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

Jaehwan Lee*
Kyeong Keol Ryu*
Vincent J. Mooney III+
{jahweh, kkryu, mooney}@ece.gatech.edu
http://codesign.ece.gatech.edu

+Assistant Professor, *School of Electrical and Computer Engineering
+Adjunct Assistant Professor, College of Computing
Georgia Institute of Technology

26 June 2002 at ERSA
HW/SW RTOS Project of the HW/SW Codesign Group at GT
Outline

- Introduction
- Goals
- Motivation
- Methodology
- Experimental Results
- Conclusion
Introduction

- Specify custom HW/SW RTOS in a graphical user interface (GUI)
- Generate configuration files used to make a custom RTOS
  - A custom RTOS may contain HW (as well as SW) components
- Compile both hardware and software with an application
- Simulate the system to evaluate the result
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 after chip fabrication as well as before chip fabrication
- To help the user examine different system-on-a-chip (SoC) architectures subject to a custom RTOS
Motivation (1/5)

- 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
System-on-a-Chip 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
- Improving the performance criteria in terms of lock latency, lock delay and bandwidth consumption
Motivation (3/5)

- SoCDMMU: System-on-a-Chip Dynamic Memory Management Unit
  - Provides fast, deterministic and yet dynamic memory management of a global on-chip memory
  - Achieves flexible, efficient memory utilization
  - Provides APIs for applications
Motivation (4/5)

- **SoCDDU: System-on-a-Chip Deadlock Detection Unit**
  - Performs a novel parallel hardware deadlock detection based on implementing deadlock searches on the resource allocation matrix in hardware
  - Provides a very fast deadlock detection at run-time with dedicated hardware performing simple bit-wise boolean operations
  - Reduces deadlock detection time by 99% as compared to software
  - Requires at most $O(2 \times \min(m,n))$ iterations as opposed to $O(m \times n)$ required by all previously reported (sequential) software algorithms
Motivation (5/5)

Constraints about using three previous innovations

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

⇒ Our framework

• Enables automatic generation of different mixes of the three previous innovations for different versions of a HW/SW RTOS
• Can be generalized to instantiate additional HW or SW RTOS components
Methodology (1/2)

- Translates the user choices into a custom RTOS
  - Given the IP library of processors and HW/SW RTOS components
- Generates configuration files to glue together a custom RTOS executable in the Seamless Co-Verification Environment from Mentor Graphics
  - Makefile and User.h for SW link
  - Verilog header file for HW glue
Explores the HW/SW RTOS design space defined by the available HW/SW RTOS components easily.
Our RTOS and Possible Target SoC

- A multiprocessor System-on-a-Chip (*Base* architecture)
- A multiprocessor RTOS
- Application(s) running on the SoC using the RTOS APIs
Our RTOS in Detail

- 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
Selectable RTOS IP components

- **Software (Atalanta RTOS)**
  - Inter-Process Communication (IPC) components (semaphore, queue, event, mailbox, etc)
  - CPU schedulers (priority, round-robin)
  - Memory management module (gmm)
  - Deadlock detection module (ddm)

- **Hardware**
  - SoC Lock Cache for fast IPC (SoCLC)
  - Dynamic Memory Management Unit (SoCDMMU)
  - Deadlock Detection Unit (SoCDDDU)
Implementation (1/8)

- SW module linking method
- HW integration method
- IPC module linking method
- SW may over-ride task size

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

PE selection
- PE1: PowerPC
- PE2: PowerPC
- PE3: PowerPC
- PE4: PowerPC

Miscellaneous
- Number of cpus: 4
- Number of tasks: 4
- Task stack size: 2048

Specialized Software Components
- Deadlock Detection
- Memory Management

Hardware Components
- SoCDDU
- SoCMDMU
- SoCLC

IPC methods
- Semaphore
- Event
- MailBox
- Queue
- Mutex
- Allocate
Implementation (2/8)

with Example Use of GUI Tool

- The user
  - Selects
    - Deadlock detection SW module
    - Semaphores for synchronization
    - SoCLC for critical section
  - Clicks *Generate* button

- The tool
  - Generates
    - Makefile & User.h
    - Verilog header file
1) Software module linking method

- Command-line object inclusion method (well-known)
- Used for the same function but different implementations
- Implemented in Makefile
- Used for linking the deadlock detection SW module in the example
1) Software module linking method (cont’d)

$\texttt{(LD) -o @} \$(OTHER\_OBS) \$(OPT\_OBJ1) \$(OPT\_OBJ2) \$(LIBRARY)$

to

gcc -o ddutest.x [all other objects including ddutest.o] ddm.o atalanta.a
2) Inter-process communication module linking method

- Library function linking method (common)
- Implemented in User.h
- Used for the semaphore function in the example
2) IPC module linking method (cont’d)

Selection flow of IPC methods

GUI Tool

Pre-processing

View included

Generated Configuration

#define semaphores TRUE

Making Stage

user.c user.i user.o

Linking Stage

Library (Atalanta.a)

Semaphore functions

Queue functions

Mailbox functions

Event functions

Other functions

application

ddutest.x

user.c

user.i

user.o

ddutest.c

ddutest.o
3) HW RTOS component integration method

- Novel HW integration method
- Construct a Verilog header file
- Integrate user-selected HW RTOS components into the Base architecture
- Start with an SoCLC architecture description (an example with SoCLC)
Implementation (8/8)

Verilog header file generation example

- Start with SoCLC description
- Extract modules
- Add wires
- Insert the instantiation code for each module
Experimental Setup

- Five custom RTOSes
  - With semaphores and spin-locks, no HW components in the RTOS
  - With SoCLC, no SW IPCs
  - With deadlock detection software, no HW RTOS components
  - With SoCDDU, no SW IPCs
  - With SoCLC and SoCDDU
- Each with the Base architecture
- Each with application(s)
- Each executable in Seamless CVE
  - 4 MPC750 processors
  - Reconfigurable logic
  - Single bus

June 2002 at ERSA
Example 1: Database transaction application [1]

Experimental Results (2/4)

- 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>Lock Latency</td>
<td>1200</td>
<td>908</td>
<td>1.32x</td>
</tr>
<tr>
<td>Lock Delay</td>
<td>47264</td>
<td>23590</td>
<td>2.00x</td>
</tr>
<tr>
<td>Execution Time</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.

Example 2: Interactions between multiple processors and resources [3]

Experimental Results (4/4)

- Comparison with deadlock detection example [4]
  - RTOS3 with a software deadlock detection module, no HW RTOS
  - RTOS4 with SoCDDU

<table>
<thead>
<tr>
<th>Method of Deadlock Detection</th>
<th>Software Algorithm</th>
<th>SoCDDU</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection Time $\Delta$ (clock cycles)</td>
<td>16928</td>
<td>2</td>
<td>8463x</td>
</tr>
<tr>
<td>Execution time up to deadlock detection</td>
<td>61131</td>
<td>44205</td>
<td>1.38x</td>
</tr>
</tbody>
</table>

## Hardware Area

<table>
<thead>
<tr>
<th></th>
<th>SoCLC</th>
<th>SoCDDU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total area in</strong></td>
<td>(For 64 short CS locks + 64 long CS locks)</td>
<td>(For 5 Processors x 5 Resources)</td>
</tr>
<tr>
<td><strong>Semi-custom VLSI</strong></td>
<td>7435 gates using TSMC 0.25(\mu)m standard cell library</td>
<td>364 gates using AMI 0.3(\mu)m standard cell library</td>
</tr>
<tr>
<td><strong>Xilinx</strong></td>
<td># Seq. logic 532</td>
<td>10</td>
</tr>
<tr>
<td>XC4000E 4003EPC84</td>
<td># Other gates 9036</td>
<td>559</td>
</tr>
</tbody>
</table>

*TSMC 0.25\(\mu\)m standard cell library*
Conclusion

- A framework for automatic generation of configuration files for a custom HW/SW RTOS
- A novel HW header file generation methodology
- Experimental results showing
  - the configured systems are correct
- A framework used to explore the RTOS design space.
- Future work
  - support for heterogeneous processors
  - support for multiple bus systems/structures