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

©Vincent J. Mooney III, 2002

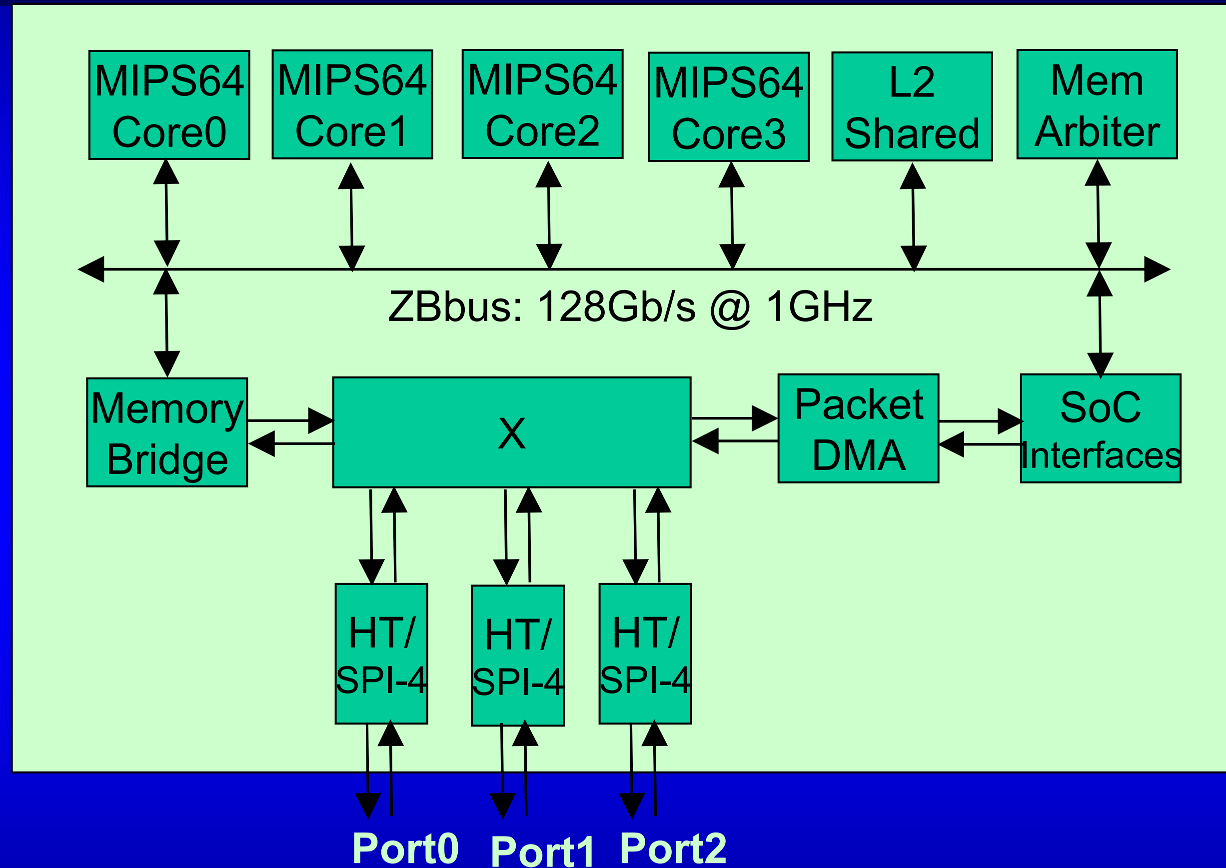
Vision: Dynamic Software/ Hardware RTOS Design

Key to System-on-a-Chip architecture optimization and customization

©Vincent J. Mooney III, 2002

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



[Levy02] M. Levy, "Chip Combines Four 1GHz Cores," *Microprocessor Report*, pp. 12-14, October 2002.

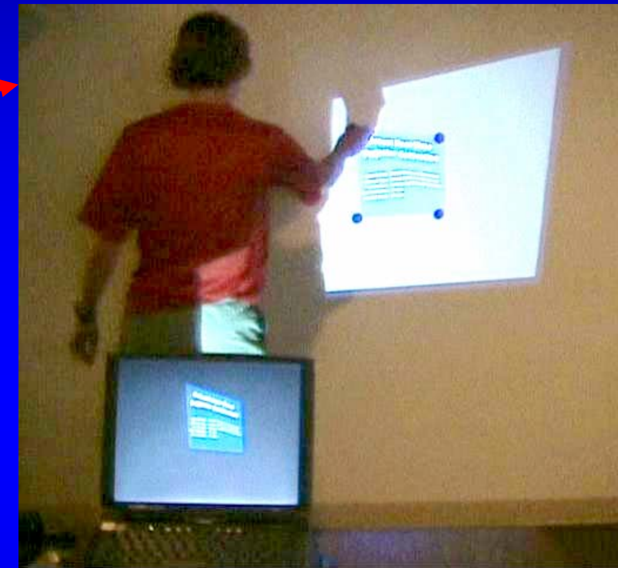
©Vincent J. Mooney III, 2002

Motivational Example: Home 2005

Programmable PDA

Projectors

Projected Light Displays



SoC Device

Central Storage Unit (e.g., PC)

wireless link

wired link

Analogy

- Microprocessor design
  - Compiler
  - Computer architecture
- SoC design
  - Dynamic hw/sw RTOS
  - SoC architecture

©Vincent J. Mooney III, 2002

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

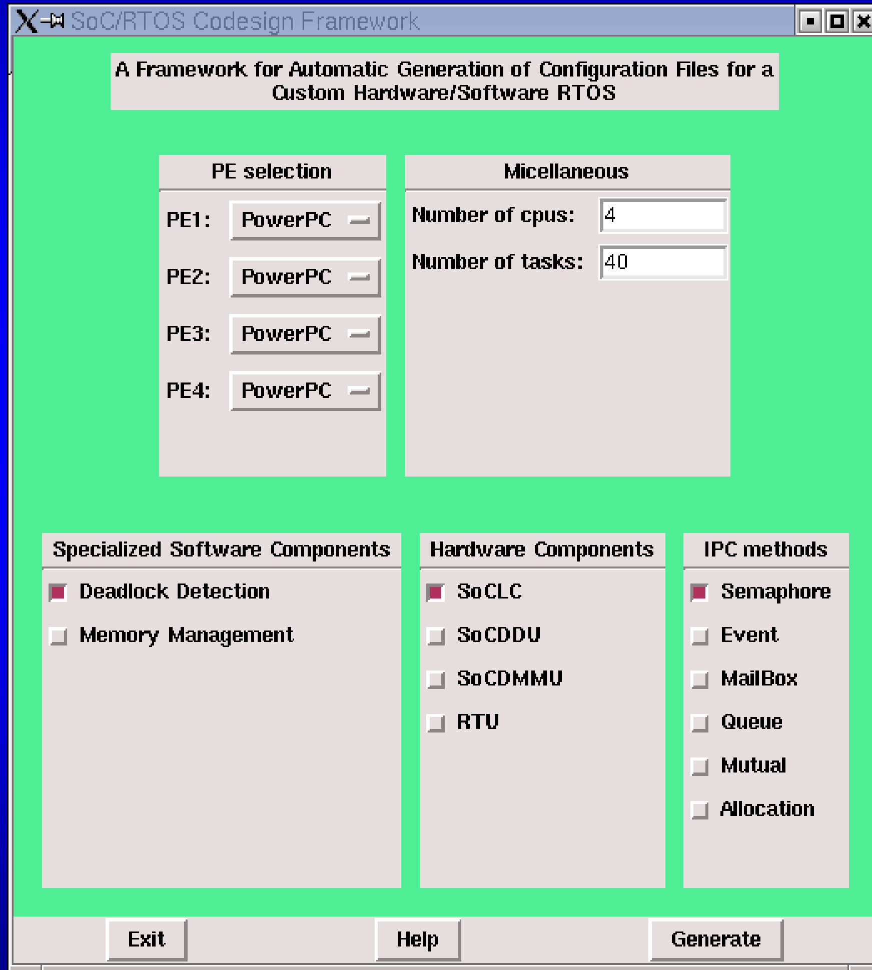
©Vincent J. Mooney III, 2002

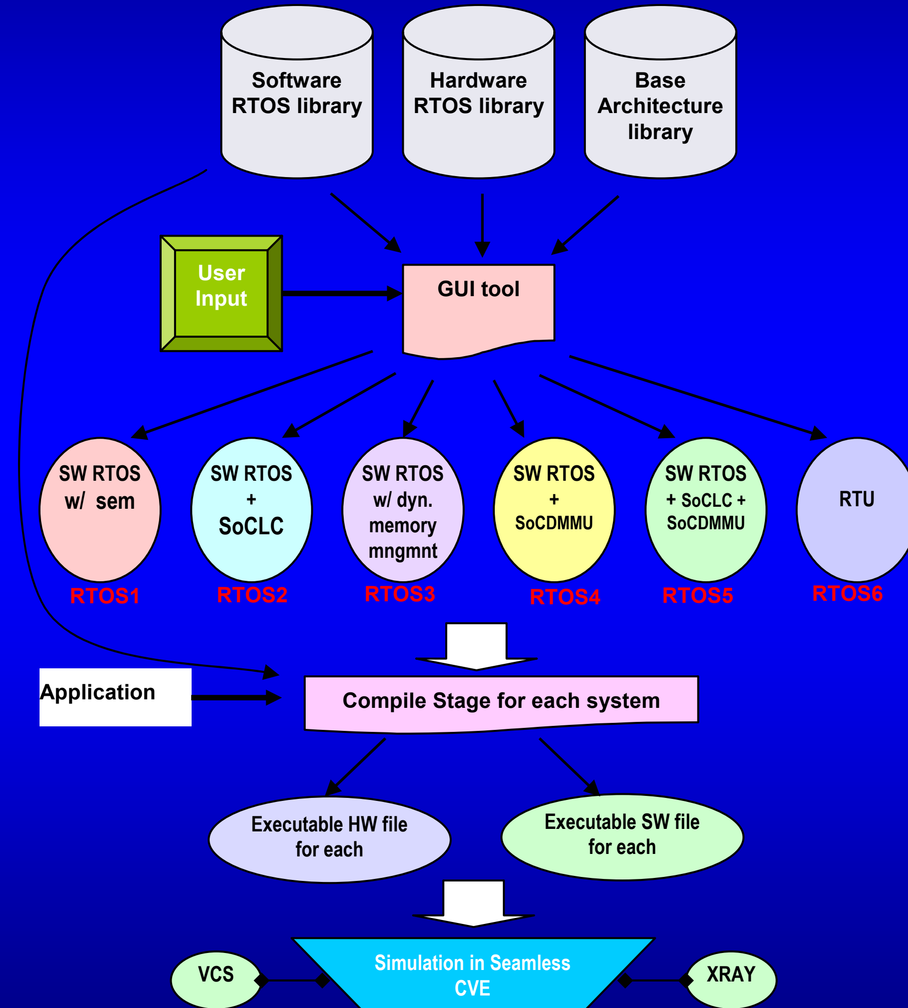
Approach

- 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

©Vincent J. Mooney III, 2002

The δ Hardware/Software RTOS Generation Framework





©Vincent J. Mooney III, 2002

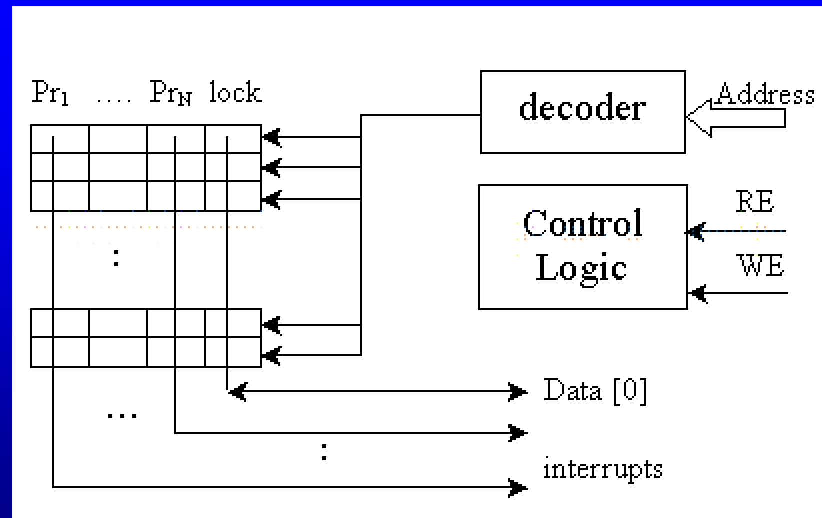
Outline

- 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

©Vincent J. Mooney III, 2002

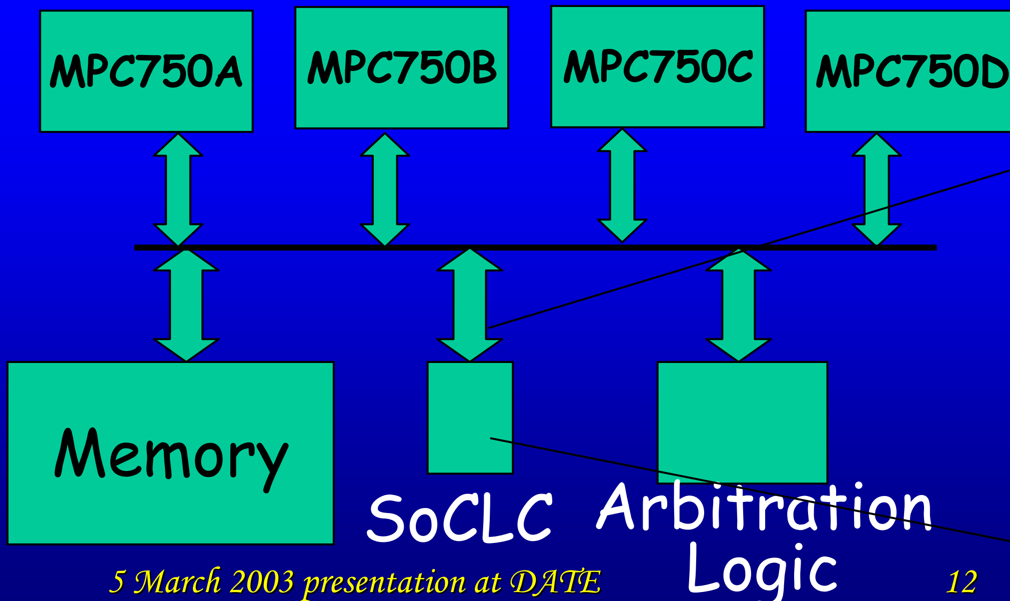
SoC Lock Cache

- 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



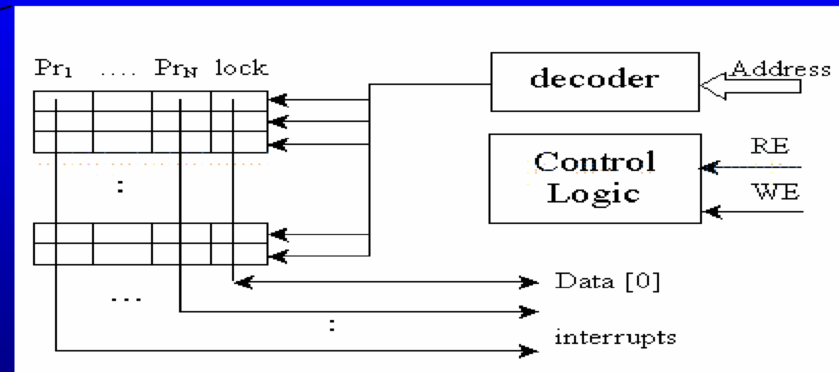
©Vincent J. Mooney III, 2002

Software/Hardware Architecture



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

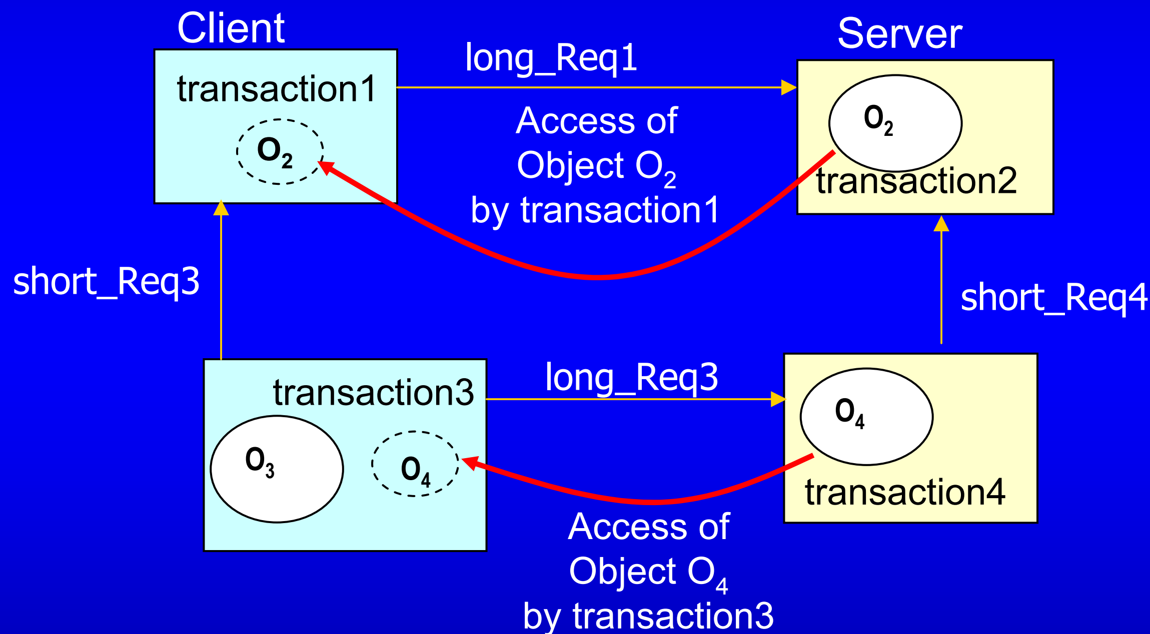
SoC Lock Cache



©Vincent J. Mooney III, 2002

Experiment

- Example: Database transaction application [1]



[1] M. A. Olson, "Selecting and implementing an embedded database system," *IEEE Computer*, pp.27-34, September 2000.

©Vincent J. Mooney III, 2002

Experimental Result

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

(clock cycles)	* Without SoCLC	With SoCLC	Speedup
Lock Latency	1200	908	1.32x
Lock Delay	47264	23590	2.00x
Execution Time	36.9M	29M	1.27x

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

[2] B. S. Akgul, J. Lee and V. Mooney, "System-on-a-chip processor synchronization hardware unit with task preemption support," *CASES '01*, pp.149-157, November 2001.

©Vincent J. Mooney III, 2002

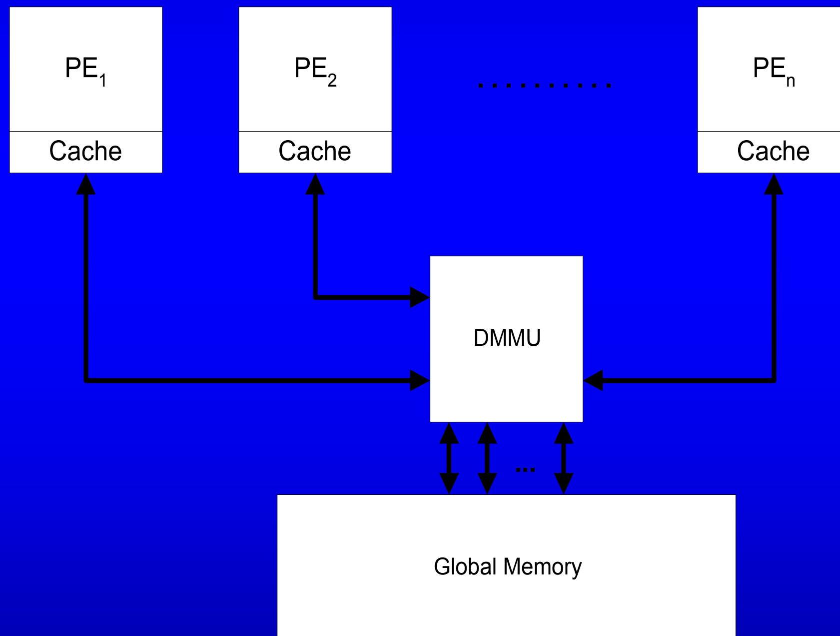
Outline

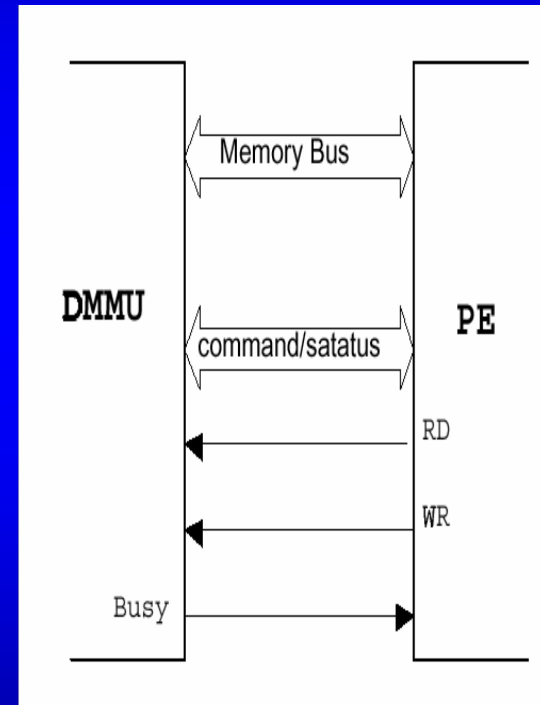
- Vision: Hardware/Software Real-Time Operating System
- Custom RTOS Hardware IP Components
  - System-on-a-Chip Lock Cache (SoCLC)
  - SoC Dynamic Memory Management Unit (SoCDMMU)
- The δ Hardware/Software RTOS Generation Framework
  - Comparison with the RTU Hardware RTOS
- Conclusion

©Vincent J. Mooney III, 2002

SoCDMMU: Move L2 Memory Allocation to One (Hardware) Unit

=> "Undistribute" L2 Memory Allocation Algorithm



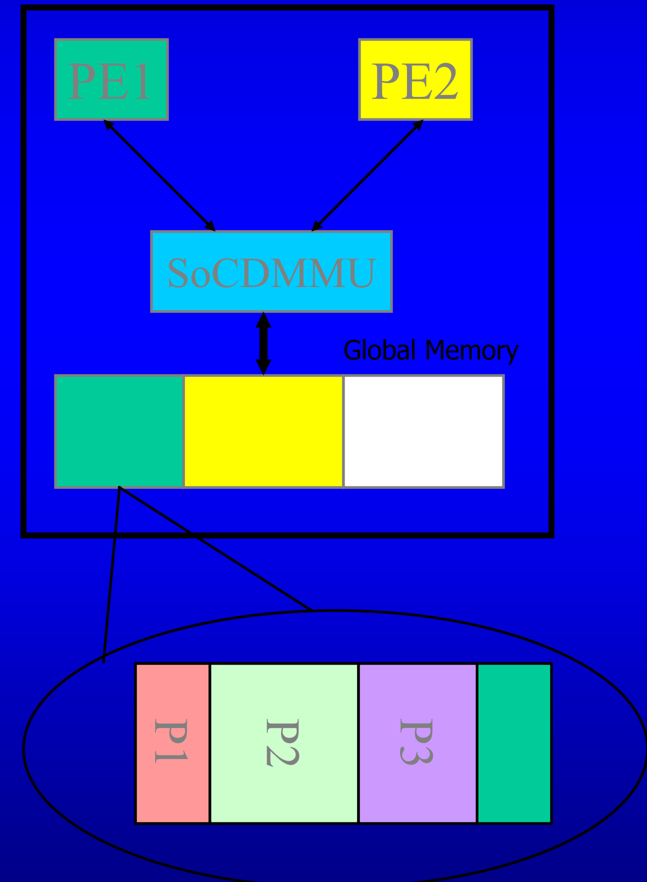


©Vincent J. Mooney III, 2002

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).



©Vincent J. Mooney III, 2002

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).

Command	Number of Cycles
G_alloc_ex	4
G_alloc_rw	4
G_alloc_ro	3
G_dealloc	4
4-Processors WCET	**16**

©Vincent J. Mooney III, 2002

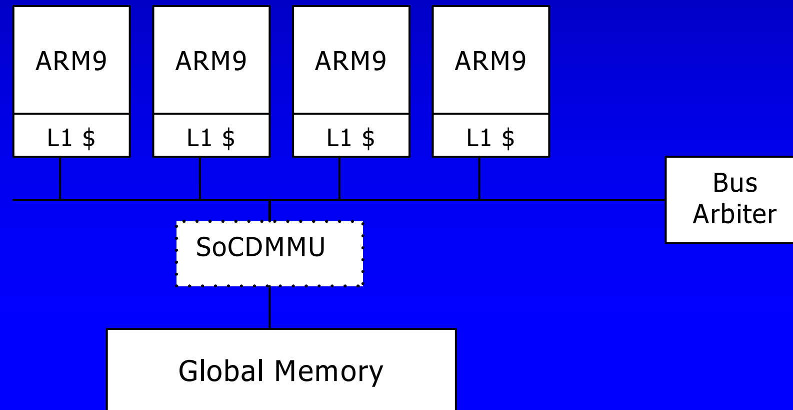
Atalanta Support for the SoCDMMU

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

©Vincent J. Mooney III, 2002

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.

©Vincent J. Mooney III, 2002

Area Estimate of The SoC

- ARM9TDMI Core: 112k transistors
- L1 $ (128KB: 64KB I$ + 64KB D$): ~6.5M* transistors
- SoCDMMU (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.

©Vincent J. Mooney III, 2002

Comparison to a Fully Shared-Memory Multiprocessor System

- Sequence of Memory Allocations Required

MPEG-2 Player	OFDM Receiver
2 Kbytes	34 Kbytes
500 Kbytes	32 Kbytes
5 Kbytes	1 Kbytes
1500 Kbytes	1.5 Kbytes
1.5 Kbytes	32 Kbytes
0.5 Kbytes	8 Kbytes
	32 Kbytes

©Vincent J. Mooney III, 2002

Comparison to a Fully Shared-Memory Multiprocessor System

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

	Using the SoCDMMU	Using ARM SDT *malloc()* and *free()*	Speedup
Average Case	281 cycles	1240 cycles	4.4X*
Worst Case	1244 cycles	4851 cycles	3.9X

***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

©Vincent J. Mooney III, 2002

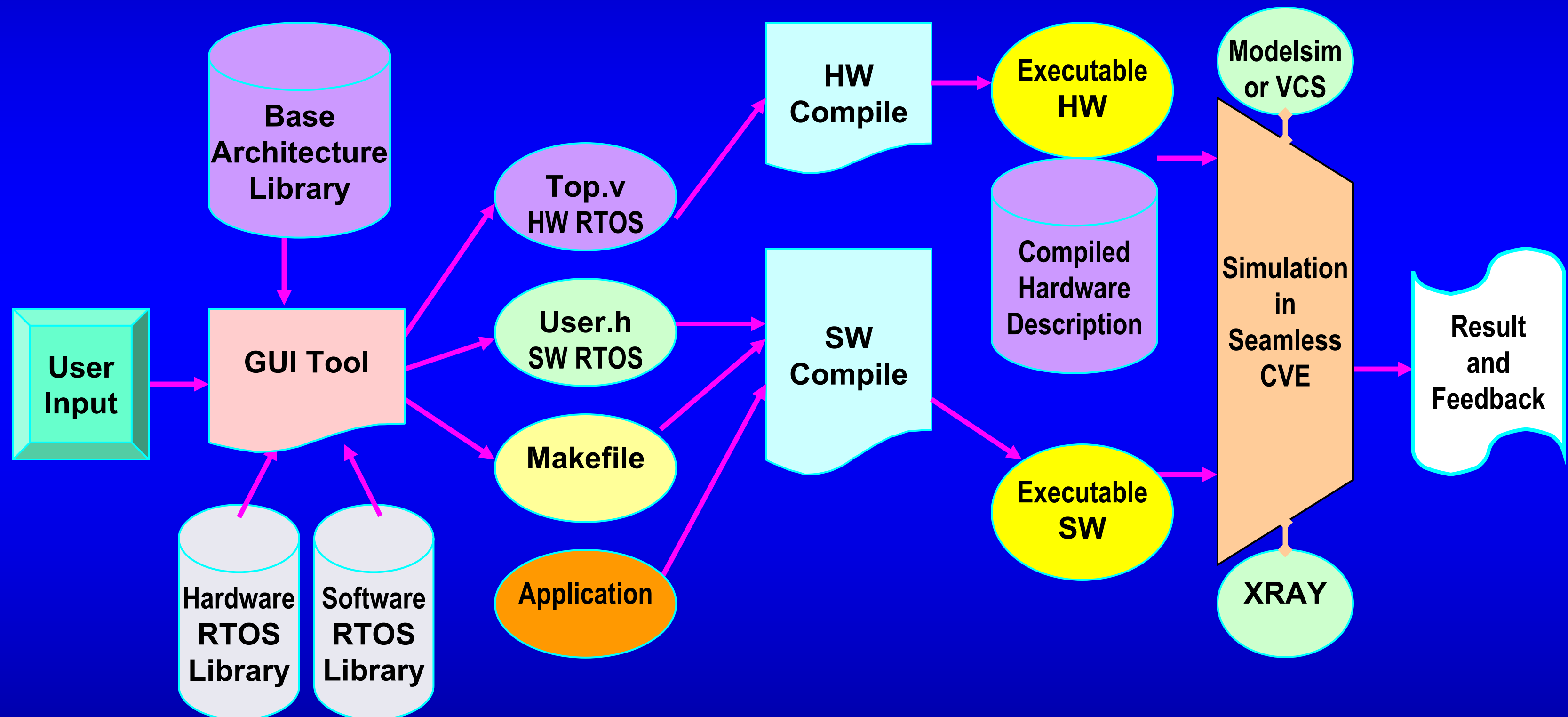
Outline

- 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

©Vincent J. Mooney III, 2002

δ Hardware/Software RTOS Generation Framework

and current simulation platform



©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

©Vincent J. Mooney III, 2002

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

©Vincent J. Mooney III, 2002

Motivation (2/2)

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

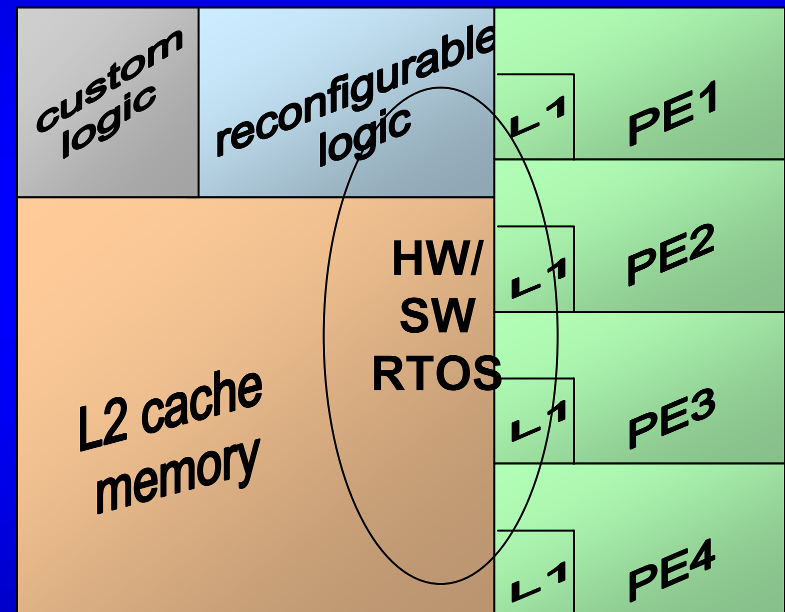
$\Rightarrow$ The δ framework

- 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

©Vincent J. Mooney III, 2002

Our RTOS in Post-Fabricaton Scenario

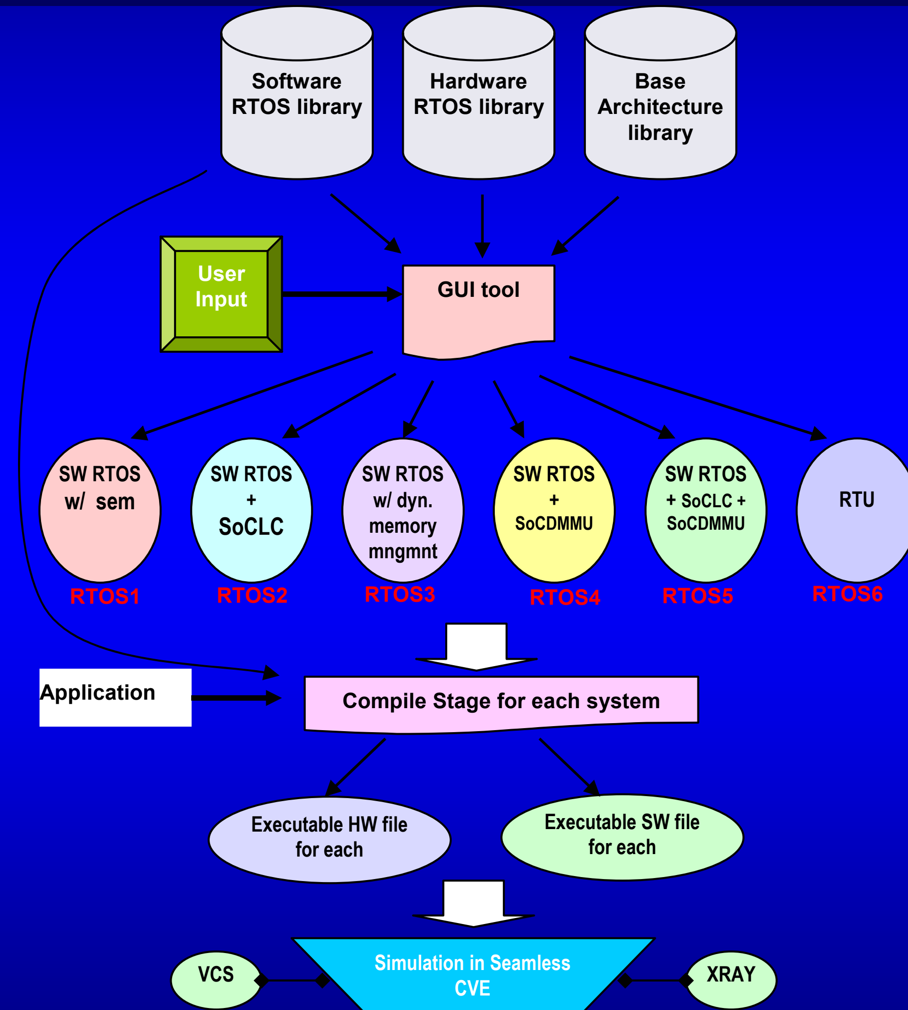
- Application(s) run on the SoC using standard RTOS APIs
- Atalanta software RTOS
  - A multiprocessor SoC RTOS
    - The RTOS and device drivers are loaded into the L2 cache memory
  - All Processing Elements (PEs)
    - share the kernel code and data structures
- Hardware RTOS components are downloaded into the reconfigurable logic



©Vincent J. Mooney III, 2002

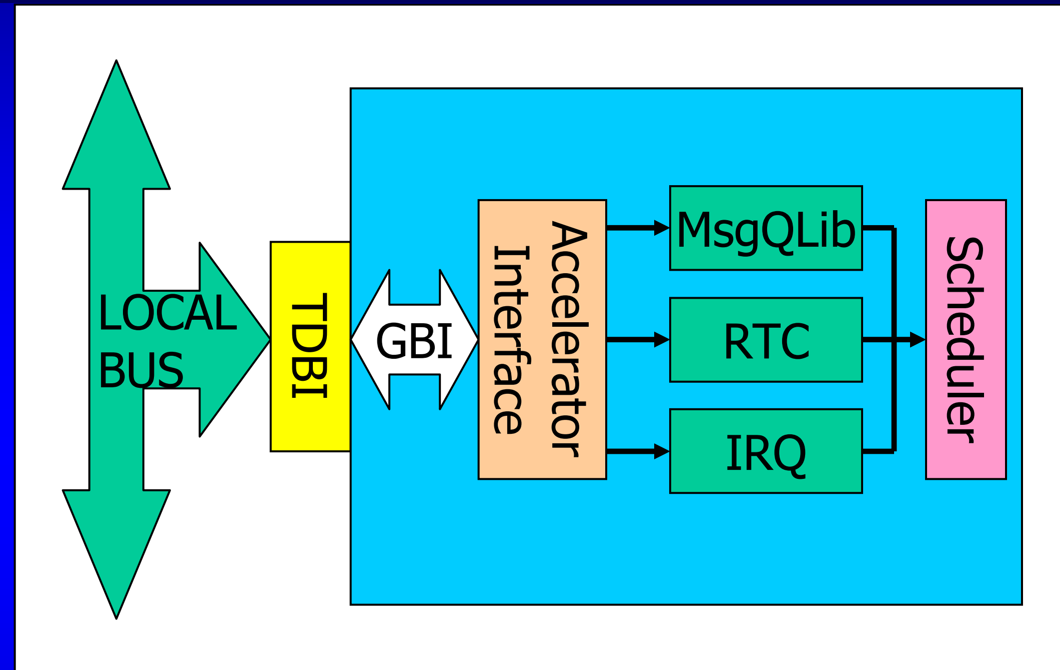
Experimental Setup

- 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



©Vincent J. Mooney III, 2002

RTU Hardware RTOS

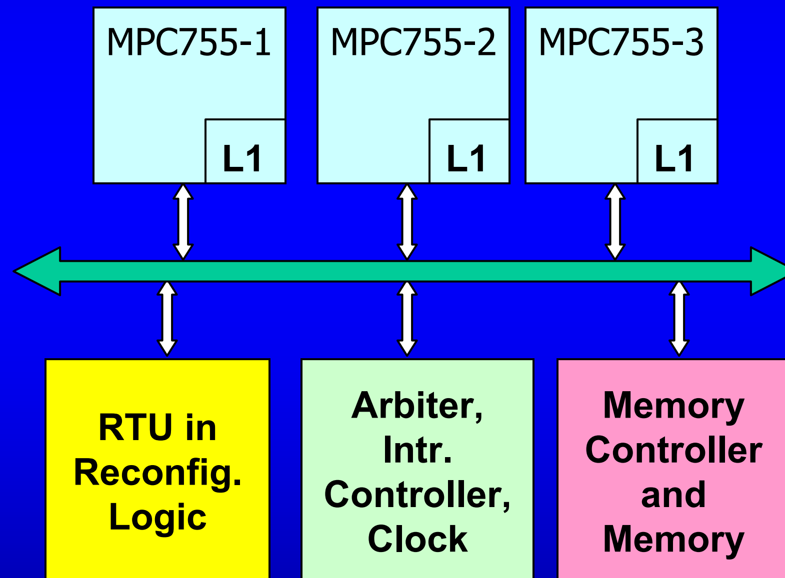


- An RTOS in hardware
  - Real-Time Unit (RTU)
    - scheduling
    - IPC
    - dynamic task creation
    - timers
  - Custom hw => upper bound on # tasks
  - Reconfigurable hw => can alter max. # tasks, max. # priorities
  - Prof. Lennart Lindh, Mälardalens U., Västerås, Sweden
  - RealFast, www.realfast.se

©Vincent J. Mooney III, 2002

Methodology

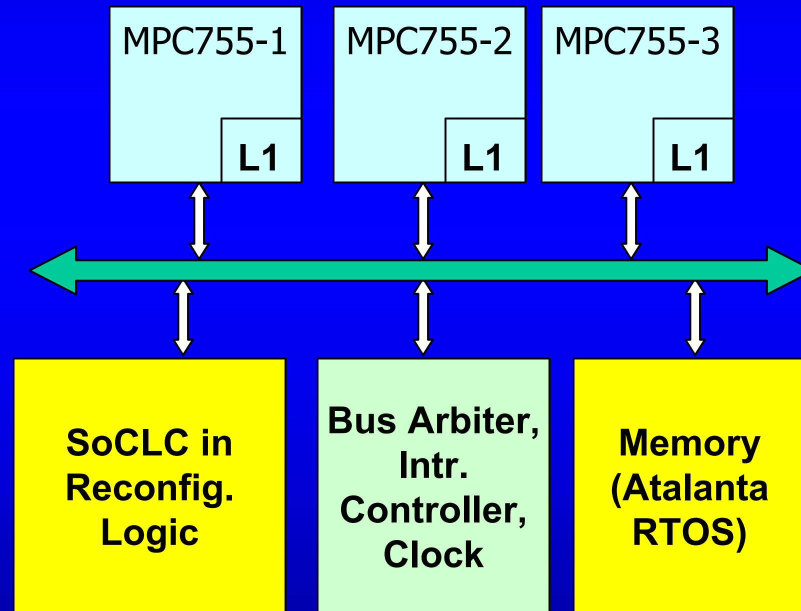
- An SoC architecture with the RTU Hardware RTOS



©Vincent J. Mooney III, 2002

Methodology

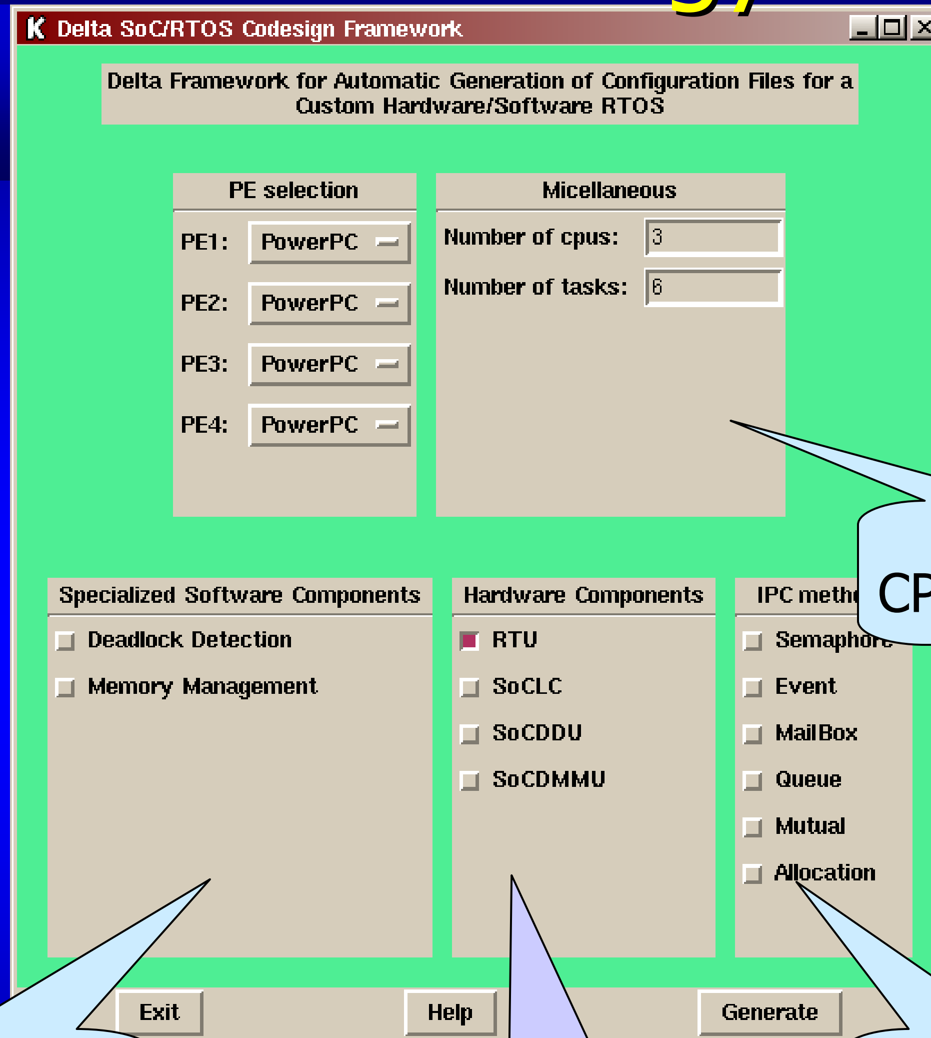
- An SoC architecture with a hardware/software RTOS



©Vincent J. Mooney III, 2002

Methodology

- δ Framework
  – GUI



Number of CPUs in system

Specilized SW RTOS component

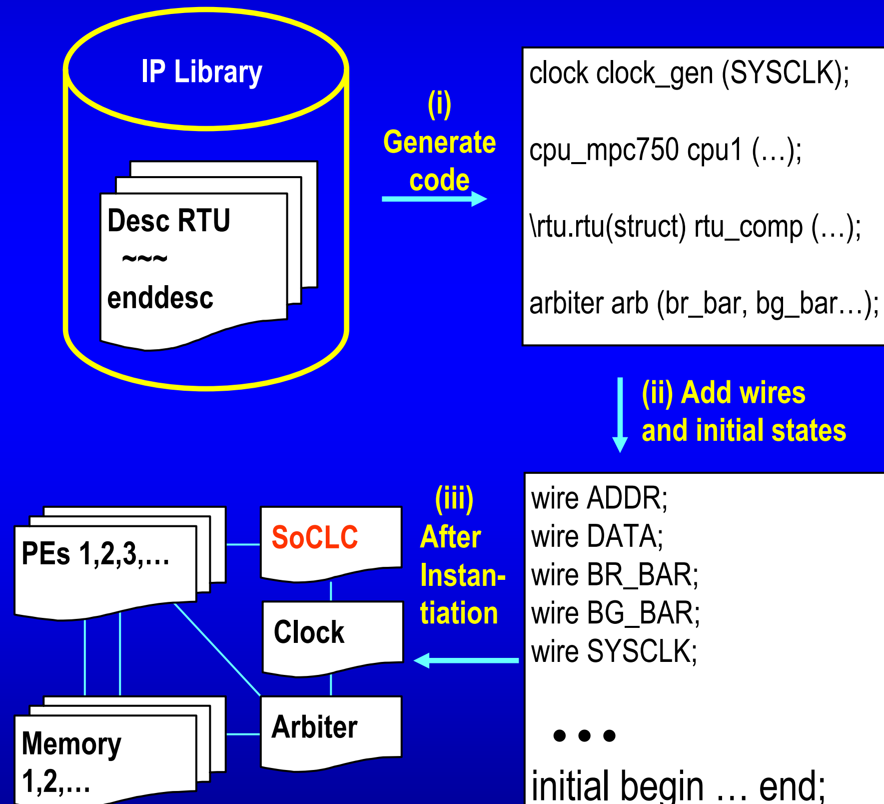
HW RTOS component

IPC module linking method

©Vincent J. Mooney III, 2002

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



©Vincent J. Mooney III, 2002

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

Total Execution Time		Pure SW *	With SoCLC	With RTU
6 tasks	(in cycles)	100398	71365	67038
	Reduction	0%	29%	33%
30 tasks	(in cycles)	379440	317916	279480
	Reduction	0%	16%	26%

* A semaphore is used in pure software and a hardware mechanism is used in SoCLC and RTU.

©Vincent J. Mooney III, 2002

Experimental Results (2/3)

- The number of interactions

Times	6 tasks	30 tasks
Number of semaphore interactions	12	60
Number of context switches	3	30
Number of short locks	10	58

©Vincent J. Mooney III, 2002

Experimental Results (4/4)

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

cycles	Pure SW	With SoCLC	With RTU
communication	18944	3730	2075
context switch	3218	3231	2835
computation	8523	8577	8421

©Vincent J. Mooney III, 2002

Hardware Area

Total area	SoCLC (64 short CS locks + 64 long CS locks)	RTU for 3 processors
TSMC 0.25μm library from LEDA	7435 gates	About 250000 gates

©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

©Vincent J. Mooney III, 2002