Hardware/Software Partitioning of Operating Systems

The δ Hardware/Software RTOS Generation Framework for SoC

Vincent J. Mooney III
http://codesign.ece.gatech.edu

Assistant Professor, School of Electrical and Computer Engineering
Adjunct Assistant Professor, College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA

5 March 2003 presentation at DATE

©Vincent J. Mooney III, 2002
Outline

• Vision: Hardware/Software Real-Time Operating System

• Custom RTOS Hardware IP Components
  • System-on-a-Chip Lock Cache (SoCLC)
  • SoC Dynamic Memory Management Unit (SoCDMMU)

• The δ Hardware/Software RTOS Generation Framework
  • Comparison with the RTU Hardware RTOS

• Conclusion
Vision: Dynamic Software/Hardware RTOS Design

Key to System-on-a-Chip architecture optimization and customization
Recent SoC Example: Broadcom BCM1400

- Four Processor Cores
  - MIPS64
  - 1GHz
  - 8-way 1MB Shared L2
- On-chip ZBbus
  - maintains coherency
  - proprietary
- Off-chip HT/ SPI-4 19Gb/s

MIPS64 Core0  
MIPS64 Core1  
MIPS64 Core2  
MIPS64 Core3  
L2 Shared  
Mem Arbiter

Memory Bridge  
Packet DMA  
SoC Interfaces

ZBbus: 128Gb/s @ 1GHz

Motivational Example: Home 2005

Programmable PDA

Projectors

Central Storage Unit (e.g., PC)

SoC Device

wireless link

wired link

5 March 2003 presentation at DATE

©Vincent J. Mooney III, 2002

HW/SW RTOS Project
Analogy

- Microprocessor design
  - Compiler
  - Computer architecture

- SoC design
  - Dynamic hw/sw RTOS
  - SoC architecture
Building Blocks

• SoC Programming Model
  – multi-threading, shared mem., message passing, control-data flow graph
• SoC Programming Environment
  – δ Hardware/Software RTOS
• Microprocessor Programming Model
  – C/C++/Java/other serial language
• Microprocessor Programming Environment
  – gcc, various
Approach

- δ HW/SW RTOS made up of library components
- Library component = predefined C code, assembly code or HDL code
- Similar to existing RTOS’s, except for the HDL code
  - ex.: SoC Lock Cache in hardware [1]
- RTOS HDL code can be automatically generated by a custom “IP Generator”
  - ex.: PARLAK SoC Lock Cache generator, poster 5P.11 here in DATE 2003
The δ Hardware/Software RTOS Generation Framework

A Framework for Automatic Generation of Configuration Files for a Custom Hardware/Software RTOS

<table>
<thead>
<tr>
<th>PE selection</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE1: PowerPC</td>
<td>Number of cpus: 4</td>
</tr>
<tr>
<td>PE2: PowerPC</td>
<td>Number of tasks: 40</td>
</tr>
<tr>
<td>PE3: PowerPC</td>
<td></td>
</tr>
<tr>
<td>PE4: PowerPC</td>
<td></td>
</tr>
</tbody>
</table>

Specialized Software Components
- Deadlock Detection
- Memory Management

Hardware Components
- SoCLC
- SoCDDU
- SoCDMMU
- RTU

IPC methods
- Semaphore
- Event
- MailBox
- Queue
- Mutual
- Allocation

GUI tool

User Input

Application

Compile Stage for each system

Executable HW file for each

Executable SW file for each

Simulation in Seamless CVE

VCS

XRAY

Software RTOS library

Hardware RTOS library

Base Architecture library

RTOS1

SW RTOS w/ sem

RTOS2

SW RTOS + SoCLC

RTOS3

SW RTOS w/ dyn. memory mngmnt + SoCDMMU

RTOS4

SW RTOS + SoCLC + SoCDMMU

RTOS5

RTU
Outline

• Vision: Hardware/Software Real-Time Operating System
• Custom RTOS Hardware IP Components
  • System-on-a-Chip Lock Cache (SoCLC)
  • SoC Dynamic Memory Management Unit (SoCDMMU)
• The δ Hardware/Software RTOS Generation Framework
  • Comparison with the RTU Hardware RTOS
• Conclusion
SoC Lock Cache

- A hardware mechanism that resolves the critical section (CS) interactions among PEs
- Lock variables are moved into a separate “lock cache” outside of the memory
- Improves the performance criteria in terms of lock latency, lock delay and bandwidth consumption
Software/Hardware Architecture

- Multiple application tasks
- Atalanta-RTOS
- Four MPC750s
- SoCLC provides lock synchronization among PEs

Application Software (Tasks)

Atalanta-RTOS  Extension

Software

Hardware

MPC750A  MPC750B  MPC750C  MPC750D

Memory

SoCLC  Arbitration Logic

5 March 2003 presentation at DATE

©Vincent J. Mooney III, 2002
Example: Database transaction application [1]

### Experimental Result

- **Comparison with database application example [2]**
  - RTOS1 with semaphores and spin-locks
  - RTOS2 with SoCLC, no SW semaphores or spin-locks

<table>
<thead>
<tr>
<th>(clock cycles)</th>
<th>* Without SoCLC</th>
<th>With SoCLC</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lock Latency</strong></td>
<td>1200</td>
<td>908</td>
<td>1.32x</td>
</tr>
<tr>
<td><strong>Lock Delay</strong></td>
<td>47264</td>
<td>23590</td>
<td>2.00x</td>
</tr>
<tr>
<td><strong>Execution Time</strong></td>
<td>36.9M</td>
<td>29M</td>
<td>1.27x</td>
</tr>
</tbody>
</table>

* Semaphores for long critical sections (CSes) and spin-locks for short CSes are used instead of SoCLC.

Outline

- Vision: Hardware/Software Real-Time Operating System
- Custom RTOS Hardware IP Components
  - System-on-a-Chip Lock Cache (SoCLC)
  - SoC Dynamic Memory Management Unit (SoCDMMU)
- The δ Hardware/Software RTOS Generation Framework
  - Comparison with the RTU Hardware RTOS
- Conclusion
**SoCDMMU: Move L2 Memory Allocation to One (Hardware) Unit**

=> "Undistribute" L2 Memory Allocation Algorithm

![Diagram showing memory allocation and distribution]

- **PE_1**, **PE_2**, ..., **PE_n**
- **Cache**
- **DMMU**
- **Global Memory**
- **Memory Bus**
- **command/status**
- **RD**
- **WR**
- **Busy**
Levels of Memory Management

- The SoCDMMU dynamically allocates the global on-a-chip memory among the PE’s (Level 2).

- Each PE handles the local dynamic memory allocation among the processes/threads (Level 1).
Execution Times

- Synthesized using the TSMC 0.25u .
- Clock Speed: 200MHz.
- Size: ~7500 gates per PE (not including Memory Elements: Allocation Table and Address Converter).

<table>
<thead>
<tr>
<th>Command</th>
<th>Number of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_alloc_ex</td>
<td>4</td>
</tr>
<tr>
<td>G_alloc_rw</td>
<td>4</td>
</tr>
<tr>
<td>G_alloc_ro</td>
<td>3</td>
</tr>
<tr>
<td>G_dealloc</td>
<td>4</td>
</tr>
<tr>
<td>4-Processors WCET</td>
<td>16</td>
</tr>
</tbody>
</table>
Atalanta Support for the SoCDMMU

Objectives

• Port SoCDMMU hardware to an RTOS (ease of use)
• Atalanta is an open-source RTOS written at Georgia Tech
  – similar to uC-OS II or VRTXoc
• Add Dynamic Memory Management to Atalanta
• Use the same Memory Management API Functions
• Keep the Memory Management Deterministic
Comparison to a Fully Shared-Memory Multiprocessor System

- Global memory of 16MB; L1 $ is 64 kB.
- Each ARM processor runs at 200MHz.
- Accessing the Global Memory costs 5 cycles.
- A handheld device that utilizes this SoC can be used for OFDM communication as well as other applications (MPEG2 video player).
- Initially the device runs an MPEG2 video player. When the device detects an incoming signal it switches to the OFDM receiver. The switching time (which includes the time for memory management) should be short or the device might lose the incoming message.
Area Estimate of The SoC

- ARM9TDMI Core: 112k transistors
- L1 $ (128KB: 64KB I$ + 64KB D$): ~6.5M* transistors
- SoC DMA (w/o the memory elements -- Allocation Table and Address Converters): ~30k transistors.
- Allocation Table: ~30k transistors
- Address Converter: ~60k* transistors
- Total-L1-L2: (4*112 + 30 + 30 + 4*60)=748k trans. =~.75M
- Total-L2: ~.75M+(4*~6.5M) = ~26.75M transistors
- L2 (Global Memory)=~16M * 8 = ~128M transistors

* Using dual-port 6T SRAM cells.
Comparison to a Fully Shared-Memory Multiprocessor System

- Sequence of Memory Allocations Required

<table>
<thead>
<tr>
<th>MPEG-2 Player</th>
<th>OFDM Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Kbytes</td>
<td>34 Kbytes</td>
</tr>
<tr>
<td>500 Kbytes</td>
<td>32 Kbytes</td>
</tr>
<tr>
<td>5 Kbytes</td>
<td>1 Kbytes</td>
</tr>
<tr>
<td>1500 Kbytes</td>
<td>1.5 Kbytes</td>
</tr>
<tr>
<td>1.5 Kbytes</td>
<td>32 Kbytes</td>
</tr>
<tr>
<td>0.5 Kbytes</td>
<td>8 Kbytes</td>
</tr>
<tr>
<td></td>
<td>32 Kbytes</td>
</tr>
</tbody>
</table>
Comparison to a Fully Shared-Memory Multiprocessor System

Memory Management Execution time during transition from the MPEG2 player to the OFDM Receiver

<table>
<thead>
<tr>
<th></th>
<th>Using the SoCDMMU</th>
<th>Using ARM SDT malloc() and free()</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Case</td>
<td>281 cycles</td>
<td>1240 cycles</td>
<td>4.4X*</td>
</tr>
<tr>
<td>Worst Case</td>
<td>1244 cycles</td>
<td>4851 cycles</td>
<td>3.9X</td>
</tr>
</tbody>
</table>

*Note this number exceeds 10X when using GCC libc memory management functions instead of ARM SDT2.5 embedded malloc() and free() functions.

=> For this 154.75 Million transistor chip, 30K + 30K + 240K = 300K (0.19% of 154.75M), or, if memory can be allocated by the SoCDMMU, 30K (0.02% of 154.75M) yields a 4-10X speedup in memory allocation.
Outline

• Vision: Hardware/Software Real-Time Operating System

• Custom RTOS Hardware IP Components
  • System-on-a-Chip Lock Cache (SoCLC)
  • SoC Dynamic Memory Management Unit (SoCDMMU)

• The δ Hardware/Software RTOS Generation Framework
  • Comparison with the RTU Hardware RTOS

• Conclusion
δ Hardware/Software RTOS Generation Framework

and current simulation platform

---

User Input

GUI Tool

Base Architecture Library

Top.v HW RTOS

User.h SW RTOS

Makefile

Application

HW Compile

SW Compile

Compiled Hardware Description

Executable HW

Executable SW

Modelsim or VCS

Simulation in Seamless CVE

XRAY

Compiled SW

User

Input

Hardware RTOS Library

Software RTOS Library

Result and Feedback

---

©Vincent J. Mooney III, 2002
δ Hardware/Software RTOS Generation Framework Goals

- To help the user examine which configuration is most suitable for the user’s specific applications.
- To help the user explore the RTOS design space before chip fabrication as well as after chip fabrication (in which case reconfigurable logic must be available on the chip).
- To help the user examine different System-on-a-Chip (SoC) architectures subject to a custom RTOS.
Motivation (1/2)

- HW/SW RTOS partitioning approach
- Three previous innovations in HW/SW RTOS components
  - SoCLC: System-on-a-Chip Lock Cache
  - SoCDMMU: System-on-a-Chip Dynamic Memory Management Unit
  - SoCDDU: System-on-a-Chip Deadlock Detection Unit
- RTU Hardware RTOS
Constraints about using three previous HW/SW RTOS innovations

- Perhaps not enough chip space for all three of them
- All of them may not be necessary

⇒ The δ framework

- Enables automatic generation of different mixes of the three previous innovations for different versions of a HW/SW RTOS
- Enables selection of the RTU hardware RTOS
- Can be generalized to instantiate additional HW or SW RTOS components
Our RTOS in Post-Fabrication Scenario

- Application(s) run on the SoC using standard RTOS APIs

- Atalanta software RTOS
  - A multiprocessor SoC RTOS
    - The RTOS and device drivers are loaded into the L2 cache memory
  - All Processing Elements (PEs)
    - share the kernel code and data structures

- Hardware RTOS components are downloaded into the reconfigurable logic
Experimental Setup

- Six custom RTOSes
  - With semaphores and spin-locks, no HW components in the RTOS
  - With SoCLC, no SW IPCs
  - With dynamic memory management software, no HW RTOS components
  - With SoCDMMU, no SW IPCs
  - With SoCLC and SoCDMMU
  - With RTU
- Each with the *Base* architecture
- Each with application(s)
- Each executable in Seamless CVE
  - 4 MPC750 processors
  - Reconfigurable logic
  - Single bus
RTU Hardware RTOS

- An RTOS in hardware
  - Real-Time Unit (RTU)
    - scheduling
    - IPC
    - dynamic task creation
    - timers
  - Custom hw => upper bound on # tasks
  - Reconfigurable hw => can alter max. # tasks, max. # priorities
  - Prof. Lennart Lindh, Mälardalens U., Västerås, Sweden
  - RealFast, www.realfast.se
Methodology

- An SoC architecture with the RTU Hardware RTOS

Diagram:
- MPC755-1
- MPC755-2
- MPC755-3
- RTU in Reconfig. Logic
- Arbiter, Intr. Controller, Clock
- Memory Controller and Memory
Methodology

- An SoC architecture with a hardware/software RTOS

```
MPC755-1  MPC755-2  MPC755-3
L1       L1       L1

SoCLC in Reconfig. Logic

Bus Arbiter, Intr. Controller, Clock

Memory (Atalanta RTOS)
```
Methodology

- δ Framework
  - GUI

Specific SW

- HW RTOS component

IPC module linking method

Number of CPUs in system

Specilized SW

RTOS component

Hardware Components

IPC method

- Semaphore
- Event
- MailBox
- Queue
- Mutual
- Allocation
Implementation

- Verilog top file generation example
  - Start with RTU description
  - Generate instantiation code
    - Multiple instantiations of same unit if needed (e.g., PEs)
  - Add wires and initial statements

```
Verilog top file generation example

• Start with RTU description
• Generate instantiation code
  ✓ multiple instantiations of same unit if needed (e.g., PEs)
• Add wires and initial statements

Start with RTU description:

- Generate instantiation code
  - Multiple instantiations of same unit if needed (e.g., PEs)
- Add wires and initial statements

Verilog top file generation example

Generate code

• Generate instantiation code
  - Multiple instantiations of same unit if needed (e.g., PEs)
• Add wires and initial statements

Add wires and initial statements

• Add wires and initial statements
  - Multiple instantiations of same unit if needed (e.g., PEs)
- Add wires and initial statements

Add wires and initial states

- Add wires and initial statements
  - Multiple instantiations of same unit if needed (e.g., PEs)
- Add wires and initial statements
```
**Experimental Results (1/3)**

- **Comparison**
  - A system with RTU hardware RTOS
  - A system with SoCLC hardware and software RTOS
  - A system with pure software RTOS

<table>
<thead>
<tr>
<th>Total Execution Time</th>
<th>Pure SW *</th>
<th>With SoCLC</th>
<th>With RTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 tasks: (in cycles)</td>
<td>100398</td>
<td>71365</td>
<td>67038</td>
</tr>
<tr>
<td>Reduction</td>
<td>0%</td>
<td>29%</td>
<td>33%</td>
</tr>
<tr>
<td>30 tasks: (in cycles)</td>
<td>379440</td>
<td>317916</td>
<td>279480</td>
</tr>
<tr>
<td>Reduction</td>
<td>0%</td>
<td>16%</td>
<td>26%</td>
</tr>
</tbody>
</table>

* A semaphore is used in pure software and a hardware mechanism is used in SoCLC and RTU.*
### Experimental Results (2/3)

- The number of interactions

<table>
<thead>
<tr>
<th>Times</th>
<th>6 tasks</th>
<th>30 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of semaphore interactions</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td>Number of context switches</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>Number of short locks</td>
<td>10</td>
<td>58</td>
</tr>
</tbody>
</table>
Experimental Results (4/4)

- The average number of cycles spent on communication, context switch and computation (6 task case)

<table>
<thead>
<tr>
<th></th>
<th>Pure SW</th>
<th>With SoCLC</th>
<th>With RTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>18944</td>
<td>3730</td>
<td>2075</td>
</tr>
<tr>
<td>context switch</td>
<td>3218</td>
<td>3231</td>
<td>2835</td>
</tr>
<tr>
<td>computation</td>
<td>8523</td>
<td>8577</td>
<td>8421</td>
</tr>
</tbody>
</table>
# Hardware Area

<table>
<thead>
<tr>
<th>Total area</th>
<th>SoCLC (64 short CS locks + 64 long CS locks)</th>
<th>RTU for 3 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSMC 0.25(\mu)m library from LEDA</td>
<td>7435 gates</td>
<td>About 2500000 gates</td>
</tr>
</tbody>
</table>

©Vincent J. Mooney III, 2002
Conclusion

- A framework for automatic generation of a custom HW/SW RTOS
- Experimental results showing
  - a multiprocessor SoC that utilizes the SoCDMMU has a 4X overall speedup of the application transition time over fully shared memory that does not utilize the SoCDMMU
  - speedups with the SoCLC, RTU
  - addition hw RTOS component in references: SoCDDU
- Future work
  - support for heterogeneous processors
  - support for multiple bus systems/structures