

# Hardware/Software Codesign of a Real-Time Operating System

*The  $\delta$  Hardware/Software RTOS Generation Framework*

**Vincent J. Mooney III**

<http://codesign.ece.gatech.edu>

<http://www.crest.gatech.edu>

**Associate Professor, School of Electrical and Computer Engineering**

**Adjunct Associate Professor, College of Computing**

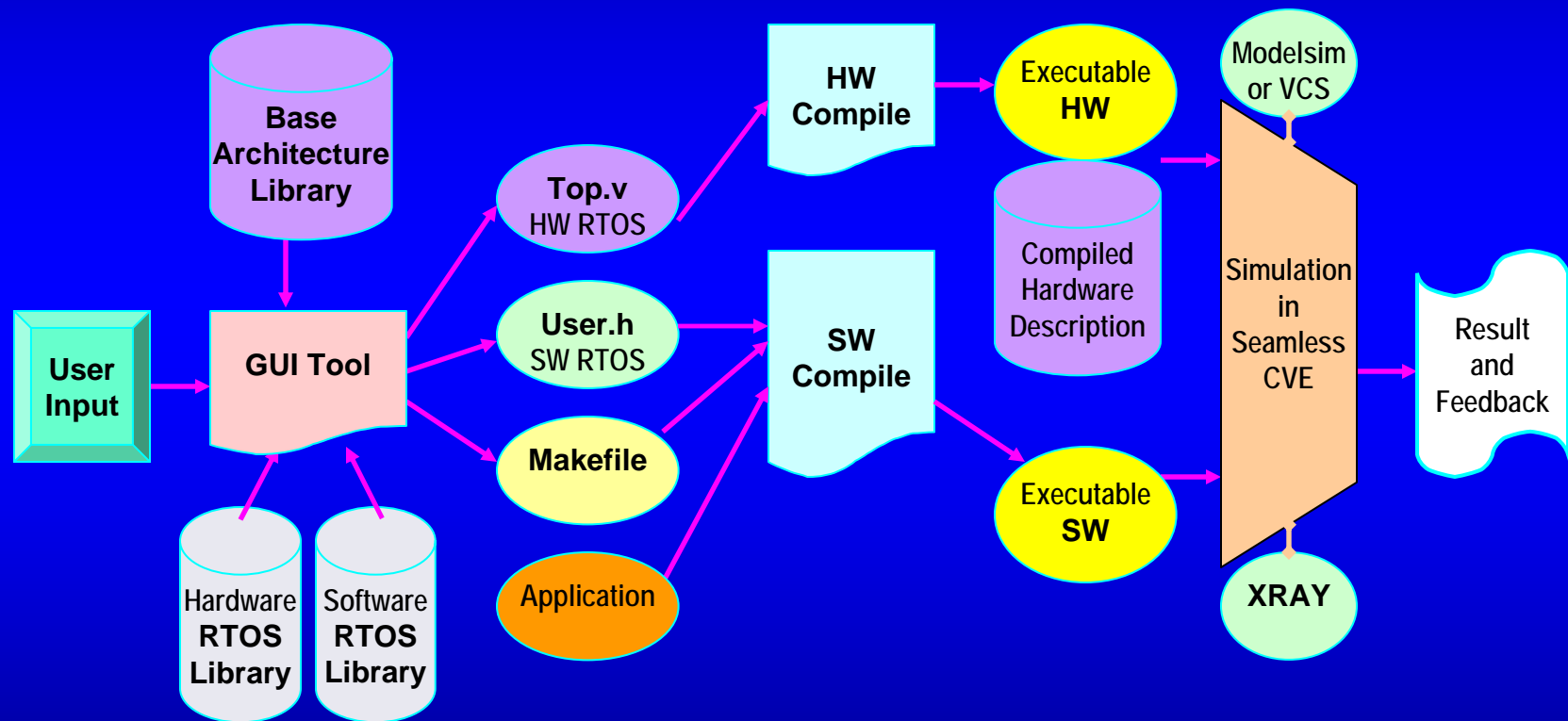
**Founder, Hardware/Software Codesign Laboratory**

**Co-Director, Center for Research on Embedded Systems and Technology**

**Georgia Institute of Technology**

**Atlanta, Georgia, USA**

# $\delta$ Hardware/Software RTOS Generation Framework *and current simulation platform*



# $\delta$ Hardware/Software RTOS Generation Framework Goals

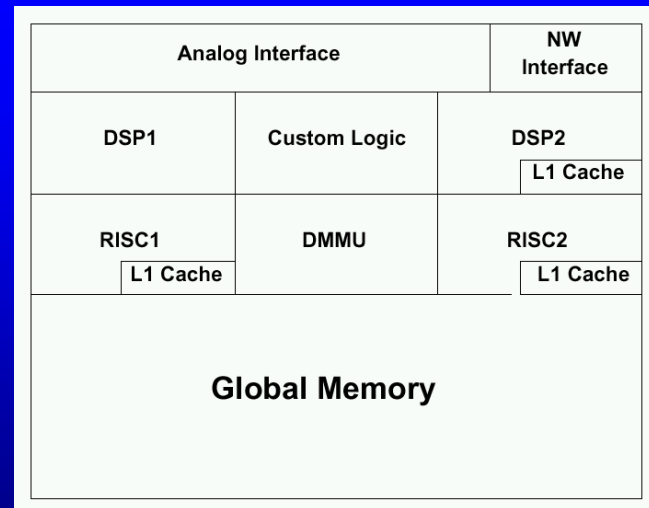
- To help the user examine which configuration is most suitable for the user's specific applications
- To help the user explore the RTOS design space before chip fabrication as well as after chip fabrication (in which case reconfigurable logic must be available on the chip)
- To help the user examine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

# Motivation (1/3)

- HW/SW RTOS partitioning approach
- Previous innovations in HW/SW RTOS components
  - System-on-a-Chip Lock Cache (SoCLC)
  - System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)
  - System-on-a-Chip Deadlock Detection Unit (SoCDDU), Deadlock Avoidance Unit (SoCDAU) and Parallel Banker's Algorithm Unit (PBAU)
- RTU Hardware RTOS

# Motivation (2/3)

- 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 (3/3)

Constraints about using the previous HW/SW RTOS innovations

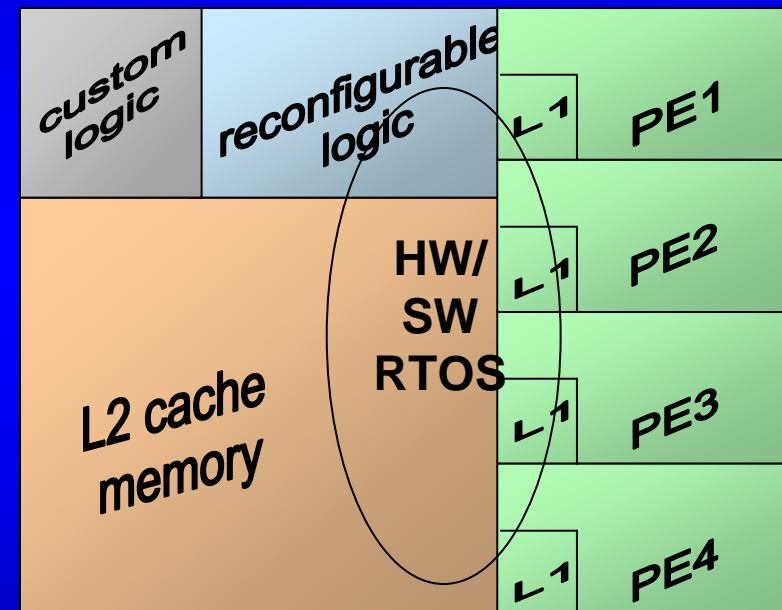
- Perhaps not enough chip space for all three of them
- All of them may not be necessary

⇒ The  $\delta$  framework

- Enables automatic generation of different mixes of the 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

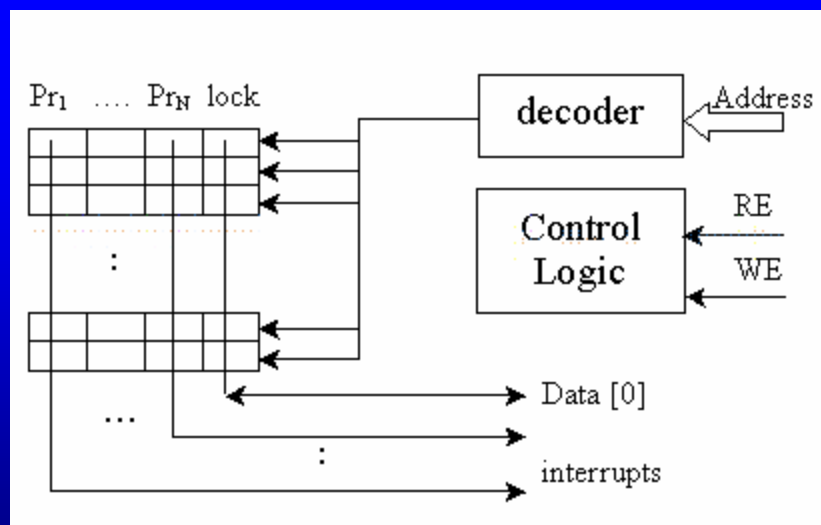
# Our RTOS in Post-Fabrication Scenario

- 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

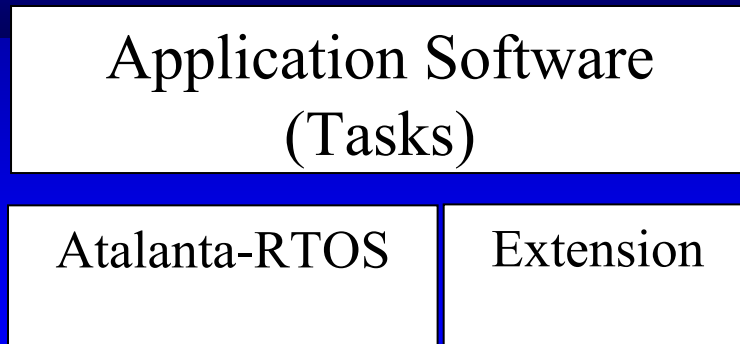


# SoC Lock Cache

- 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



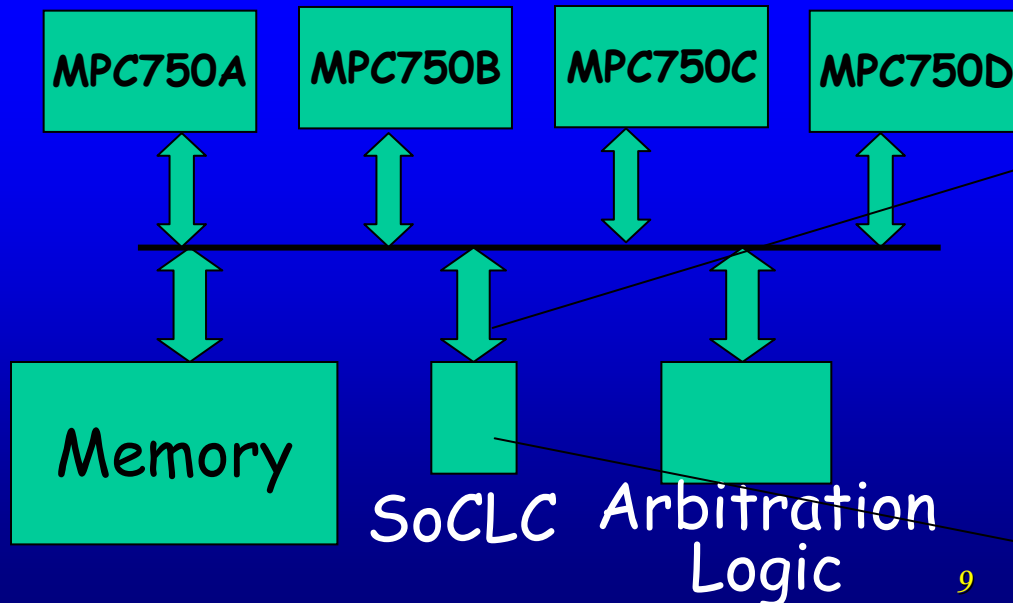
# Software/Hardware Architecture



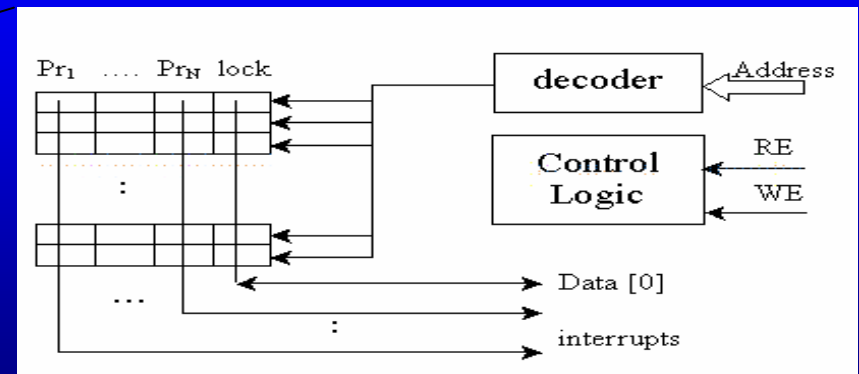
- Multiple application tasks
- Atalanta-RTOS
- Four MPC750s
- SoCLC provides lock synchronization among PEs

Software

Hardware

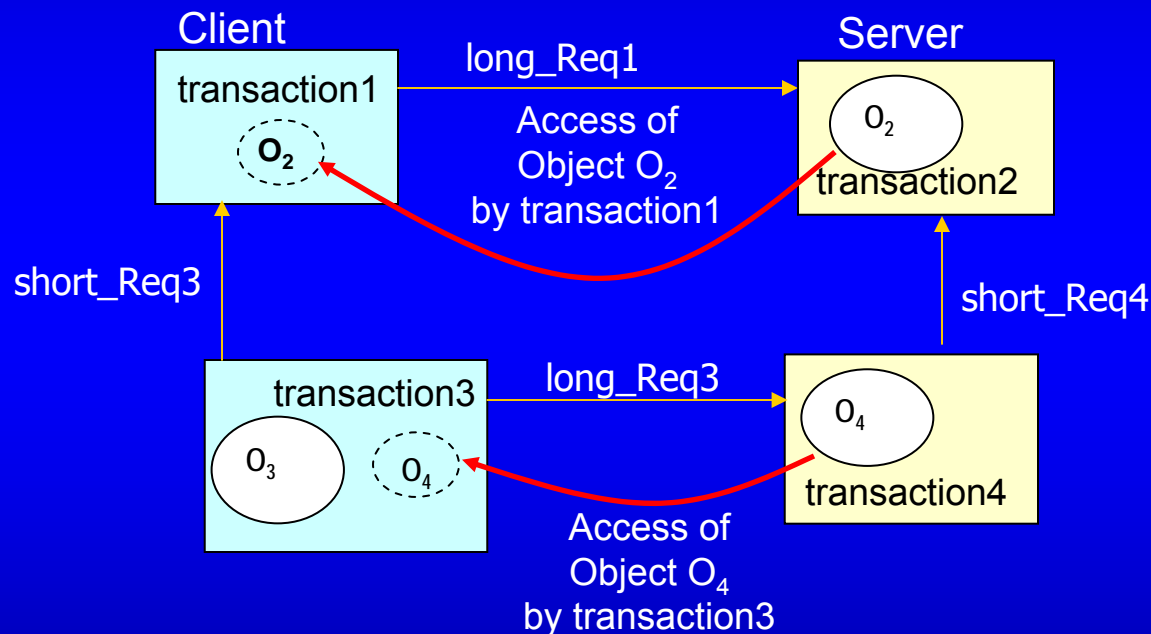


## SoC Lock Cache



# Experiment

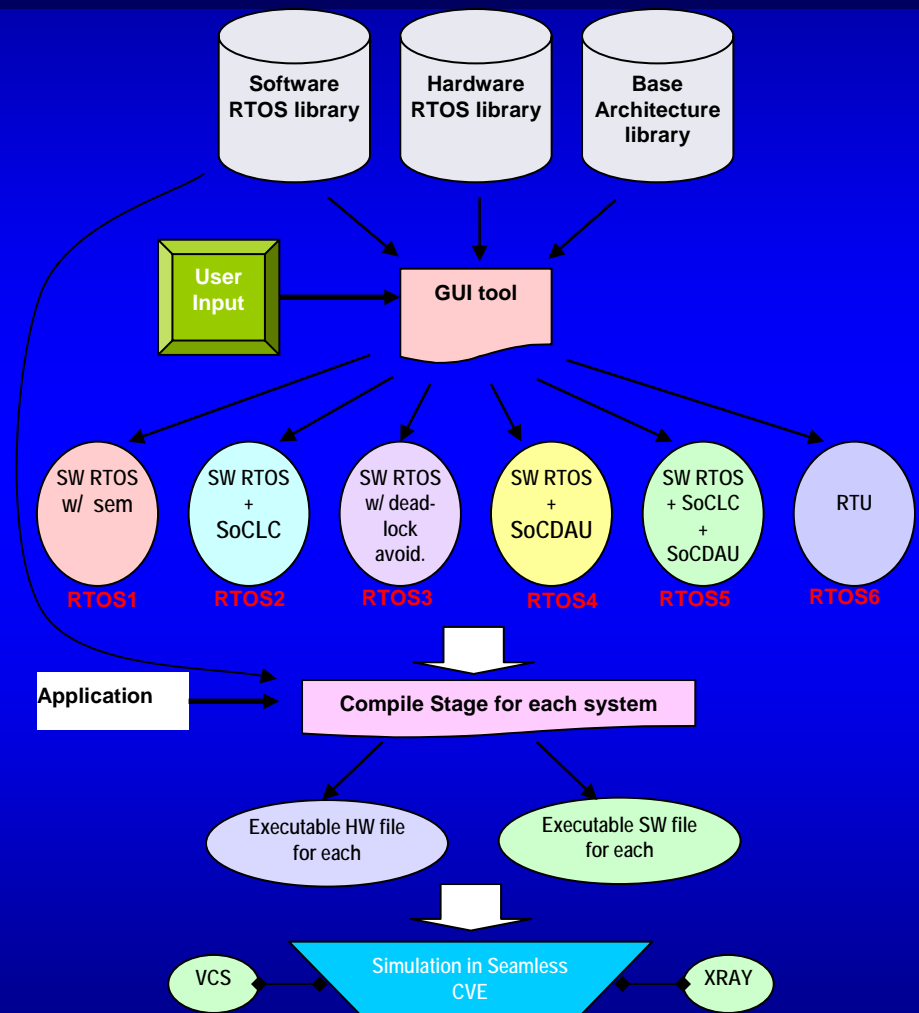
- Example: Database transaction application [1]



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

# Experimental Setup

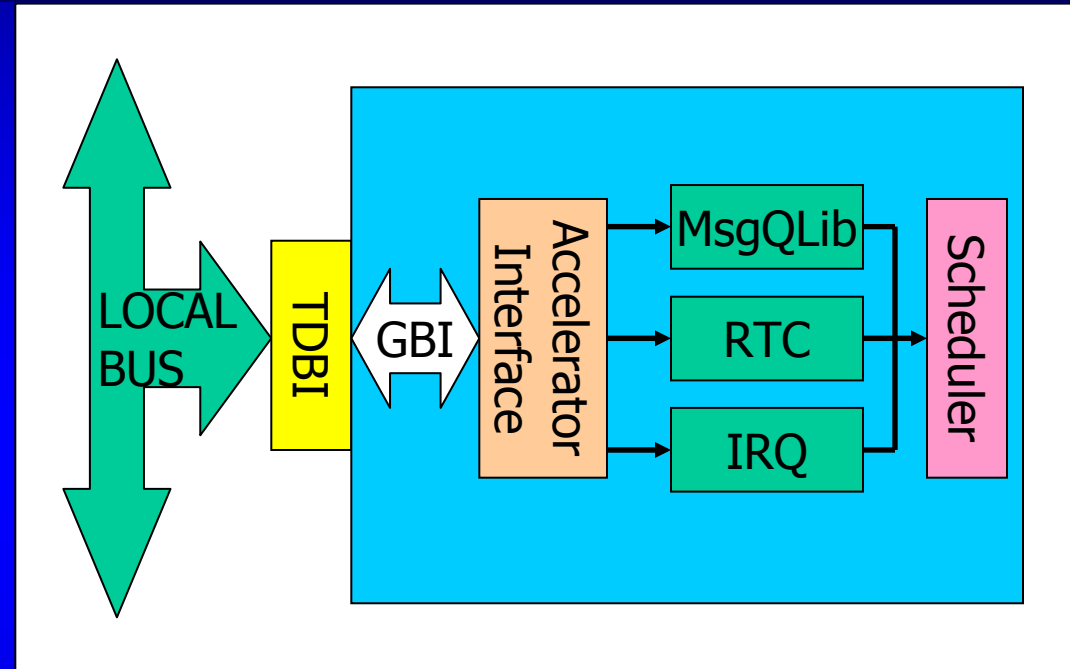
- Six custom RTOSes
  - With semaphores and spin-locks, no HW components in the RTOS
  - With SoCLC, no SW IPCs
  - With deadlock avoidance software, no HW RTOS components
  - With SoCDAU
  - With SoCLC and SoCDAU
  - With RTU
- Each with the *Base* architecture
- Each with application(s)
- Each executable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



# RTU Hardware RTOS

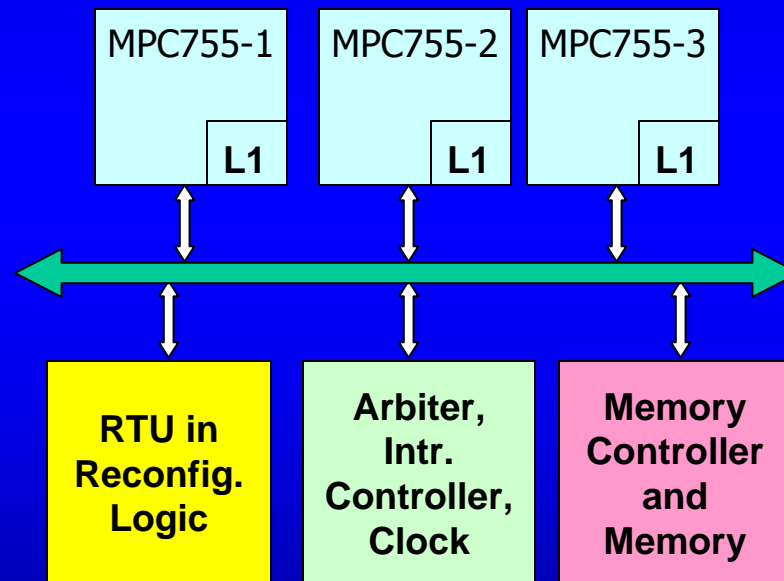
## ■ An RTOS in hardware

- 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](http://www.realfast.se)



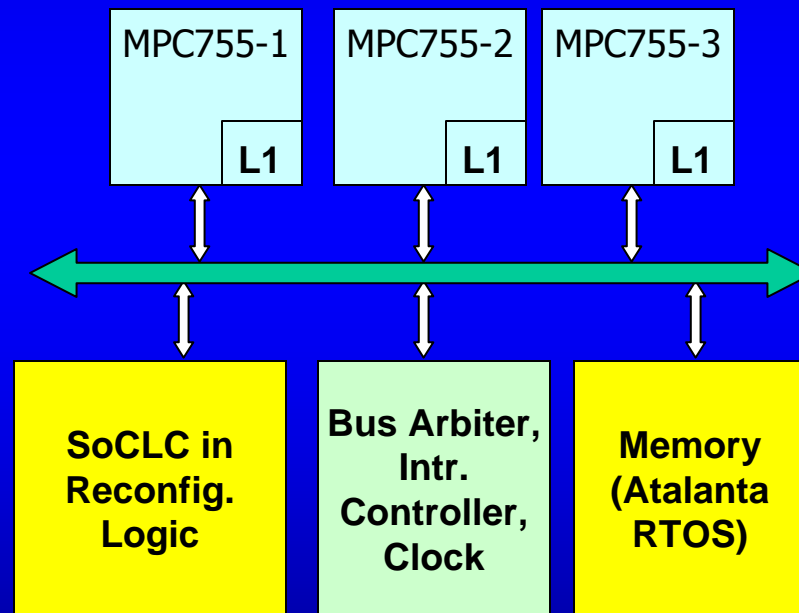
# Methodology

- An SoC architecture with the RTU Hardware RTOS



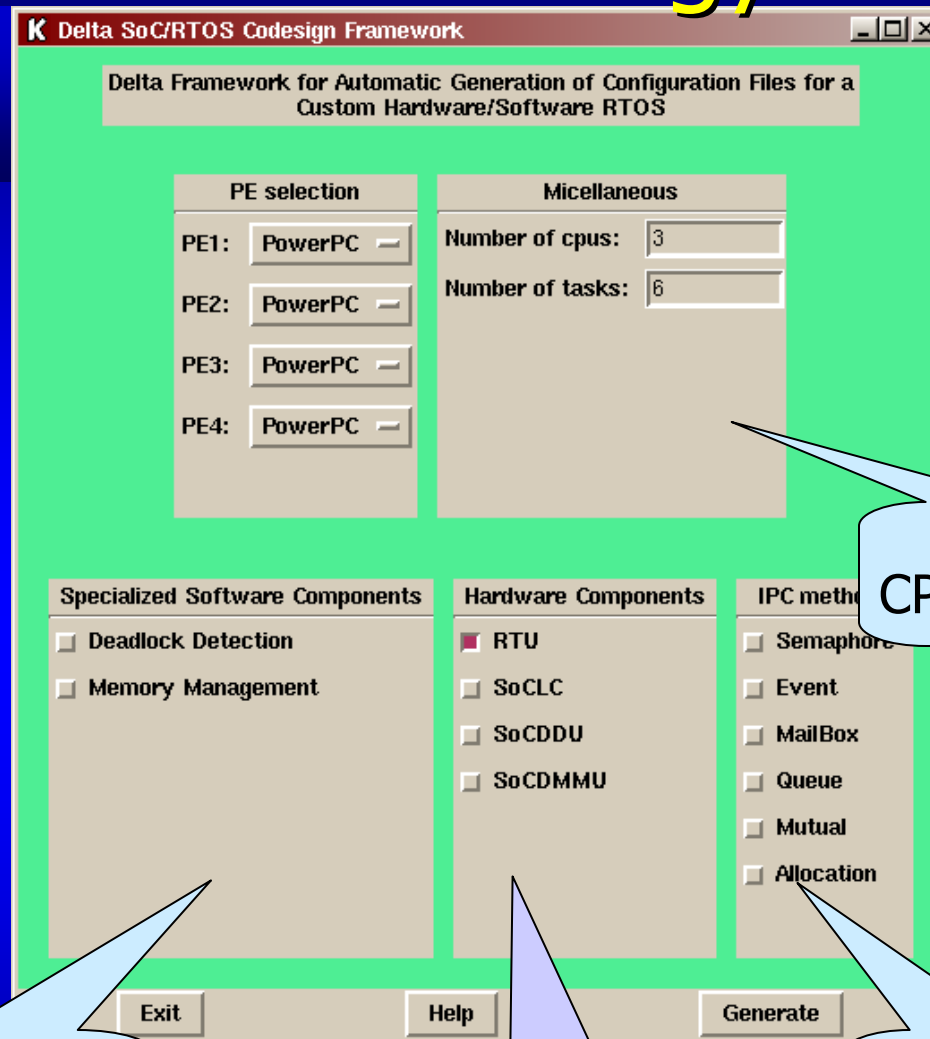
# Methodology

- An SoC architecture with a hardware/software RTOS



# Methodology

- $\delta$  Framework
- GUI



Number of CPUs in system

Specilized SW RTOS component

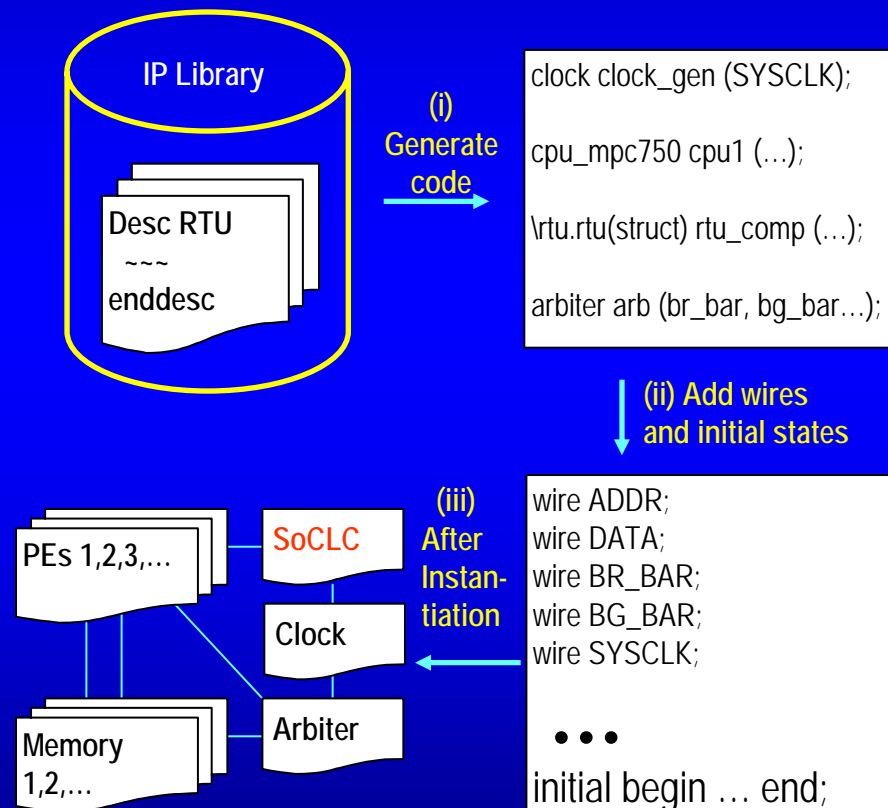
HW RTOS component

IPC module linking method

# 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



# 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.

# 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

# Experimental Results (3/3)

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

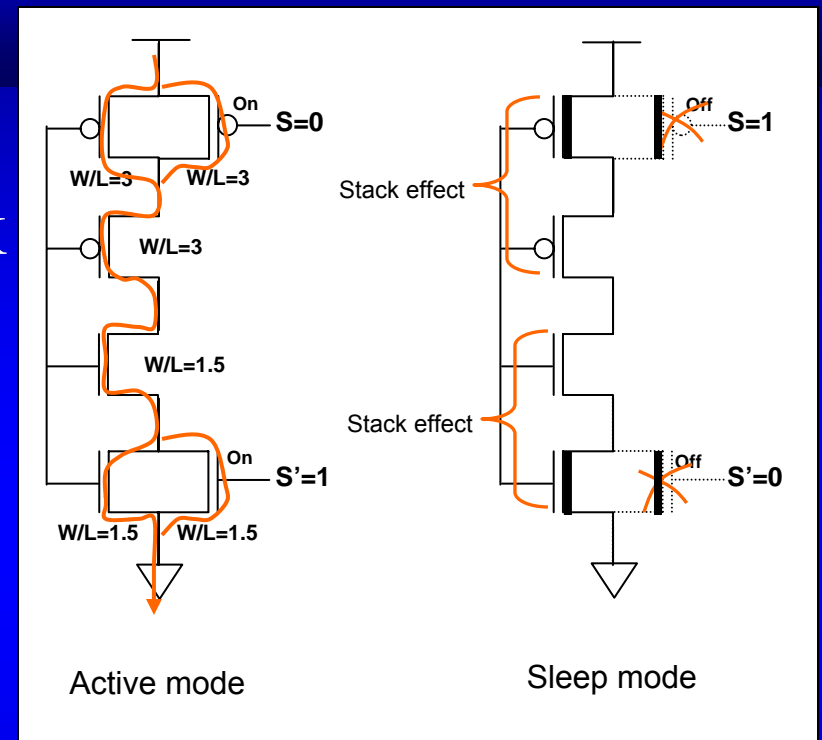
cycles	Pure SW	With SoCLC	With RTU
communication	18944	3730	2075
context switch	3218	3231	2835
computation	8523	8577	8421

# Hardware Area

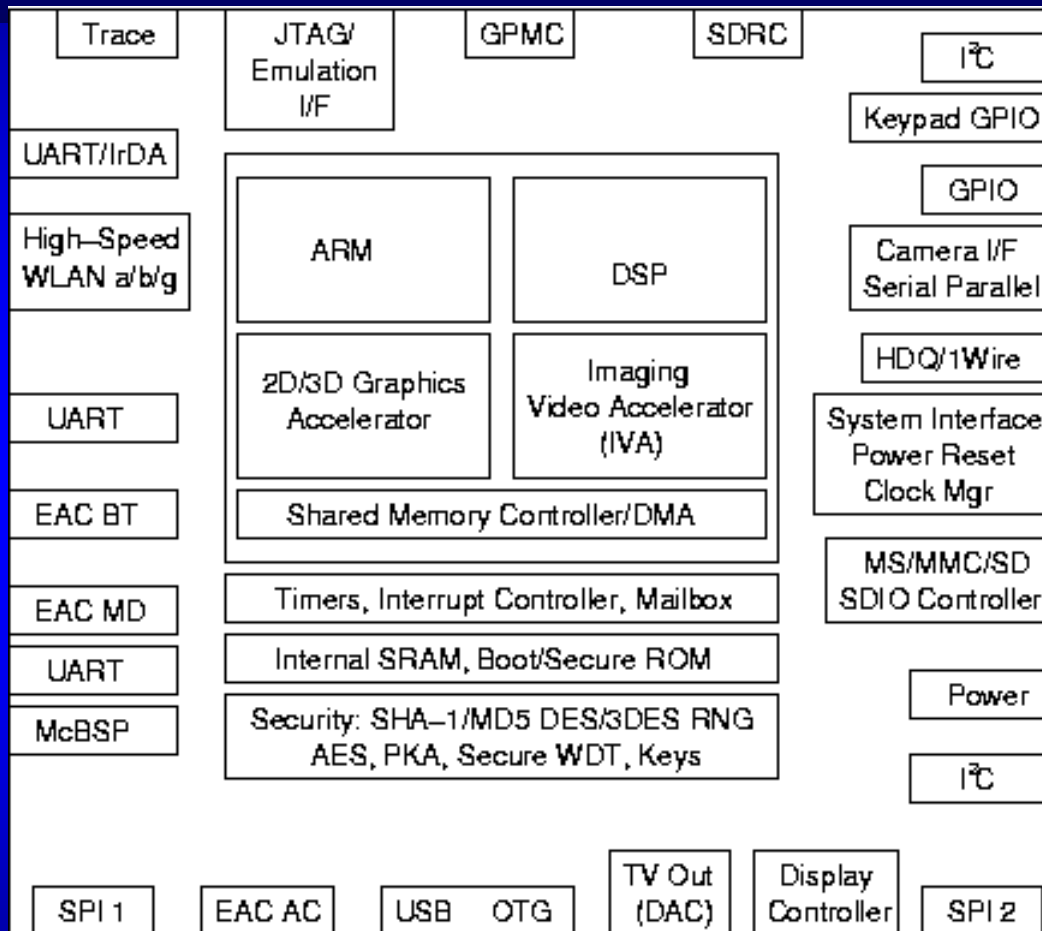
Total area	SoCLC (64 short CS locks + 64 long CS locks)	RTU for 3 processors
TSMC 0.25 $\mu$ m library from LEDA	7435 gates	About 250000 gates

# Other Topics

- Sleepy Stack VLSI – J.C. Park
- Reverse Execution for Debugging – Tankut Akgul
- Worst Case Response Time – Yudong Tan
- Papers and pdfs of presentations available at the codesign website <http://codesign.ece.gatech.edu>



# Proliferation of Peripherals



- 26 peripherals
  - most customers use few
- Cannot scale well
- Opportunity for static & dyn. reconfig.
  - imagine thousands of downloadable configurations
  - dynamic and invisible to user

# “Third Way”?

- Whither language?
- Matlab – floating point everywhere!
- VHDL – not liked by telecom, etc., designers
- SystemC – huge class library, unclear impact beyond hardware designers
- Java – non-optimized, memory mngmnt. abstracted
- C – tight RTL, bit-optimized, not supported
- Legacy code – ARM example
- => heterogeneous language support, languages chosen by user communities

23

# Conclusion

- A framework for automatic generation of a custom HW/SW RTOS
  - experimental results showing speedups with the SoCLC, RTU
    - additional hw RTOS component in references, e.g., for deadlock avoidance (SoCDDU, SoCDAU, SoCPBAU) and dynamic memory allocation (SoCDMMU)
- Comments on opportunities, issues
  - peripherals
  - languages