Multithreading als Programmier- und Ausführungsmodell für rekonfigurierbare Hardware

Enno Lübbers

Institut für Informatik, Universität Paderborn / EADS Innovation Works, München
enno.luebbers@{upb.de / eads.net}

Abstract: Zunehmende Integrationsdichte und die Verbindung von programmierbarer Logik mit leistungsfähigen Mikroprozessoren haben dazu geführt, dass moderne Field-Programmable Gate Arrays (FPGAs) zunehmend aus ihren klassischen Rollen als Verbindungslogik, Emulations- und Prototypisierungsplattform ausbrechen und zur Umsetzung vollständiger rekonfigurierbarer "Systems-on-Chip" genutzt werden. Konventionelle Entwurfsmethoden eignen sich jedoch nur unzureichend zur Programmierung dieser hybriden rekonfigurierbaren Computer. Insbesondere überlappen sich die Programmiermodelle für Software (SW) und digitale Hardware (HW) nicht ausreichend, was das Potential für die Wiederverwendung von Modulen und die flexible Nutzung des HW/SW-Entwurfsraumes stark schmälert. Darüber hinaus findet die vielversprechende Möglichkeit zur partiellen Rekonfiguration, wie sie moderne FPGAs bieten, bisher keine Entsprechung in einem verbreiteten Entwurfsverfahren. Die vorliegende Arbeit stellt ein neuartiges und grundlegendes Konzept der Programmierung von Hardwaremodulen als Threads für programmierbare Hardware vor. Neben der Übertragung des Multithreading-Programmiermodells, welches bereits verbreitet und erfolgreich in Softwareumgebungen eingesetzt wird, auf rekonfigurierbare Hardware zeigt sie ein Laufzeitsystem namens ReconOS, das auf bestehenden eingebetteten Betriebssystemen basiert und die Erstellung von portablen und flexiblen HW/SW-Anwendungen für partiell rekonfigurierbare CPU/FPGA-Systeme ermöglicht.

1 Einleitung

Die Wahl der richtigen Architektur für ein eingebettetes System ist und war immer ein Balanceakt zwischen konkurrierenden Zielen und Beschränkungen. Heutige Entwickler sehen sich einem breiten Spektrum an verfügbaren Rechenelementen gegenüber – von flexiblen Mikroprozessoren über digitale Signalprozessoren (DSPs), Prozessoren mit anwendungsspezifischem Instruktionssatz (ASIPs) und FPGAs bis zu hochspezialisierten anwendungsspezifischen integrierten Schaltungen (ASICs). Jede dieser Klassen an Rechenelementen verfügt über einen eigenen Satz an Programmiermodellen und Entwurfswerkzeugen, die sich an den Spezifika der jeweiligen Zieltechnologie orientieren. Durch die Berücksichtigung von Entwurfsaspekten entwickelt sich die Wahl einer dem Problem angemessenen Ausführungsumgebung von einem einfachen Trade-Off zwischen Kosten und Rechenleistung zu einem deutlich komplexeren Problem.

Moderne eingebettete Systeme benötigen zunehmend ein hohes Maß an Rechenleistung, beispielsweise für Aufgaben aus der Signalverarbeitung, Kompression, oder Kryptographie. Gleichzeitig sind solchen Systeme enge Grenzen in punkto Kosten, Energieverbrauch

und auch Entwicklungsaufwand gesteckt. Getrieben durch die steigenden Kosten moderner Prozesstechnologien und die hohe Leistungsaufnahme aktueller Prozessorfamilien ist der Bedarf an sogenannten *off-the-shelf*-Komponenten (oder auch Plattformen) mit geringem oder zumindest skalierbarem Energieverbrauch stetig gestiegen. Dadurch hat sich das Spektrum der anwendbaren Rechnerarchitekturen auf eingebettete Prozessoren, DSPs und FPGAs verengt, was dazu führt, dass vermehrt Prozessoren, DSP-Elemente und andere spezialisierte Komponenten in *Plattform-FPGAs* integriert werden und nun vollständig integrierte *rekonfigurierbare Systems-on-Chip* (rSoCs) bilden.

Leider hat die Entwicklung von Entwurfsmethoden und Werkzeugen für programmierbare Logik nicht mit der rasanten Entwicklung dieser Bausteine Schritt gehalten. Traditionelle Entwurfsmuster, welche spezialisierte Hardwaremodule durchgängig als passive Koprozessoren betrachten, eignen sich nur bedingt zur Programmierung rekonfigurierbarer Rechner. Insbesondere überlappen sich die Programmiermodelle für Software (ausgeführt auf eingebetteten Betriebssystemen) und digitaler Hardware (synthetisiert auf einen FPGA) nicht ausreichend, was die Nutzung des Entwurfsraums und die Wiederverwendung von Komponenten erheblich einschränkt. Auch findet das vielversprechende Konzept der *dynamischen partiellen Rekonfiguration* (DPR), also der Fähigkeit eines Bausteins, statt der gesamten Fläche nur einen Teil seiner Logik zur Laufzeit neu zu programmieren, bisher keine Entsprechung in einem ganzheitlichen Programmiermodell. Entgegen der statischen Konfiguration von rekonfigurierbarer Hardware, welche durchgängig verstanden ist und produktiv eingesetzt wird, ist die DPR noch immer ein sehr aktives Forschungsfeld. Insbesondere existiert noch kein einheitlicher Entwurfsfluss, um sowohl die Laufzeitumgebung als auch die zur Konfiguration nötigen partiellen Bitströme aus einer gemeinsamen Applikationsbeschreibung zu generieren.

Das Thema meiner Dissertation [Lüb10] ist ein einheitliches Programmier- und Ausführungsmodell, ReconOS, für hybride Hardware / Software-Systeme. Meine Arbeit beschreibt eine grundlegend neuartige Methode zum Entwurf von Applikationen auf modernen Plattform-FPGAs, die zum ersten Mal sowohl sequentiell ausgeführte Software als auch parallel arbeitende Hardware mit einem gemeinsamen Programmiermodell einschließt. Das Programmier- und Ausführungsmodell basiert auf dem im Bereich der Softwareentwicklung weit verbreiteten expliziten Multithreading in Kombination mit einem robusten Betriebssystemkernel. Mein Ansatz hebt Hardwaremodule erstmalig aus ihrer Rolle als passive Koprozessoren heraus und setzt sie als *Hardware-Threads* auf eine Ebene mit Software-Threads – mit Zugriff auf dieselben Betriebssystemressourcen. Dieser vereinheitlichte Entwurfsprozess erlaubt ein erhöhtes Maß an Portabilität und Wiederverwendbarkeit von Software- und Hardwarekomponenten und vereinfacht die Ausnutzung des gesamten Entwurfsraums eines hybriden Hardware-/Software-Systems; das Programmiermodell erhöht die Entwurfsproduktivität für Plattform-FPGAs auf eine Stufe, die zuvor nur rein softwarebasierten Systemen vorbehalten war. Gleichzeitig verwendet das Laufzeitsystem die Möglichkeiten zur partiellen Rekonfiguration, um die durch das Programmiermodell explizit gemachte Parallelität auf Threadebene weiter auszunutzen.

2 Hintergrund

Betriebssysteme für rekonfigurierbare Rechner bieten Forschungsansätze aus mehreren unterschiedlichen Perspektiven. Brebner [Bre96] betrachtete als einer der ersten Hardware-Multitasking als Bestandteil der Betriebssystemschicht, während andere Autoren bald weitere Dienste zur geometrischen Modifikation, Scheduling, Partitionierung, Allokation und Routing von Hardware-Tasks vorschlugen.

Die weitere Forschung konzentrierte sich auf einzelne, eigenständige Funktionen zukünftiger Hardware-Betriebssysteme. Insbesondere die Themen Platzierung und Scheduling von Hardware-Tasks wurde unter einer Vielzahl von Task- und Ressourcenmodellen betrachtet [TFS00]. Viele effiziente Schedulingverfahren, insbesondere solche für Echtzeitsysteme, setzen die unmittelbare Unterbrechbarkeit von Tasks voraus, was für Hardware aufgrund ihrer spezialisierten Ausführungsumgebung eine schwierige Herausforderung darstellt [KP05].

Viele der frühen Arbeiten waren entweder theoretisch oder wurden an Simulationen oder synthetischen Problemen evaluiert, da sich bis dato keine Betriebssystemimplementierungen oder akzeptierte Benchmarks durchsetzen konnten. Es entstanden eine Reihe von Prototypen, welche jedoch stets durch die verfügbaren Technologien und Werkzeuge beschränkt waren. Im Besonderen betrachteten die genannte Arbeiten Hardware-Tasks durchgängig als passive Koprozessoren denn als *unabhängige Ausführungseinheiten*. Einen ersten Schritt in diese Richtung zeigten Steiger, Walder und Platzner [SWP04] mit einem Prototypen, der Hardware-Tasks einen höheren Grad an Autonomie einräumt und ihnen Zugriff auf Betriebssystemobjekte wie FIFOs, Speicher, und I/O-Treiber ermöglicht. Viele neuere Arbeiten versuchen zunehmend, rekonfigurierbare Hardware mit Linux zu verbinden [KMK07, BWHC06, SB06].

Die bisherigen Ansätze koppeln Schaltungen in rekonfigurierbaren Bausteinen mit bestehenden Betriebssystemen, um die Kommunikation zwischen Hardware und Software zu erleichtern. Während dadurch der Erstellung von Hardware / Software-Systemen zu einem gewissen Grad vereinfacht wird, wird der Entwurfsraum für einen einzelnen Hardware-Tasks jedoch eingeschränkt, da nur ein einzelner spezifischer Dienst zur Kommunikation zur Verfügung steht. Im Gegensatz dazu vertritt die vorliegende Arbeit die Ansicht, dass nur durch die gemeinsame Unterstützung von Hardware und Software in einem einheitlichen Programmiermodell durch gleichartig verfügbare Betriebssystemdienste das volle Potenzial hybrider Hardware / Software-Systeme ausgeschöpft werden kann, ohne die Portierbarkeit einer Applikation zwischen unterschiedlichen Zielplattformen zu gefährden.

3 Multithreading als Hardwareprogrammiermodell

Ein weitverbreitetes Modell zur Verbindung von Hardwarebeschleunigern, die in rekonfigurierbarer Hardware realisiert sind, und softwarebasierten Systemen ist das des *Slave-Koprozessors*. Dabei ist die spezialisierte Logik des Hardwarebeschleunigers – oft zusammen mit den übrigen Peripheriekomponenten des Systems – an einen zentralen Bus angeschlossen. Die Kommunikation zwischen CPU und Beschleuniger findet hier oft über spezifische Schnittstellen, wie etwa dedizierte Register, statt. Dieses Kommunikationsmodell

für rekonfigurierbare SoCs wird auch von den gängigen kommerziellen Entwurfswerkzeugen (Xilinx EDK, Altera SoPCBuilder) unterstützt.

Durch die Verwendung dieser Schnittstellen sind Slave-Koprozessoren jedoch an eine spezifische Verbindungsstruktur und ihr Kommunikationsprotokoll gebunden; zusätzlich erfordert die proprietäre Schnittstelle des Hardwarebeschleunigers vom Entwickler der jeweiligen Softwareapplikation technologiespezifische Spezialkenntnisse und verursacht zusätzlichen Aufwand für Entwurf und Test der Verbindungslogik (beispielsweise für Speicherzugriffe oder Interruptbehandlung). Insgesamt behindern proprietäre Schnittstellen die Wiederverwendbarkeit, Portierbarkeit, Skalierbarkeit und, in Konsequenz, die Entwurfsproduktivität insgesamt.

Als Alternative zum repetitiven Entwurf und der Verwendung spezialisierter Schnittstellen für Hardwarebeschleuniger verfolgt mein Ansatz die Erweiterung des Multithreading-Programmiermodells, wie es in der Softwareentwicklung breite Anwendung findet, auf rekonfigurierbare Hardware. Betriebssysteme für eingebettete Systeme wie zum Beispiel VxWorks, RTXC, oder eCos bieten dem Entwickler einen scharf definierten Satz von Objekten und verwandten Diensten, verpackt in standardisierte Programmierschnittstellen (APIs) wie POSIX [IT04]. Unter diesen Objekten finden wir typischerweise *Threads* und *Prozesse* als ausführbare Einheiten, *Semaphore* und verwandte Dienste zur Synchronisation, und *Mailboxen* oder *Message Queues* zur Kommunikation. Diese vom Betriebssystem bereitgestellten Objekte definieren die Interaktion von applikationsspezifischen Ausführungsobjekten untereinander und mit dem Kernel und bilden somit ein Programmiermodell. Zwar ist dieses Modell nicht direkt mit einem formalen Berechnungsmodell vergleichbar, jedoch bietet es dem Entwickler eine gängige Methode um seine Anwendung zu strukturieren.

Durch die Verwendung dieses Programmiermodells für den Hardwareentwurf und die damit einhergehende Modellierung von Hardwarebeschleunigern als autonome Threads gewinnt eine so beschriebene Hardware / Software (HW/SW) Applikation ein hohes Maß an Unabhängigkeit von der verwendeten Zielplattform – die Unterstützung von unterschiedlichen FPGA-Familien, Prozessorarchitekturen oder Betriebssystemen ist implizit gewährleistet, solange eine Implementierung des Multithreading-Programmiermodells für Hardware (in Form der in Abschnitt 4 beschriebenen Laufzeitumgebung) und Software (in Form einer standardisierten API wie POSIX) gegeben ist. Gleichzeitig ermöglicht die Partitionierung einer Applikation in Threads und die explizite Kommunikation und Synchronisation über Betriebssystemobjekte die effiziente Nutzung von Plattform-FPGAs als parallele Ausführungsumgebung.

3.1 Entwurf von Hardware-Threads

Die Ausführung von Software-Threads folgt einer strikt sequentiellen Semantik. Um einen Betriebssystemdienst zu nutzen, muss ein Software-Thread lediglich die korrespondierende Funktion innerhalb einer vom Betriebssystem bereitgestellten Bibliothek aufrufen. Hardware-Threads dagegen sind inhärent parallel – sie folgen typischerweise nicht einem einzelnen Kontrollfluss und haben somit keine Möglichkeit, strukturiert Betriebssystemfunktionen aufzurufen. Insbesondere bieten typische Hardwarebeschreibungssprachen wie VHDL keine Mechanismen für *blockierende Funktionsaufrufe*, wie sie für die Nutzung

von Synchronisationsobjekten notwendig sind.

Um dem Nutzer ein möglichst einheitliches Programmiermodell zur Verfügung zu stellen, verfolgt ReconOS den folgenden Ansatz: ein einzelner, speziell konstruierter Zustandsautomat initiiert und verwaltet alle Betriebssysteminteraktionen des Hardware-Threads. Zu diesem Zweck steht eine Bibliothek mit Betriebssystemfunktionen für VHDL bereit. Diese Bibliothek enthält die Implementierung der Logik für Systemaufrufe verpackt in VHDL-Prozeduren, beispielsweise `reconos_sem_wait()`. Zusammen mit dem *operating system interface* (OSIF), einem separaten synchronisierenden Logikmodul, das die Verbindung zwischen Hardware-Thread und Betriebssystem herstellt, machen diese Prozeduren die Semantik von blockierenden Funktionsaufrufen in VHDL nutzbar. Wie in Abbildung 1(b) angedeutet besteht ein Hardware-Thread also aus mindestens zwei Prozessen: einem synchronisierenden Zustandsautomaten (FSM) und der eigentlichen Thread-Logik (user logic). Die Zustandsübergänge in der FSM werden dabei jeweils vom OSIF kontrolliert; erst, wenn ein zuvor ausgelöster Betriebssystemaufruf beendet ist, wird ein Übergang in den nächsten Zustand zugelassen. Daraus ergibt sich, dass die Kommunikation des Threads mit dem Betriebssystem vollständig sequentiell abläuft, während die Ausführung der eigentlichen Threadfunktionalität in der user logic hochgradig parallel erfolgen kann. Es ist Aufgabe des Programmierers, einen Hardware-Thread in (mehrere) user logic Module und eine einzelne FSM aufzuteilen. Abgesehen von der erhöhten Komplexität eines Hardwareentwurfsprozesses unterscheidet sich dieser Vorgang nicht wesentlich von der Programmierung eines Software-Threads.

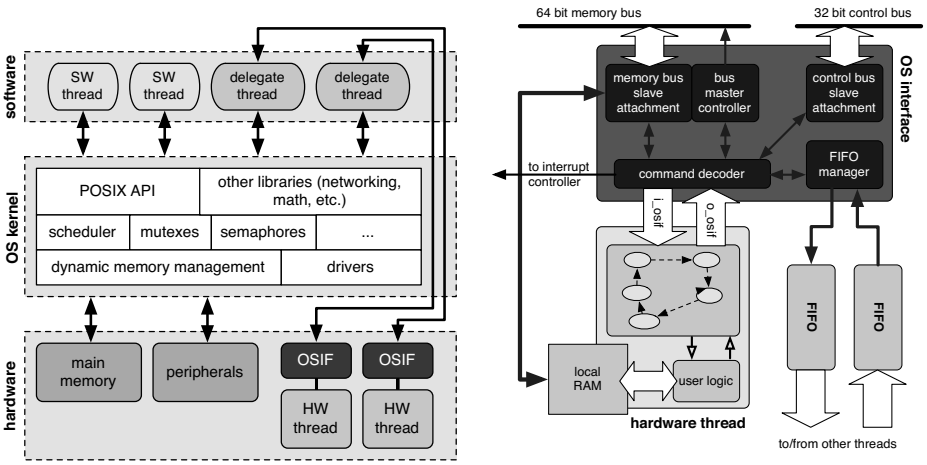
4 ReconOS Laufzeitumgebung

Um ein Höchstmaß an Flexibilität zu gewährleisten, setzt das ReconOS-Laufzeitsystem auf bestehenden Betriebssystemkernels wie eCos oder Linux auf und stellt deren Implementierungen der benötigten Betriebssystemdienste für Hardware-Threads zur Verfügung. Abbildung 1(a) zeigt ein typisches ReconOS-System. Software-Threads und der Betriebssystemkernel werden auf einem zentralen Mikroprozessor (CPU) ausgeführt, während Hardware-Threads für die Logikressourcen des FPGAs synthetisiert werden.

Das Betriebssystem-Interface (OSIF, Abb. 1(b)) bildet eine einheitliche Schnittstelle zwischen Hardware-Threads und Kernel. Es übernimmt dabei drei primäre Aufgaben:

Überwachung und Kontrolle von Hardware Threads. Wie in Abschnitt 3.1 beschrieben kontrolliert ein einzelner Zustandsautomat (FSM) des Hardware-Threads seine Interaktion mit dem Betriebssystem. Ein Threadentwickler kann innerhalb dieses Zustandsautomaten Funktionen wie `semaphore_post()` oder `thread_exit()` verwenden. Diese Funktionen kommunizieren mit dem OSIF über Signale, welches die entsprechenden Funktionen auslöst und gleichzeitig die weitere Ausführung der FSM des Threads blockiert. Zusätzlich überwacht das OSIF den Status des Hardware-Threads und spielt eine wesentliche Rolle bei seiner Initialisierung, Unterbrechung und Rekonfiguration.

Weiterleitung von Betriebssystemaufrufen. Während einige Betriebssystemfunktionen (wie beispielsweise Speicherzugriffe) direkt vom OSIF übernommen werden können, ist ein Großteil der verfügbaren Dienste im Kernel auf der CPU implementiert. Ent-



(a) ReconOS-Laufzeitsystem (b) OSIF mit Hardware-Thread

Abbildung 1: ReconOS Architektur und Betriebssystem-Interface.

sprechende Aufrufe müssen daher vom OSIF weitergeleitet werden. Beim Aufruf einer solchen Funktion wird zunächst ein Interrupt ausgelöst, der an einen sogenannten *delegate thread* (s. unten) auf der CPU weitergeleitet wird. Dieser Software-Thread fragt dann die Parameter des entsprechenden Aufrufs vom OSIF ab und führt den Betriebssystemaufruf anstelle des Hardware-Threads direkt im Betriebssystem aus. Eventuelle Rückgabewerte der Funktion werden anschließend an den Hardware-Thread zurückgegeben, bevor dieser seine Ausführung fortsetzt.

Routing von Datenkommunikation. Da die Weiterleitung von Betriebssystemaufrufen über Interrupts einen messbaren Overhead verursacht, werden Kommunikationsanfragen, die einen hohen Datendurchsatz verlangen, direkt in Hardware ohne Umweg über die CPU verarbeitet. Dazu gehören sowohl Speicherzugriffe als auch sog. Hardware-FIFOs, welche zur direkten Kommunikation zwischen Hardware-Threads eingesetzt werden und sich transparent ins Programmiermodell integrieren. Für Speicherzugriffe in komplexeren Betriebssystemen wie Linux, welche eine virtuelle Speicherverwaltung einsetzen, bietet das OSIF einen Hardware-Thread-lokalen translation-lookaside-buffer (TLB), der unter Betriebssystemkontrolle steht und selbständig Adressumsetzungen anhand der im Systemspeicher abgelegten Seitentabellen vornehmen kann.

Neben der Bereitstellung von Betriebssystemfunktionen für Hardware-Threads erweitert das ReconOS-Laufzeitsystem auch die Software-API um Funktionen zur Erstellung und Terminierung von Hardware-Threads. Dabei können sowohl statisch konfigurierte Threads als Teil der festen FPGA-Logik als auch als partielle Bitströme bereitstehende dynamisch konfigurierbare Threads verwaltet werden (s. Abschnitt 5). Diese `rthread_create()`-Aufrufe beinhalten zusätzlich Mechanismen zur Zuordnung und Verwaltung von dynamisch erzeugten Betriebssystemobjekten wie Mutexes oder Message-Queues zu ihren Hardware-Threads.

Delegate Threads Jedem Hardware-Thread auf dem FPGA steht ein Software-Thread, der sogenannte *delegate thread*, zur Seite, um für ihn Betriebssystemanfragen an den Kernel weiterzuleiten (s. Abbildung 1(a)). Dadurch, dass der delegate ein regulärer Software-Thread unter Kontrolle des Betriebssystem-Schedulers ist, bietet diese Vorgehensweise ein Höchstmaß an Flexibilität und Portierbarkeit – so erscheint ein Hardware-Thread gegenüber allen anderen Threads und dem Betriebssystemkernel als normaler Software-Thread. Dies erlaubt neben der Verwendung verschiedener Hostbetriebssysteme und assoziierter Software-Module eine stark vereinfachte Entwurfsraumexploration in Bezug auf die Hardware / Softwarepartitionierung des Gesamtsystems; ist die Applikation erst einmal in mehrere (Software-)Threads aufgeteilt, kann ein einzelner Thread einfach zwischen CPU und FPGA wechseln, ohne dass Änderungen am übrigen System nötig wären.

Hostbetriebssysteme Die transparente Integration von Hardware-Threads über delegates in das Threadmanagement des Betriebssystems ermöglicht eine Portierung des gesamten Laufzeitsystems (und damit auch des Programmiermodells) auf verschiedene Hostbetriebssysteme, um unterschiedlichen Anforderungen an die Applikation und ggf. verwendete existierende Softwarebibliotheken erfüllen zu können. Im Rahmen dieser Arbeit wurde ReconOS sowohl auf das kompakte Echtzeitbetriebssystem eCos als auch auf das weit verbreitete und vielseitige Betriebssystem Linux portiert.

Während eCos allen laufenden Software-Threads uneingeschränkten und direkten Zugriff auf die Hardware ermöglicht, unterscheidet Linux zwischen User- und Kernel-Mode mit jeweils unterschiedlichen Ausführungsprivilegien. Da die delegate threads von ReconOS als reguläre Software-Threads modelliert sind und somit nur User-Mode-Rechte genießen, unterscheiden sich die Integrationskonzepte für ReconOS/eCos und ReconOS/Linux. Der grundlegende Prozess, wie oben beschrieben, ist dabei gleich; für die Interruptverwaltung und den Hardwarezugriff des delegates unter Linux ist jedoch ein Umweg über einen Kerneltreiber nötig. Dieser übernimmt gleichzeitig die Verwaltung der Hardware-Thread-TLBs für eine transparente Umsetzung zwischen virtuellen und physikalischen Adressen.

5 Hardware Multitasking

Durch die explizite Partitionierung einer Applikation in separate, über explizite und standardisierte Schnittstellen kommunizierende Threads und die Fähigkeit moderner FPGAs zur partiellen Rekonfiguration ergibt sich die interessante Möglichkeit, Hardware-Threads zur Laufzeit auszutauschen, um so die rekonfigurierbare Fläche des FPGAs effizient zu nutzen. Das Umschalten zwischen zwei Threads kann dabei *präemptiv*, d.h. zu jedem beliebigen Zeitpunkt, *nicht-präemptiv*, d.h. nur nach Beendigung eines Threads, oder *kooperativ*, also unter Kontrolle der beteiligten Threads, erfolgen. Im Gegensatz zu Mikroprozessoren, welche den aktuellen Zustand eines Threads durch Speichern ihrer Registerinhalte jederzeit sichern können, gestaltet sich die Unterbrechung von Hardware-Threads komplizierter, da nicht explizit bekannt ist, in welchen auf dem FPGA verteilten Speicherelementen sich der aktuelle Zustand des Threads befindet. Auslesetechniken wie Bitstream-Readback [KP05] oder Scan-Chains erfordern einen hohen Aufwand in Fläche oder Unterbrechungszeit. Wird jedoch die Kontrolle des Zeitpunkts der Unterbrechung dem jeweiligen Thread überlassen, kann die Größe des zu sichernden Zustands minimiert

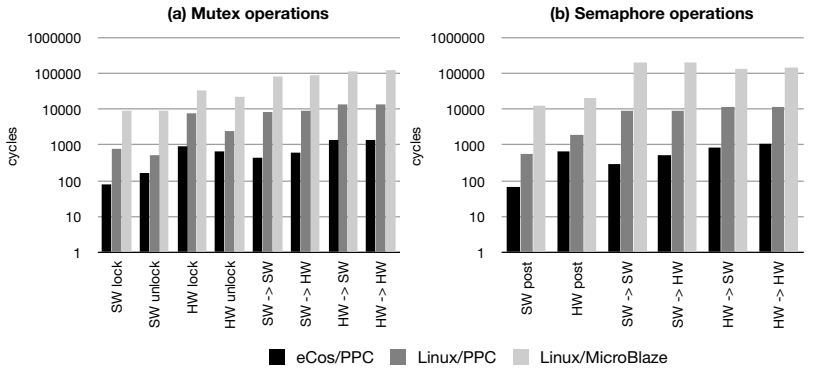


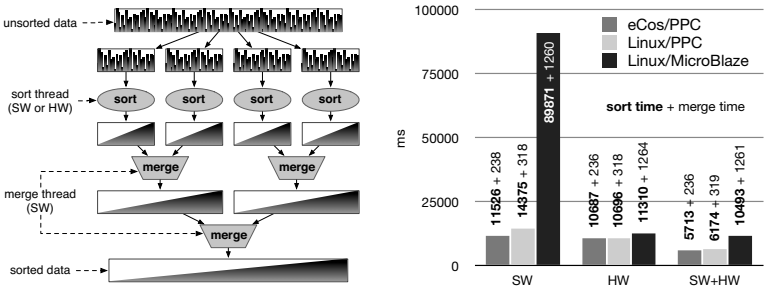
Abbildung 2: Betriebssystemoverheads am Beispiel von Synchronisationoperationen auf Mutex- und Semaphore-Objekten. Angaben in Buszyklen (10 ns).

und die tatsächliche Unterbrechung beispielsweise auf Momente gelegt werden, in denen der Thread selbst auf externe Ressourcen wartet und leicht rekonfiguriert werden kann. Blockierende Betriebssystemaufrufe stellen dafür ideale Kandidaten dar. Zusammen mit diesen Aufrufen kann ein ReconOS-Hardware-Thread dem OSIF (und dadurch dem Betriebssystem) signalisieren, dass eine Unterbrechung möglich ist, und so ein kooperatives Multitasking von Hardware-Threads mit reduziertem Aufwand ermöglichen.

6 Experimentelle Ergebnisse

Die vorgestellten Konzepte wurden in Prototypen auf verschiedenen FPGA-Familien (Virtex-II, Virtex-4), Prozessorarchitekturen (PowerPC, MicroBlaze) und Hostbetriebssystemen (eCos, Linux) implementiert und evaluiert. Abbildung 2 zeigt (repräsentativ für Thread-initiierte Betriebssystem-Funktionen) Messungen der Ausführungszeiten von Synchronisationsoperationen. Die dabei auftretenden höheren Ausführungszeiten von Betriebsystemaufrufen mit Hardwarebeteiligung fallen insofern kaum ins Gewicht, als dass typischerweise kontroll- und synchronisationslastige Teile einer Anwendung ohnehin in Software ausgeführt werden, während die rekonfigurierbaren Ressourcen einer CPU/FPGA-Plattform zur Ausnutzung der Datenparallelität genutzt werden und mit vergleichsweise weniger Synchronisationsaufrufen auskommen.

Die Anwendbarkeit des Multithreading-Ansatzes für Hardware wurde durch mehrere Fallstudien demonstriert, darunter Anwendungen zur Bildverarbeitung [LP09], Kryptographie, adaptiven Hardwarebeschleunigung von Netzwerkprotokollen [KPL+10] und statistischer Objektverfolgung [HLP09]. Zur Veranschaulichung zeigt Abbildung 3(a) eine Anwendung zum Sortieren von Zahlenfolgen mit mehreren Threads, von denen einige (*sort*) in Software und/oder Hardware ausgeführt werden können. Wie die Ergebnisse in Abb. 3(b) zeigen, erlaubt die Integration von Hardware-Threads eine erhebliche Beschleunigung der Anwendung auch bei niedriger Rechenleistung der CPU; dies wiederum belegt, dass bei entsprechender Nutzung der Synchronisationsmechanismen zwischen Hard- und Software-Threads die Overheads der Betriebssystemaufrufe nicht signifikant ausfallen.



(a) Segmentiertes HW/SW Merge-Sort (b) Laufzeiten verschiedener Zielplattformen

Abbildung 3: Fallstudie zur HW/SW Partitionierung.

7 Zusammenfassung und Ausblick

Die vorgestellte Methode zur Programmierung von CPU/FPGA-Architekturen durch Integration von Hardwarebeschleunigern als autonome Hardware-Threads in ein bestehendes Softwarebetriebssystem eröffnet neue Möglichkeiten für den portablen Systementwurf und die flexible Entwurfsraumexploration hybrider HW/SW-Systeme. Durch die konsequente Verwendung eines einheitlichen Programmiermodells für Hard- und Software wird die Entwurfsproduktivität für rekonfigurierbare SoCs auf ein der Softwareentwicklung vergleichbares Maß angehoben. Zugleich ermöglicht der neuartige Hardware-Multithreading-Ansatz erstmals die transparente Nutzung der dynamischen Laufzeitrekonfiguration von FPGAs in einem etablierten Programmiermodell und vereinfacht so den Entwurf von adaptiven Hardware / Software-Systemen durch einen durchgängigen Entwurfsfluss.

Das ReconOS-Programmier- und Laufzeitsystem erlaubt die Bearbeitung von wissenschaftlichen Fragestellungen – wie dem Scheduling von Tasks auf hybriden Systemen, der Hardware / Software Entwurfsraumexploration oder der transparenten Migration von Laufzeitkomponenten von der Applikation durch verschiedene Schichten des Betriebssystems bis hin zur Hardware – auf einem praktisch relevanten Abstraktionsgrad. ReconOS ist frei verfügbar [rec] und wird bereits zur Erstellung von Prototypen und Applikationen hybrider Systeme, beispielsweise im Bereich adaptiver Netzwerke und eingebetteter Viedoüberwachungssysteme, eingesetzt.

Literatur

[Bre96] Gordon J. Brebner. A Virtual Hardware Operating System for the Xilinx XC6200. In *6th International Workshop on Field-Programmable Logic and Applications (FPL)*, Seiten 327–336, London, UK, 1996. Springer-Verlag.

[BWHC06] Neil W. Bergmann, John A. Williams, Jie Han und Yi Chen. A Process Model for Hardware Modules in Reconfigurable System-on-Chip. In *19th International Conference on Architecture of Computing Systems, Dynamically Reconfigurable Systems Workshop*, Jgg. 81, Seiten 205–214, Mar 2006.

[HLP09] Markus Happe, Enno Lübbers und Marco Platzner. An Adaptive Sequential Monte

Carlo Framework with Runtime HW/SW Repartitioning. In *IEEE International Conference on Field Programmable Technology (FPT'09)*. IEEE, Dec 2009.

[IT04] IEEE und The Open Group. The Open Group Base Specifications Issue 6, IEEE Std. 1003.1, 2004 Edition, 2004.

[KMK07] Krzysztof Kosciuszkiewicz, Fearghal Morgan und Krzysztof Kepa. Run-Time Management of Reconfigurable Hardware Tasks Using Embedded Linux. In *IEEE International Conference on Field-Programmable Technology (FPT)*, Seiten 209–215. IEEE, 2007.

[KP05] Heiko Kalte und Mario Porrmann. Context Saving and Restoring for Multitasking in Reconfigurable Systems. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, Seiten 223–228. IEEE, 2005.

[KPL+10] Ariane Keller, Bernhard Plattner, Enno Lübbers, Christian Plessl und Marco Platzner. Reconfigurable Nodes for Future Networks. In *Proceedings of the Third International Workshop on the Network of the Future (FutureNet III)*. IEEE, 2010.

[LP09] Enno Lübbers und Marco Platzner. ReconOS: Multithreaded Programming for Reconfigurable Computers. *ACM Transactions on Embedded Computing Systems (TECS)*, 9(1), October 2009.

[Lüb10] Enno Lübbers. *Multithreaded Programming and Execution Models for Reconfigurable Hardware*. Dissertation, University of Paderborn, March 2010.

[rec] http://www.reconos.de.

[SB06] Hayden Kwok-Hay So und Robert W. Brodersen. Improving Usability of FPGA-based Reconfigurable Computers through Operating System Support. In *16th IEEE International Conference on Field Programmable Logic and Applications (FPL)*, Seiten 349–354. IEEE, 2006.

[SWP04] Christoph Steiger, Herbert Walder und Marco Platzner. Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-time Tasks. *IEEE Transactions on Computers*, 53(11):1392–1407, Nov 2004.

[TFS00] J. Teich, S. Fekete und J. Schepers. Optimization of Dynamic Hardware Reconfigurations. *The Journal of Supercomputing*, 19(1):57–75, May 2000.

Enno Lübbers wurde am 19.05.1980 in Hannover geboren. Er studierte Informations-Systemtechnik (Computer and Communication Systems Engineering) mit den Vertiefungsrichtungen eingebettete Systeme, VLSI und Chip Design, Kommunikationssysteme und Robotik an der Technischen Universität Braunschweig. Im Anschluss an sein Diplom wechselte er als wissenschaftlicher Mitarbeiter an die Universität Paderborn, wo er in der Arbeitsgruppe Technische Informatik von Prof. Marco Platzner seine Promotion zum vorliegenden Thema abschloss. Während seiner Promotionszeit verbrachte er Forschungsaufenthalte am Information and Telecommunication Technology Center der University of Kansas und an der University of Arkansas. Zur Zeit arbeitet Enno Lübbers als Forscher für eingebettete Software und Rekonfigurierbare Avionik bei EADS Innovation Works in München.