergebnisse zeigen, dass sich parallelisierbare Anwendungen ohne skalierbare Kommunikationsinfrastruktur nicht durch die Erhöhung der Prozessorenanzahl beschleunigen lassen. Das *Amdahlsche Gesetz* besagt, dass der Geschwindigkeitszuwachs von Programmen durch parallele Ausführung durch den sequentiellen Anteil des Programms beschränkt wird, da sich dessen Ausführungszeit durch Parallelisierung nicht verringern lässt. Folglich ist bei schlecht-parallelisierbaren Anwendungen mit einem geringeren 'Speedup' zu rechnen.

SYSTEM 1

	Matrices	Encrypt	Decrypt
Speedup	1.9853	1.9662	1.9809

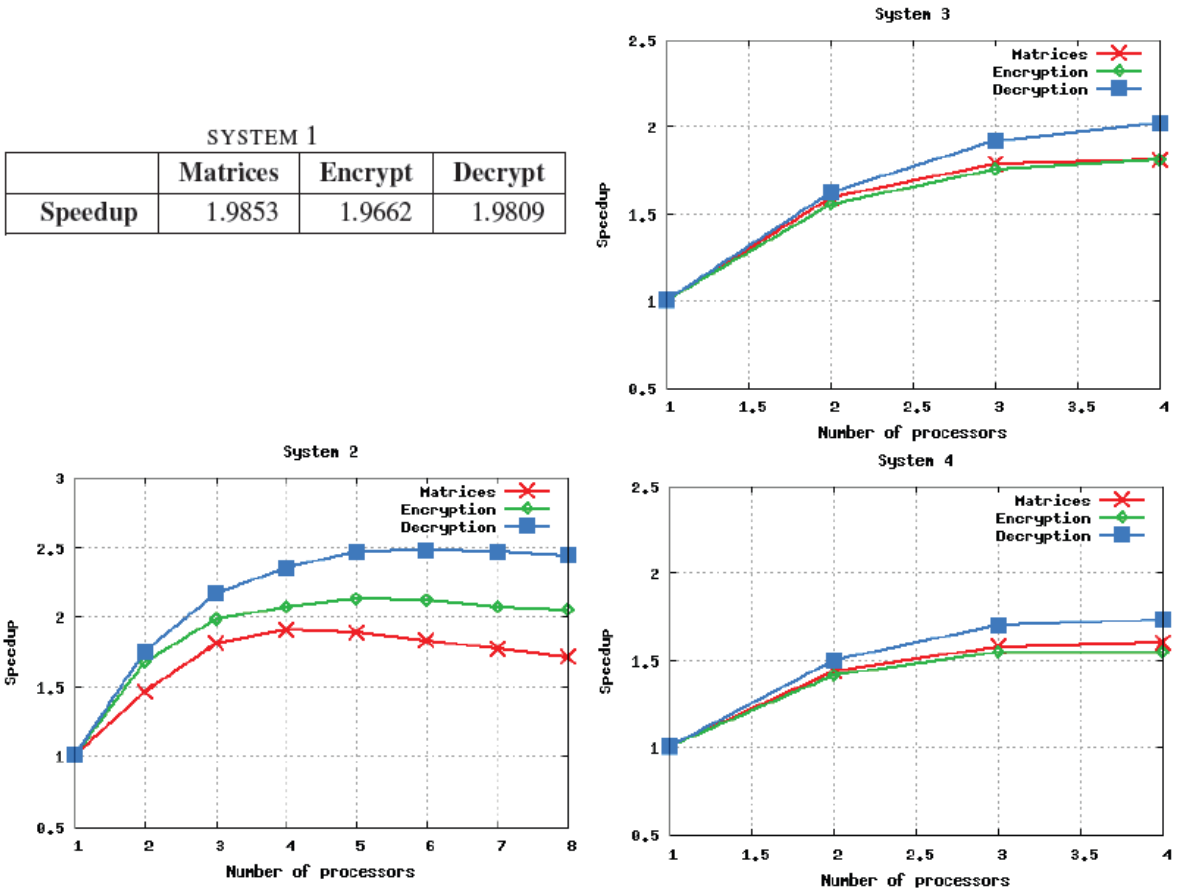

Abbildung 2: 'Speedup' - Ergebnisse für die vier getesteten Systeme in Abhängigkeit der Prozessoranzahl [9].

2.2 Hardware Transactional Memory Model

Greifen mehrere Prozessoren auf gemeinsamen Speicher zu, sind Synchronisationsmechanismen zur Erhaltung der Datenkonsistenz erforderlich. Bei der Verwendung von 'Locks', wird dies durch das Sichern des exklusiven Zugriffs auf eine Ressource gewährleistet. Während exklusiv auf eine Ressource zugegriffen wird, blockieren die Zugriffe der anderen Prozessoren bis der Speicherbereich wieder freigegeben wird. Die Nachteile dieser Ausschlussverfahren sind potentielle 'Deadlocks', bei denen das gesamte System blockiert sowie 'Priority Inversions', die auftreten wenn ein niedrig-priorisierter Software-Thread durch den exklusiven Zugriff auf eine Ressource einen höher-priorisierten Thread am Weiterarbeiten hindert.

Die "Hardware Transactional Memory" (HTM) Modelle , stellen einen Ansatz zur Konsistenzerhaltung bei gleichzeitigen Zugriffen auf einen gemeinsamen Speichers dar [13]. Eine Transaktion ist definiert als eine endliche Sequenz von Instruktionen, die atomar also nicht-unterbrechbar ausgeführt werden. Grundprinzip der HTM-Modelle ist die spekulative Annahme, das nebenläufige Transaktionen statistisch gesehen meistens konfliktfrei bleiben und diese nur bei dem Erkennen eines Zugriffskonfliktes zeitaufwendig abgebrochen und neu gestartet werden müssen. Erforderlich für den Einsatz von HTM-Modellen ist eine "Network-on-Chip" (NoC) Kommunikationsstruktur, welche durch Router nachrichtenbasiert Prozessoren und Speicher verbindet (Abbildung 3). Befindet sich ein angeforderter Speicherblock nicht im Cache des jeweiligen Prozessors (*cache miss*), fordert der entsprechende Cache-Controller diesen beim Directory an. Das Directory führt eine Tabelle, welcher Block sich zur Zeit in welchem Cache befindet. Liegt der angeforderte Block zu diesem Zeitpunkt im Cache eines anderen Prozessors, leitet das Directory die Anfrage an den aktuellen Besitzer weiter. Dieser prüft

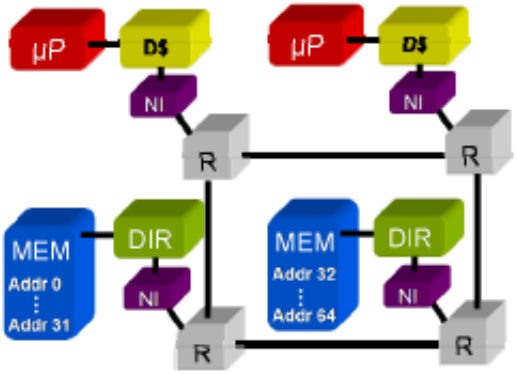

Abbildung 3: NoC-Struktur, welche die Prozessoren und Datencache über Netzwerkschnittstellen und Router mit dem 'Directory' und dem gemeinsamen verteilten Speicher verbindet [13].

anhand eines Lese- und Schreibbits in jedem Block, ob ein Zugriffskonflikt vorliegt und verweigert im Konfliktfall die Anfrage durch das Senden einer "Not Acknowledged" (NACK) Nachricht an den Anfordernden. Dieser bricht dann die initiierte Transaktion ab, stellt seinen ursprünglichen Zustand (Datenwerte und Registerinhalte) aus einem lokalen Puffer wieder her (*rollback*) und startet die Transaktion nach einer Wartezeit erneut.

Als ein Vertreter der HTM-Modelle für eingebettete Systeme wurde *LogTM* in [13] auf einer virtuellen MPSoC-Plattform im Vergleich mit Lock-Mechanismen simuliert und schnitt mit zunehmender Prozessoranzahl in Rechenleistung mit bis zu 30% und in Energieeinsparung mit bis zu 32% besser ab. Für FPGA-basierten MPSoCs stellt die NoC-Struktur, im Vergleich zu einem gemeinsamen Bus, eine skalierbare Hardware-Infrastruktur dar. Der im begleitenden Projekt verwendete Cortex-A9 Prozessor ist mit zwei Prozessorkernen konfiguriert von denen jeder über einen 4-Wege assoziativen 32KB Instruktions- und 32KB Datencache verfügt. Die L1-Cache-Kohärenz wird dabei über eine "Snoop Control Unit" sichergestellt: Diese überwacht die Speicherzugriffe und überträgt von einem Prozessorkern modifizierte Daten beim Lesezugriff des anderen Kerns innerhalb des Prozessors ohne Umweg über den Hauptspeicher direkt in dessen Cache.

2.3 Design Methodology for Application Partitioning on MPSoCs

Ein vollständiger Partitionierungsvorgang einer Anwendung auf einer MPSoC-Plattform besteht aus mehreren Teilschritten. Am *Fraunhofer-Institut* in Ettlingen wurde in Zusammenarbeit mit dem *Karlsruher Institut für Technologie* eine Methodologie zur Partitionierung entwickelt [7]. Ausgehend von einem vorliegenden C/C++ Quellcode und der Ziel-MPSoC-Plattform durchläuft der Vorgang drei Phasen, deren Teilaufgaben manuell als auch zum Teil automatisiert durch den Einsatz von kommerziellen und frei verfügbaren Werkzeugen erfüllt werden (Abbildung 4). In der ersten Phase werden zeitliche Reihenfolge, Beziehungen und Abhängigkeiten der C/C++ Funktionen analysiert und diese per hierarchischem Clustering [11] auf die Prozessoren verteilt. Kriterien für die Aufteilung sind: Gleichmäßige Verteilung der Arbeitslast auf die Prozessoren, Unabhängigkeit zwischen den Prozessoren zwecks minimalen Kommunikations-Overhead, Erfüllung der Echtzeitanforderung der Anwendung und minimaler Energieverbrauch. Phase 2 analysiert die aufgeteilten Code-Fragmente auf rechenintensive Sektionen wie zum Beispiel tief-verschachtelte Schleifen und schlägt anschließend ein Hardware-Software Co-Design vor, in dem diese Sektionen zur Steigerung der Rechenleistung als RTL-Modelle in Hardware ausgelagert werden. In der dritten Phase erfolgt anschließend die Integration der Kommunikationsschnittstellen zwischen den Hardware- und Software Komponenten, die Kompilierung der Software

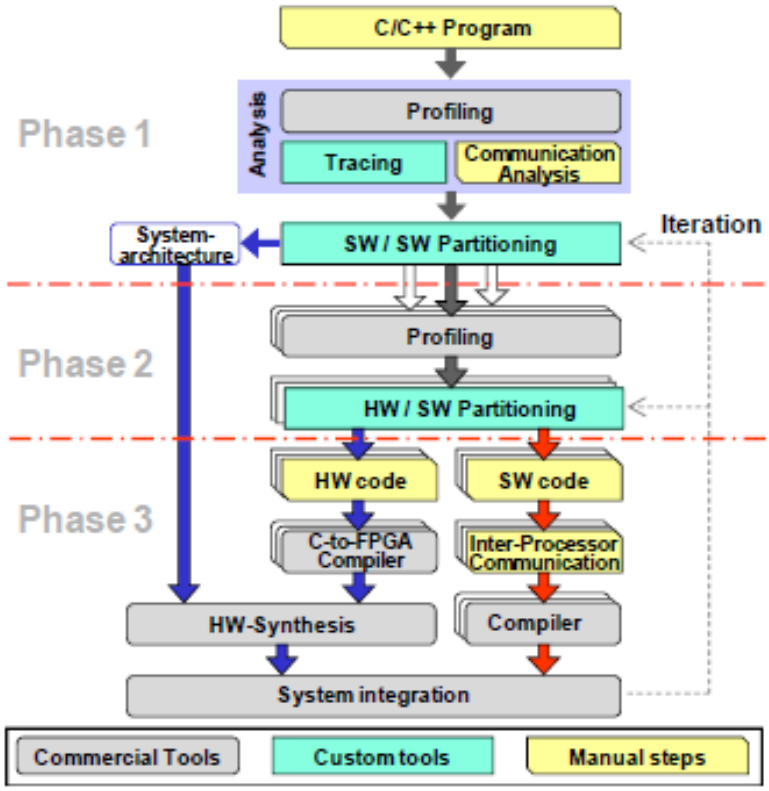

Abbildung 4: Die Partitionierungs-Methodologie unterteilt in drei Phasen und Teilaufgaben mit den Formen der beteiligten Werkzeuge [7].

für die Prozessoren und die Synthese der RTL-Modelle auf dem FPGA. Die drei Phasen werden ggf. iterativ solange wiederholt bis das Ergebnis sämtliche Anforderungstests besteht.

Die Funktionalität dieser Design-Methodologie wurde auf einem "Runtime Adaptive MPSoC" [6] anhand komplexer Bildverarbeitungsalgorithmen evaluiert. Für die Hardware-Software als auch Software-Software Partitionierung der Anwendungen im Projekt-Kurs stellt die Methodologie einen Leitfaden dar, der für jeden Teilschritt geeignete Werkzeuge benennt und auf dessen Zweck und Bedeutung für den gesamten Prozess hinweist.

2.4 Co-Optimization of Memory Access and Task Scheduling

Die maximal verfügbare Datenverarbeitungsgeschwindigkeit von Prozessoren wird durch zeitaufwendige Speicherzugriffe nicht erreicht. Um diese Zeitkosten bei häufigen Speicherzugriffen zu reduzieren, verfügen die Prozessoren über Caches, in denen Speicherblöcke (*pages*) lokal zwischengespeichert werden. Auf diese kann in der Regel um den Faktor 10 schneller zugegriffen werden als auf den Hauptspeicher. Da Caches aus Kostengründen nur eine begrenzte Speicherkapazität besitzen, sind Strategien zur Seitenersetzung notwendig, die entscheiden wann welcher Speicherblock aus dem Cache durch einen anderen Block aus dem Hauptspeicher ersetzt wird. Um solche zeitaufwendige Ersetzungen (*memory swaps*) zu reduzieren, gilt es das Ausführen von Software-Tasks, die häufig auf den selben Speicherblock zugreifen, zeitlich dicht aneinander zu gruppieren. Bei der Standard-Strategie "Least Recently Used" (LRU) wird stets der Block ersetzt, der am längsten unbenutzt blieb. Die theoretisch optimale zeitliche Partitionierung der Threads lässt sich durch das Aufstellen von Ungleichungen, welche die einzuhaltenden Randbedingungen beschreiben, mit einer "Ganzzahligen linearen Optimierung" (ILP) berechnen [14]. Das Lösen von ILPs besitzt ein exponentielles Laufzeitverhalten und ist daher in der Praxis nicht effizient durchführbar.

Mit "The Least Projected Cost Algorithm" wurde eine Heuristik entwickelt (Abbildung 5), die mit linearem Laufaufwand einen bestehenden Scheduling-Plan optimiert. Dieser schnitt in sechs experimentell getesteten Benchmarks im Durchschnitt um 30% besser ab als der ursprüngliche Scheduling-Plan mit der LRU-Strategie, während zum Vergleich die theoretisch optimale Lösung des ILPs im Durchschnitt um 43% besser abschnitt.

Require: Task Graph $G_t = \langle V, E, P \rangle$, the number of cores in the system *cores*, the page load cost for a page *cost*, the capacity of the level 1 memory *capacity*
Ensure: a task schedule SL to minimize the accumulative page load cost APLC
Schedule $SL \leftarrow \emptyset$
Ready List $RL \leftarrow \emptyset$
for all $v \in V \wedge v \notin RL$ **do**
  if v has no incoming edge **then**
    $RL \leftarrow RL \cup \{v\}$
    $min \leftarrow \infty$
    for all $p \in P(v)$ (pages v accesses) **do**
      $t \leftarrow$ the minimum steps away p is accessed again
    end for
    if $t < min$ **then**
      $min \leftarrow t$
    end if
    $projected(v) \leftarrow min$
  end if
end for

while $RL \neq \emptyset$ **do**
  $v \leftarrow$ the node with the smallest $projected(v)$
  Collection $PS \leftarrow$ positions that v can be inserted in SL
  for all $ps \in PS$ **do**
    calculate $APLC$ for the schedule
    use MOPT as the page replacement policy
    remember the position pos if $APLC$ is the minimum
  end for
  schedule v at position pos
  $SL \leftarrow SL \cup v$
  remove v from V and all out going edges from v
  update Ready List RL
  for all $v \in RL$ **do**
    update $projected(v)$ if affected
  end for
end while

Abbildung 5: "The Least Projected Cost Algorithm" als Pseudocode [8].

Die Tasks werden mit den Speicherblöcken auf die sie zugreifen zusammen als Knoten in einem gerichteten azyklischen Graphen an den Algorithmus übergeben. Eine gerichtete Kante repräsentiert eine Abhängigkeit zwischen zwei Tasks, die besagt das der Task, aus dem die Kante herausgeht, zeitlich in jedem Fall vor dem Task ausgeführt werden muss, in den die Kante hinein zeigt. Im ersten Schritt werden alle Tasks ohne eingehende Kante in der "Ready List" (RL) gespeichert. Für jeden Speicherblock auf den jeder Task in der RL zugreift wird gemäß des bestehenden Scheduling-Plans der minimale zeitliche Abstand ermittelt, nachdem von einem anderen Task erneut auf diesen Speicherblock zugegriffen wird. Anschließend wird der Task, der auf den Speicherblock zugreift, für den der kleinste Abstand ermittelt wurde, aus der RL entfernt und in die "Scheduling List" (SL) eingefügt. Für jede mögliche Position an welcher der Task in der SL platziert werden kann, werden die gesamten Zeitkosten (*accumulative page load cost* APLC) des aktuellen Scheduling-Plans berechnet und die Position mit der geringsten APLC ausgewählt. Zuletzt wird die RL aktualisiert und das Verfahren solange wiederholt bis alle Tasks in der SL stehen. Anstelle von LRU werden die Seiten nach der für den Scheduling-Plan optimalen Strategie ersetzt (MOPT).

Liegt der Task-Graph einer oder mehrerer (Teil)Anwendungen vor, lassen sich diese unabhängig von der Kommunikationsinfrastruktur und Parallelisierbarkeit mit dieser Heuristik durch die Reduzierung der Seitenersetzungen beschleunigen. Bei Anwendungen für MPSoCs, deren Teilaufgaben und ihre Abhängigkeiten sich dynamisch zur Laufzeit ändern, kann der Task-Graph periodisch aufgestellt und optimiert werden. Sind dagegen bereits in der Entwicklungsphase die Teilaufgaben bekannt und die Speicheradressen statisch, kann der Task-Graph einmalige aufgestellt und optimiert werden.

2.5 Inter-core Prefetching Using Helper Threads

Mit der *inter-core prefetching* Technik [12] wurde ein Konzept entwickelt, welches zur Migration von rein sequentiell entwickelter Programme unter Nutzung aller verfügbaren Prozessoren auf Mehrkern-Plattformen dient. Ziel dabei ist das Beschleunigen der Anwendungen ohne explizites Parallelisieren bzw. Partitionieren des Programmcodes durch den Entwickler. Dazu werden Helper-Threads (oder Prefetch-Thread) erzeugt, die parallel zum Main-Thread auf den anderen Prozessoren Lese-Zugriffe auf die Daten ausführen, auf die zukünftig vom Main-Thread zugegriffen wird. Dadurch gelangen die Datenblöcke in die Caches der Prozessoren, auf denen die Helper-Threads laufen (*fetching*). Der Main-Thread lößt nach der Abarbeitung seines aktuellen Programmcodeabschnitts (*chunk*) den Helper-Thread auf dessen Prozessor ab und kann nun mit der nächsten Teilaufgabe fortfahren (Abbildung 6). Die Daten, auf die in der nächsten Teilaufgabe zugegriffen wird, liegen nun bereits im Cache des Prozessors vor und ersparen somit das zeitaufwendige Nachladen der Daten aus dem gemeinsamen Speicher.

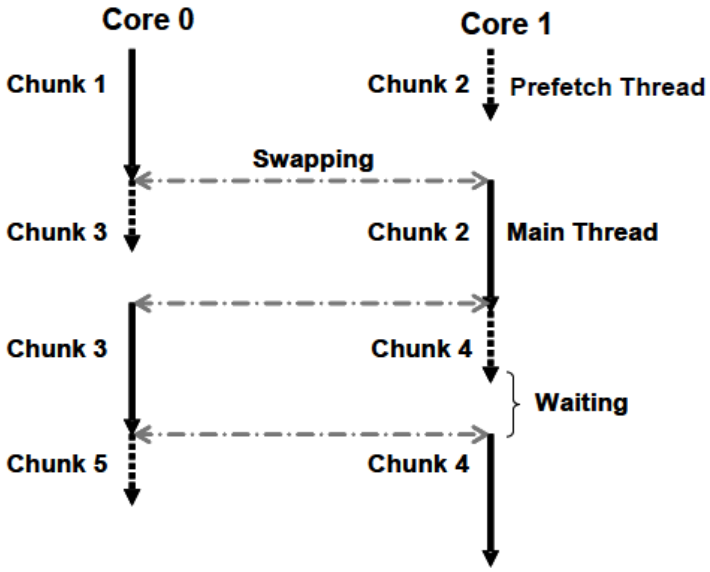

Abbildung 6: Ausführungspfad mit *inter-core prefetching* bei sich der Main-Thread (durchgehende Linie) mit einem Helper-Thread (gestrichelte Linie) abwechselnd zwischen zwei Prozessoren (Core 1 und 2) bewegen (*swapping*) [12]

Zur Nutzung aller verfügbaren Ressourcen wird für jeden Prozessor ein Helper-Thread generiert, die vom Main-Thread nacheinander 'verfolgt' werden. Der abgelöste Helper-Thread wechselt synchron auf den Prozessor, der zuvor vom Main-Thread belegt wurde. Dieser Wechsel wird durch die Verwendung von POSIX-Threads und den Aufruf der Standard-Linux-APIs: *setcontext, getcontext* und *swapcontext* realisiert. Das Konzept kann in eingebetteten MPSoC-Plattformen für eine spezifische Anwendung mit plattformspezifischen Funktionen für den Thread-Kontextwechsel auch ohne Betriebssystem verwendet werden und somit mehrere Prozessoren nutzen, wenn sich der Quellcode nicht statisch auf die Prozessoren verteilen lässt.

Es handelt sich um eine Softwaretechnik, die keine Unterstützung oder Modifizierung der verwendeten Hardware erfordert. Lediglich die Größe der Chunks wirkt sich je nach verwendeten Prozessoren und Cache-Kohärenz Protokoll auf die Steigerung der Rechenleistung gegenüber der Ausführung mit einem Prozessor aus. Zu klein gewählte Chunk-Größen führen zu häufigeren Thread-Wechseln, deren Overhead die Beschleunigung der Anwendung reduziert. In den Testergebnissen konnten die Anwendungen auf 3 Mehrkernprozessor-Architekturen mit 10 Cache-Größen in 13 Benchmarks zwischen 31 und 63% beschleunigt und gleichzeitig der Energieverbrauch um bis zu 50% reduziert werden.

3 Untersuchung der SMP-fähigen OMAP4430-Plattform zur SW/SW-Partitionierung

Im FAUST-SCV Projekt [5] wurde ein fahrerloses Transportsystem entwickelt, welches über Laserscanner-gestützte Park-, Brems- und Ausweichassistenten [19] verfügt und durch eine Kamera-gestützte Spurführung ergänzt worden ist [15] (Abbildung 7). Ein FPGA basiertes System on Chip

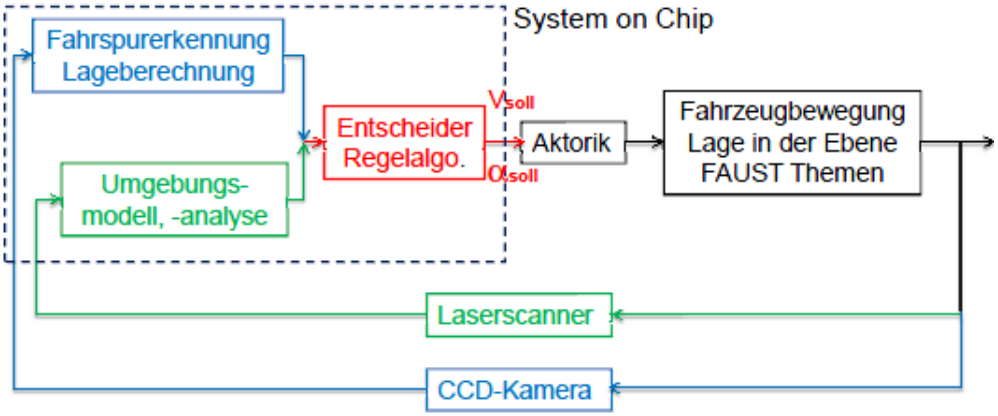

Abbildung 7: SoC-implementierte Funktionalitäten des "Sensor Controlled Vehicle" [5].

bietet für die ereignis- und zeitgesteuerten Teilfunktionen des SCV eine Hardware- und Softwareplattform, die mit paralleler und rekonfigurierbarer Hardware die Echtzeitanforderungen der digitalen Signalverarbeitungsaufgaben unterstützt.

Die Fahrerassistenzfunktionen werden mit einem Hardware-Software Co-Design im "High Performance Embedded Computing" (HPEC) Projekt aktuell für ein Modellfahrzeug im Maßstab 1:10 auf dem FPGA Spartan3e implementiert [1]. Bisher läuft die Software für den Einparkassistent auf einem *MicroBlaze*-IP Prozessor, während die Bildverarbeitung für die Spurführung mit RTL-Modellen beschrieben und in Beschleuniger-Pipelines implementiert ist. Die Datenreduktion zur Objekterkennung des auf die SoC-Plattform zu übertragenden Ausweichassistenten sowie weitere Funktionen der Bildverarbeitung erfordern zur Einhaltung der Echtzeitanforderungen eine rechenleistungsfähigere MPSoC-Plattform. Das Erscheinungsdatum der dafür vorgesehenen *Extensible Processing Platform* [17], die einen ARM Cortex™-A9 mit zwei Kernen und konfigurierbaren Hardware-Beschleunigern enthält, steht erst im Laufe des Jahres 2012 mit Evaluation-Boards zur Verfügung

Zur Einarbeitung in die SW/SW-Partitionierung wird deshalb die "Open Multimedia Application Platform" (OMAP) 4430 von *Texas Instruments* (Abbildung 8) auf dem *PandaBoard* verwendet. Anforderungen und Ablauf der SW-Thread-Partitionierung können damit auf Hardware-, Betriebssystem- und Anwendungsebene erschlossen werden, da sowohl die Schaltpläne als auch die Quellcodes der SMP-fähigen Betriebssysteme *Linux-Ubuntu* und *Android* für die Plattform verfügbar sind. Ziel ist es zunächst, die Arbeitsschritte zum Ausführen von Software ohne Betriebssystem zu entwickeln und für anschließende Projekte zu dokumentieren. Diese DualCore-nahe Auseinandersetzung mit der vorliegenden SMP-Architektur dient zur Erschließung der partitionierungsrelevanten Eigenschaften, die es bei SW/SW-Partitionierungsprozessen zu berücksichtigten gilt. Vorteil der Betriebssystem-freien Nutzung ist dabei die vollständige Kontrolle über die Prozessor- und Peripheriekomponenten der Plattform. So werden beispielsweise ausschließlich die Komponenten Strom verbrauchen, die für die konkrete Anwendung aktiviert und genutzt werden. Die Untersuchungsschwerpunkte dabei sind:

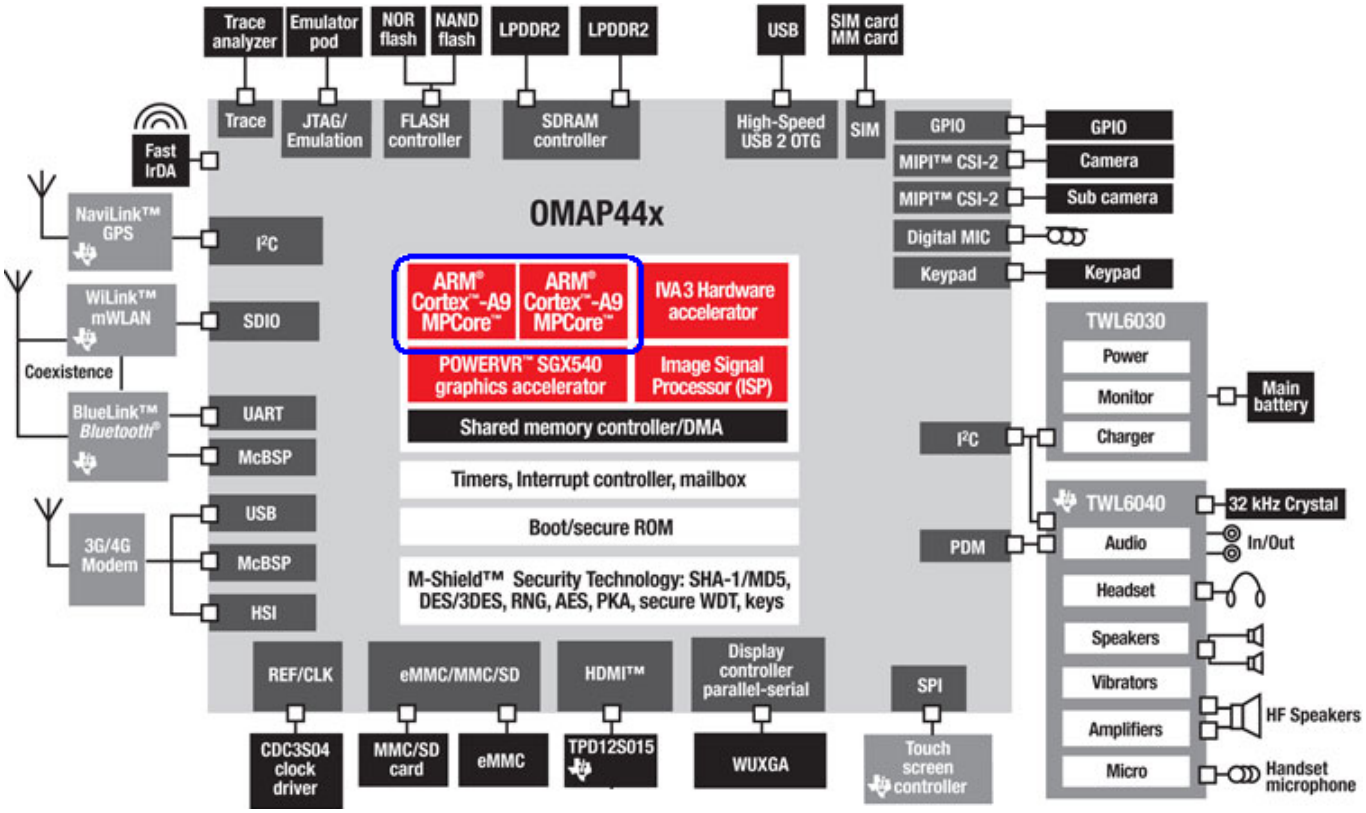

Abbildung 8: OMAP-Plattform mit integrierten DualCore ARM Cortex™-A9 Prozessor (1 GHz), 1GB DDR2-RAM und verfügbaren Peripherieschnittstellen [10]

- Anwendung von eingebetteten, ARM-spezifischen Assembler-Instruktionen im C/C++ Quellcode, mit denen ein manuelles Starten und Anhalten der einzelnen Prozessorkerne sowie die Implementierung von Synchronisationsmechanismen wie Mutexe, Semaphoren oder Message-Parsing realisierbar sind.
- Der Ablauf des Boot-Vorgangs, beim dem der *StartUp*-Code im ROM das kompilierte Programm über eine SD-Karte oder eine serielle Schnittstelle (RS232, USB) ins RAM lädt,von dort aus ausführt und ggf. weitere Programme nachlädt.
- Identifizierung der Memory-Mapped I/O Adressen mit denen per Lesen und Schreiben von Registerinhalten (anwendungsspezifische) Treiber für die Peripherieanschlüssen entwickelt werden.
- Verwendung der Timer-Bausteine und Interrupt-Serviceroutinen zur Realisierung von Echtzeitanwendungen durch das Reagieren auf externe und zeitabhängige Ereignisse.
- Manuelles Freigabesteuerung der mehrstufigen Caches und der "Memory Management Unit" (MMU) um festzustellen, welchen Einfluss die Caches und virtuelle Speicherverwaltung mit ihren Seiten-Ersetzungsstrategien auf die Rechenleistung haben.
- Sichern und Wiederherstellen der (Co)Prozessorregister beim Kontextwechsel von Software-Threads für die Erprobung von statischen Scheduling-Strategien zur Arbeitslastaufteilung auf die Prozessorkerne.

4 Fazit

4.1 Zusammenfassung

MPSoC-Plattformen lösen aktuell SoC-Plattformen für die Implementierung eingebetteter Anwendungen ab. MPSoCs verfügen über mehrere Prozessoren oder einen Mehrkernprozessor auf denen Software parallel ausgeführt wird, um den Instruktionsdurchsatz zu steigern, ohne dass die Taktfrequenz erhöht wird. Bei einem "Symmetrischen Multiprozessorsystem" (SMP) handelt es sich um die aktuelle Standard-Architektur, bei der mehrere identische Prozessoren über einen gemeinsamen Adressraum verfügen. Dadurch ergeben sich neue Teilaufgaben während der Software-Software Partitionierungsphase, wie die Verteilung der Software-Threads auf die verfügbaren Prozessoren und die Einhaltung der Datenkonsistenz bei Zugriffen auf den gemeinsamen Speicher. Ziel der Partitionierungen ist die maximale Parallelisierung der Teilaufgaben der Anwendung. Die vorgestellten Arbeiten beschreiben dazu evaluierte Konzepte und liefern Erkenntnisse sowohl auf Hardware- als auch auf Softwarebene:

1. Beschleunigung der Ausführungsgeschwindigkeit von parallelisierbaren Anwendungen unter Nutzung mehrerer Prozessoren, die über einen gemeinsamen Bus kommunizieren, ist ohne Caches nicht realistisch.
2. Ein "Network-on-Chip" ’(NoC) ist eine skalierbare Hardware-Infrastruktur, die durch Nachrichten-basierte Kommunikation zwischen Prozessoren und Speicher durch den Einsatz von "Hardware Transactional Memory" (HTM) Modellen ohne Lock-Mechanismen die Cache-Kohärenz sicherstellt.
3. Der vollständige Anwendungs-Partitionierungsvorgang lässt sich in drei sequentielle und wiederholbare Phasen unterteilen, deren Teilaufgaben durch Kombination durch manuellen Vorgängen sowie den Einsatz von kommerziellen und frei verfügbare Werkzeugen gelöst werden.
4. Eine Verteilung von Tasks auf mehrere Prozessoren, so dass Speicherzugriffszeiten durch Vermeiden von Seitenersetzungen minimal werden, lässt sich nicht effektiv (voraus)berechnen, kann jedoch durch Heuristiken mit linearer Laufzeit annähernd erreicht werden.
5. Software-generierte Prefetch-Threads dienen auf ungenutzten Prozessoren zum Füllen ihrer Caches, damit der Main-Thread nach Abarbeitung einer Teilaufgabe auf den Prozessoren mit den vorbereiteten Caches weiter arbeiten kann.

4.2 Ausblick

Die OMAP-Plattform mit einem DualCore ARM Cortex™-A9 Prozessor wird im begleitenden HPEC-Projekt für Software-Software Partitionierungsvarianten untersucht. Ziel ist die Erfassung der Hardware- und Softwareeigenschaften des symmetrischen Multiprozessorsystems dieser Plattform, auf der laserscannerbasierte Objekterkennungsfunktionen für ein Modellfahrzeug implementiert werden. Es gilt zu dokumentieren, wie eine SW-SW Partitionierungsvariante auf der Plattform ohne Betriebssystem mit welchen Prozessorfunktionen zu Nutzung des Mehrkernprozessors durchgeführt werden kann. Anschließend ist für weiterführende Implementierungen von Fahrerassistenzfunktionen zu entscheiden, ob eine SMP mit einem RTOS oder Master-Slave Prozessorkonfigurationen den Anforderungen genügt .

Literatur

[1] ALPERS, Thorsten: *Modellierung eines Einparkassistenten für ein autonomes Fahrzeug implementiert auf einer SoC-Plattform*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2010

[2] BENGEL, Günther ; BAUN, Christian ; KUNZE, Marcel ; STUCKY, Karl-Uwe: *Masterkurs Parallele und Verteilte Systeme: Grundlagen und Programmierung von Multicoreprozessoren, Multiprozessoren, Cluster und Grid*. Vieweg+Teubner, 2008. – ISBN 978-3834803948

[3] BORDASCH, Heiko: *Network on Chip - Routing and Switching*. Hochschule für Angewandte Wissenschaften Hamburg: Seminar Anwendungen 2 Sommersemester 2010. 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-aw2/bordasch/bericht.pdf. – Zugriffsdatum: 01.06.2011

[4] DAMM, Werner ; ACHATZ, Reinhold ; BEETZ, Klaus ; BROY, Manfred ; DAEMBKES, Heinrich ; GRIMM, Klaus ; LIGGESMEYER, Peter: *Nationale Roadmap Embedded Systems*. Springer, 2010. – 67–136 S. – ISBN 978-3642149016

[5] FAUST-PROJEKT: *Sensor Controlled Vehicle*. Website. 2011. – URL http://www.informatik.haw-hamburg.de/faust.html. – Zugriffsdatum: 20.01.2011

[6] GÖHRINGER, Diana ; BECKER, Jürgen: High Performance Reconfigurable Multi-Processor-Based Computing on FPGAs. In: *IEEE* (2010)

[7] GÖHRINGER, Diana ; HÜBNER, Michael ; BENZ, Michael ; BECKER, Jürgen: A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip. In: *18th IEEE Annual International Symposium* (2010)

[8] HE, Yi ; XUE, Chun J. ; XU, Cathy Q. ; SHA, Edwin H.-M.: Co-Optimization of Memory Access and Task Scheduling on MPSoC Architectures with Multi-Level Memory. In: *IEEE* (2010), März

[9] HUERTA, Pablo ; CASTILLO, Javier ; PEDRAZA, Cesar ; CANO, Javier ; MARTINEZ, Jose I.: Symmetric multiprocessor systems on FPGA. In: *International Conference on Reconfigurable Computing and FPGAs* (2009), November

[10] INSTRUMENTS, Texas: *OMAP4430 Platform*. Website. 2011. – URL http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?contentId=53243&navigationId=12843&templateId=6123. – Zugriffsdatum: 29.05.2011

[11] JESTEL, Andre: *Hardware-Software Partitionierung für SoC-Plattformen*. Hochschule für Angewandte Wissenschaften Hamburg: Seminar Anwendungen 1 Wintersemester 2010/11. 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-aw1/jestel/bericht.pdf. – Zugriffsdatum: 05.04.2011

[12] KAMRUZZAMAN, Md ; SWANSON, Steven ; TULLSEN, Dean M.: Inter-core Prefetching for Multicore Processors Using Migrating Helper Threads. In: *ACM* (2011), März

[13] KUNZ, Leonardo ; GIRAO, Gustavo ; WAGNER, Flavio R.: Evaluation of a Hardware Transactional Memory Model in an NoC-based Embedded MPSoC. In: *ACM* (2010), September

[14] MARWEDEL, Peter: *Embedded System Design*. Springer, 2005. – ISBN 978-0387292373

[15] MELLERT, Dennis: *Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx SystemGenerator für eine SoC-Plattform.*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2010

[16] MIGNOLET, Jean-Yves ; WUYTS, Roel: Embedded Multiprocessor Systems-on-Chip Programming. In: *IEEE Software* (2009), September, S. 34–37

[17] SANTARINI, Mike: Xilinx Architects ARM-Based Processor-First, Processor-Centric Device. In: *Xcell Journal* (2010), März, S. 6–11

[18] SASS, Ronald ; SCHMIDT, Andrew G.: *Embedded Systems Design with Platform FPGAs: Principles and Practices*. Morgan Kaufmann, 2010. – ISBN 978-0123743336

[19] SCHETLER, Denis: *Automatischer Ausweichassistent mit einer Laserscanner - basierten Abstandsregelung für ein fahrerloses Transportsystem*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2007

[20] XILINX: *MicroBlaze Soft Processor Core*. Website. 2011. – URL http://www.xilinx.com/tools/microblaze.htm. – Zugriffsdatum: 04.05.2011

[21] XILINX: *Xilkernel 5.00a: OS and Libraries Document Collection*. Website. 2011. – URL www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/oslib_rm.pdf. – Zugriffsdatum: 04.05.2011