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

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

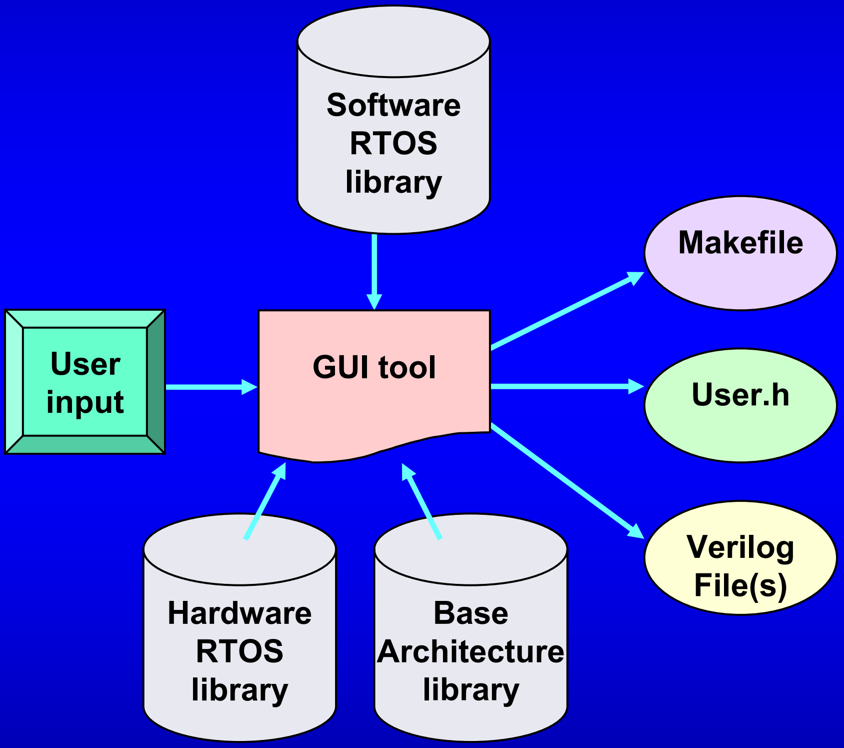
+Assistant Professor, *School of Electrical and Computer Engineering
+Adjunct Assistant Professor, College of Computing
Georgia Institute of Technology

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

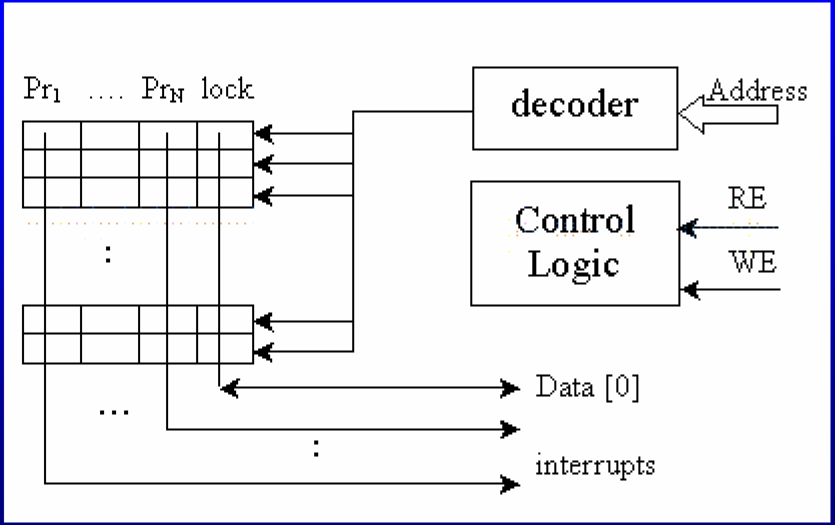
- To help the user examine which configuration is most suitable for the user's specific applications
- To help the user explore the RTOS design space after chip fabrication as well as before chip fabrication
- To help the user examine different system-on-a-chip (SoC) architectures subject to a custom 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

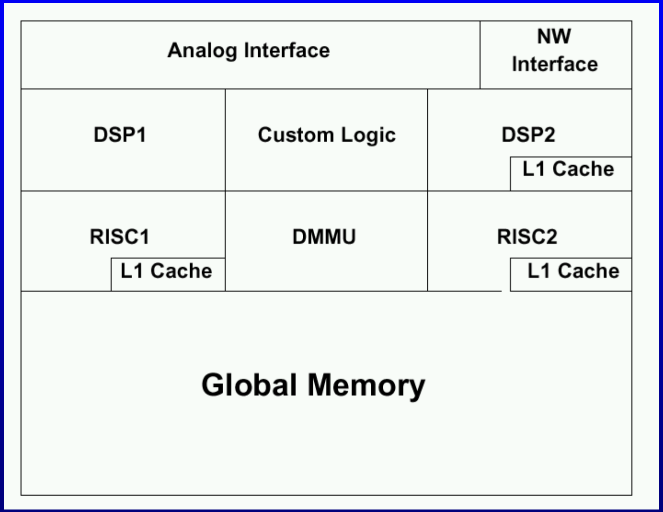
Motivation (2/5)

- 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*min(m,n)) iterations as opposed to O(m*n) required by all previously reported (sequential) software algorithms

Motivation (5/5)

Constraints about using three previous innovations

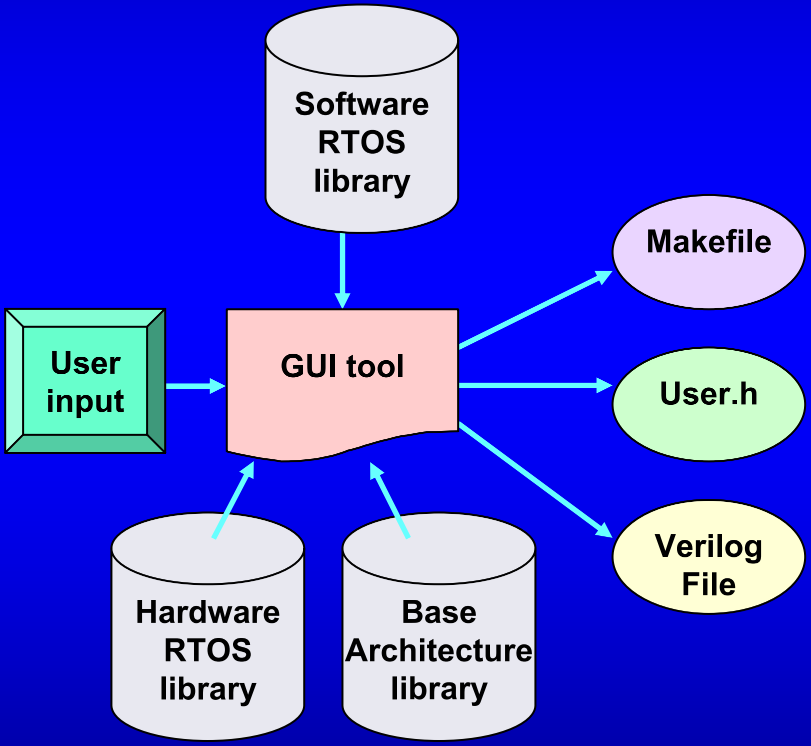
- Perhaps not enough chip space for all three of them
- All of them may not be necessary

$\Rightarrow$ 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

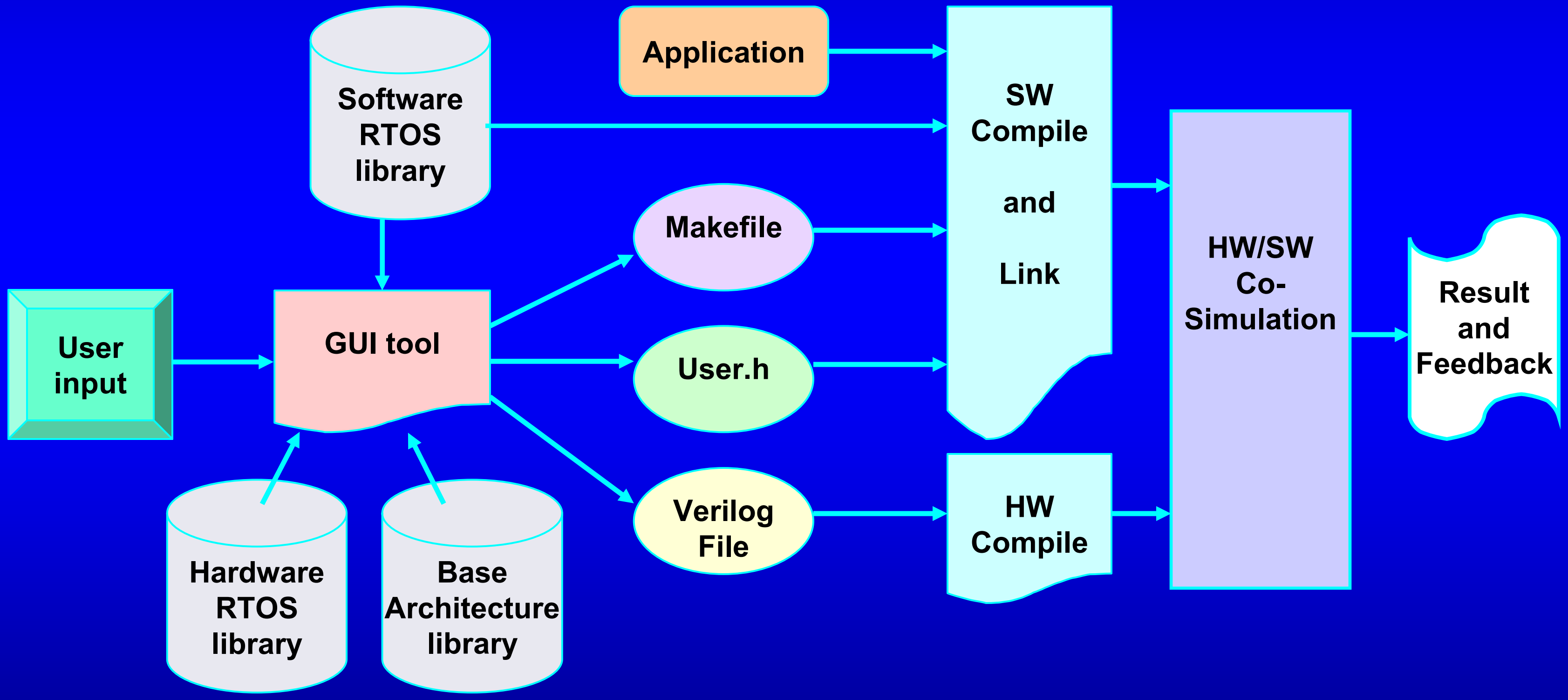
Methodology (1/2)

- 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

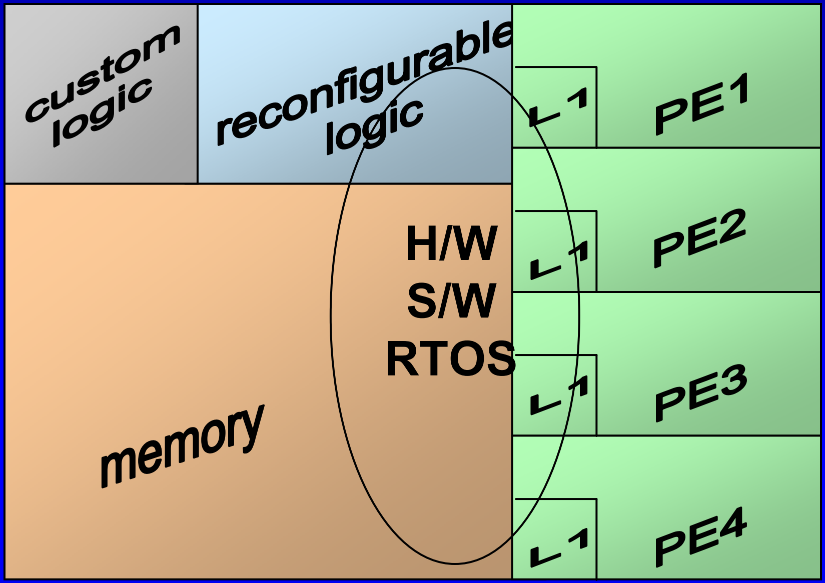


Methodology (2/2)

- Explores the HW/SW RTOS design space defined by the available HW/SW RTOS components easily



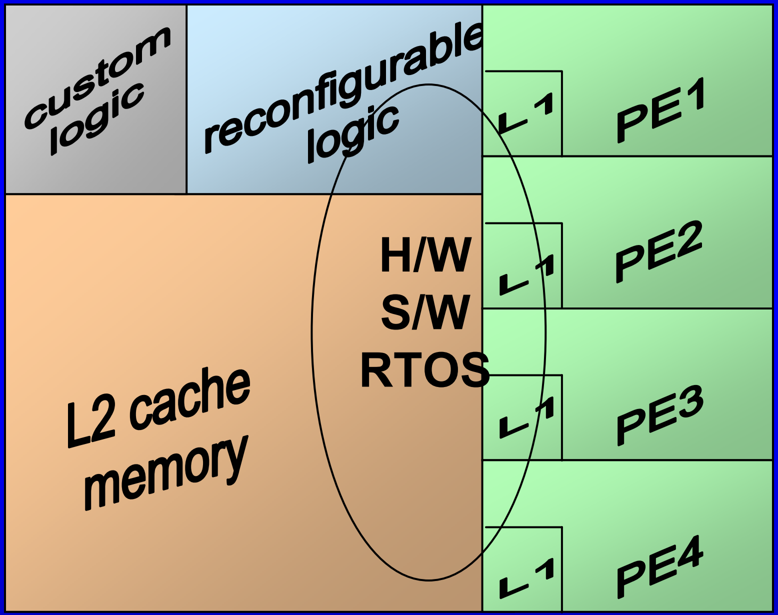
Our RTOS and Possible Target SoC



- 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

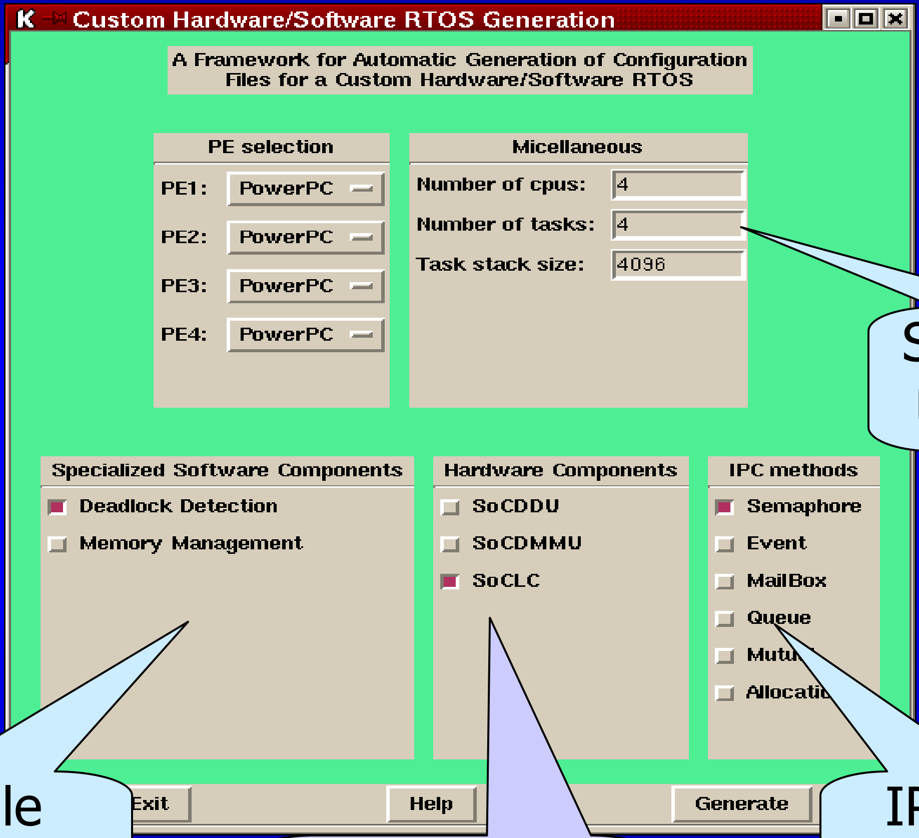
- Atalanta software RTOS
  - A multiprocessor SoC RTOS
    - The RTOS and device drivers are loaded into the L2 cache memory
  - All Processing Elements (PEs)
    - share the kernel code and data structures
- Hardware RTOS components are downloaded into the reconfigurable logic



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 (SoCDDU)

Implementation (1/8)



Custom Hardware/Software RTOS Generation

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

PE selection

PE1:	PowerPC
PE2:	PowerPC
PE3:	PowerPC
PE4:	PowerPC

Micellaneous

Number of cpus:	4
Number of tasks:	4
Task stack size:	4096

Specialized Software Components

- [✓ Deadlock Detection]
- Memory Management

Hardware Components

- SoCDDU
- SoCDMMU
- [✓ SoCLC]

IPC methods

- [✓ Semaphore]
- Event
- MailBox
- Queue
- Mutu
- Allocati

Exit | Help | Generate

SW may over-ride task size

SW module linking method

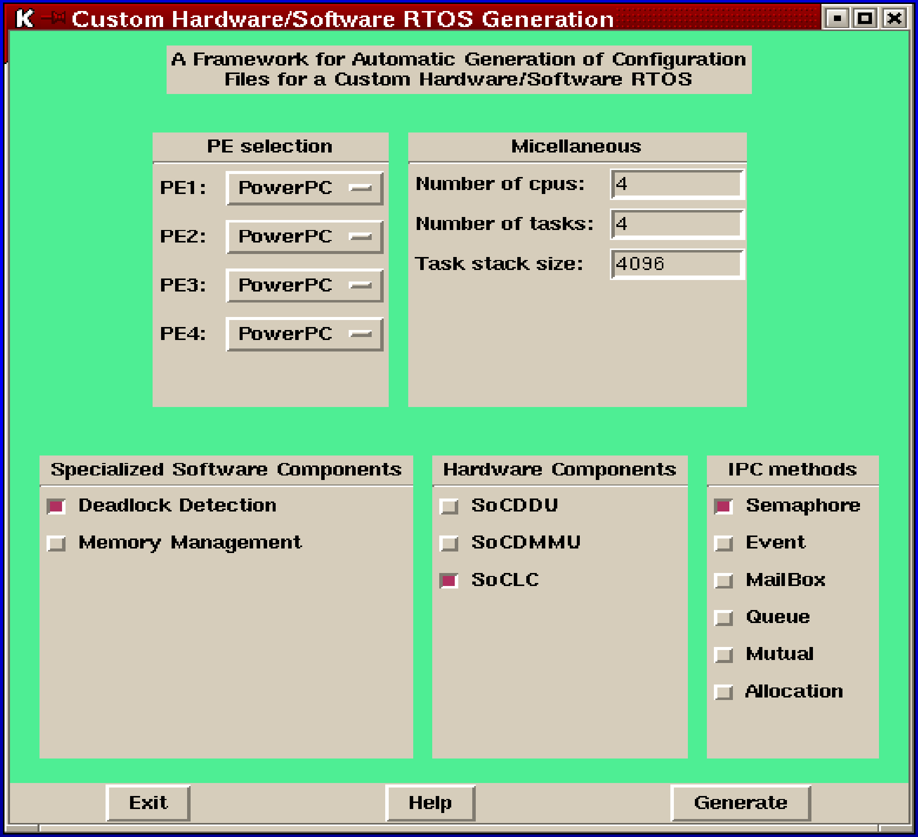
HW integration method

IPC module linking method

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



Custom Hardware/Software RTOS Generation

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

PE selection

PE1: PowerPC
PE2: PowerPC
PE3: PowerPC
PE4: PowerPC

Micellaneous

Number of cpus: 4
Number of tasks: 4
Task stack size: 4096

Specialized Software Components

- [✓] Deadlock Detection
- [ ] Memory Management

Hardware Components

- [ ] SoCDDU
- [ ] SoCDMMU
- [✓] SoCLC

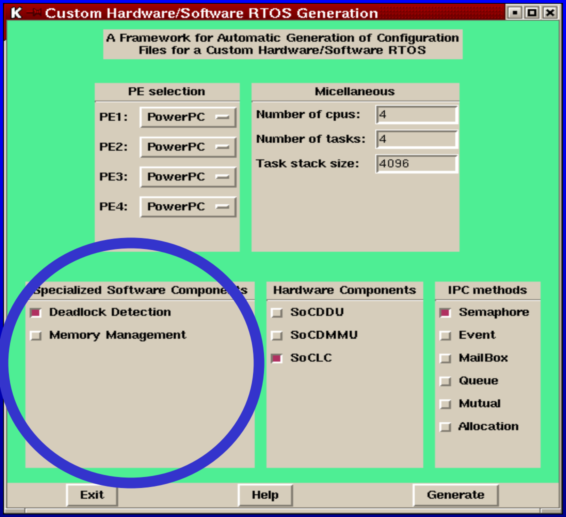
IPC methods

- [✓] Semaphore
- [ ] Event
- [ ] MailBox
- [ ] Queue
- [ ] Mutual
- [ ] Allocation

Implementation (3/8)

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



Custom Hardware/Software RTOS Generation

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

PE selection

PE1: PowerPC
PE2: PowerPC
PE3: PowerPC
PE4: PowerPC

Micellaneous

Number of cpus: 4
Number of tasks: 4
Task stack size: 4096

Specialized Software Components

[✓ Deadlock Detection]
Memory Management

Hardware Components

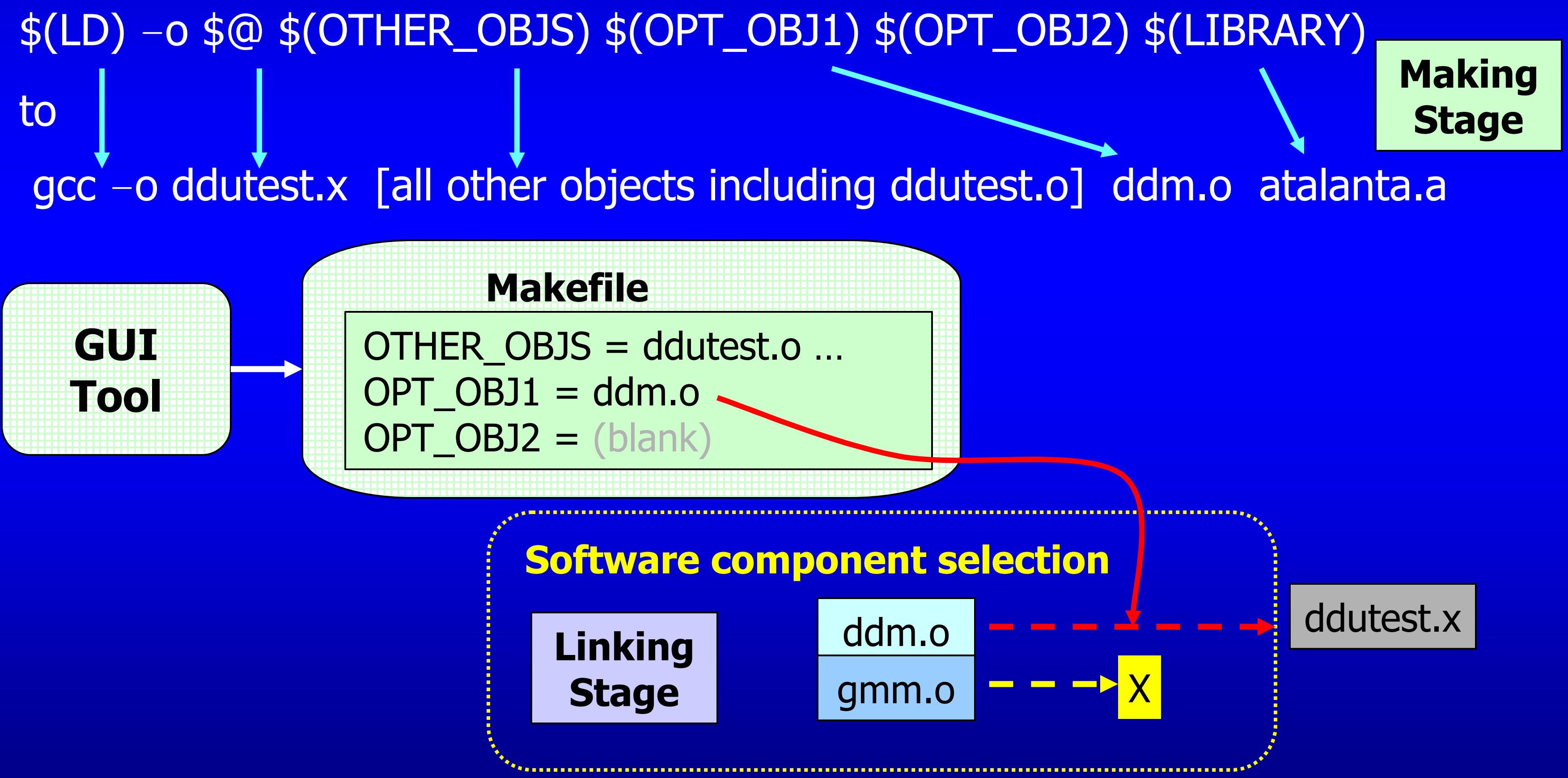
SoCDDU
SoCDMMU
[✓ SoCLC]

IPC methods

[✓ Semaphore]
Event
MailBox
Queue
Mutual
Allocation

Implementation (4/8)

1) Software module linking method (cont'd)

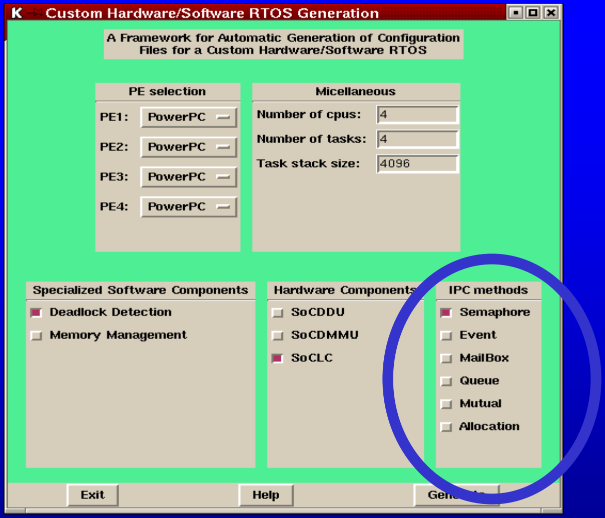


```
$(LD) –o $@ $(OTHER_OBJS) $(OPT_OBJ1) $(OPT_OBJ2) $(LIBRARY)
to
gcc –o ddutest.x [all other objects including ddutest.o] ddm.o atalanta.a
```

Implementation (5/8)

2) Inter-process communication module linking method

- Library function linking method (common)
- Implemented in User.h
- Used for the semaphore function in the example



Custom Hardware/Software RTOS Generation

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

PE selection

PE1:	PowerPC
PE2:	PowerPC
PE3:	PowerPC
PE4:	PowerPC

Micellaneous

Number of cpus:	4
Number of tasks:	4
Task stack size:	4096

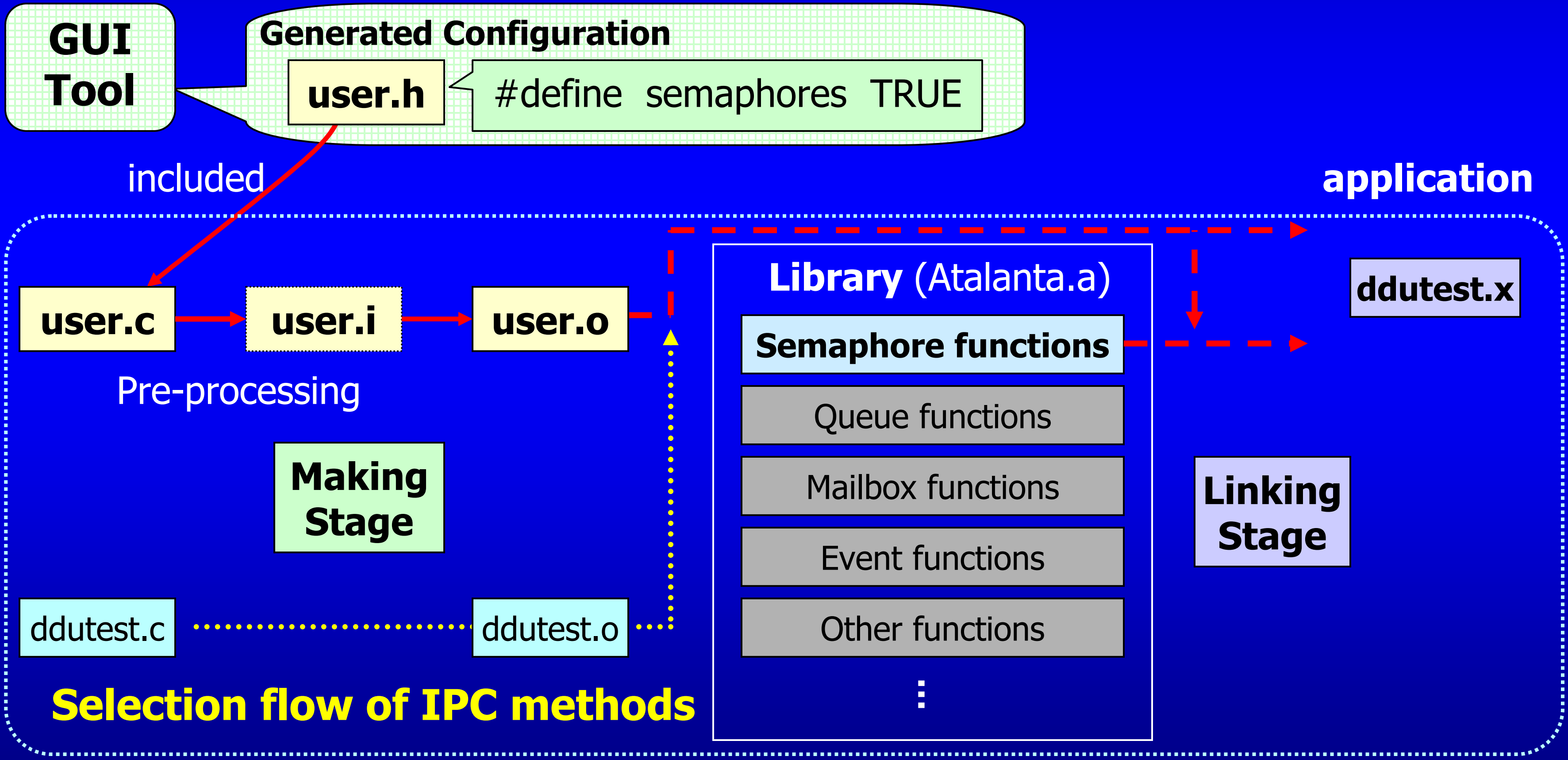
Specialized Software Components
- [✓] Deadlock Detection
- [ ] Memory Management

Hardware Components
- [ ] SoCDDU
- [ ] SoCDMMU
- [✓] SoCLC

IPC methods
- [✓] Semaphore
- [ ] Event
- [ ] MailBox
- [ ] Queue
- [ ] Mutual
- [ ] Allocation

Implementation (6/8)

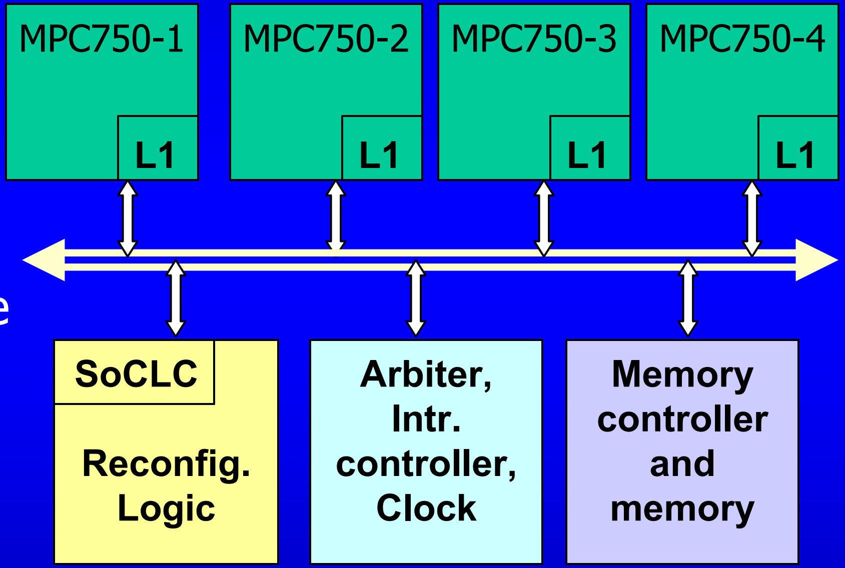
2) IPC module linking method (cont'd)



Implementation (7/8)

3) HW RTOS component integration method

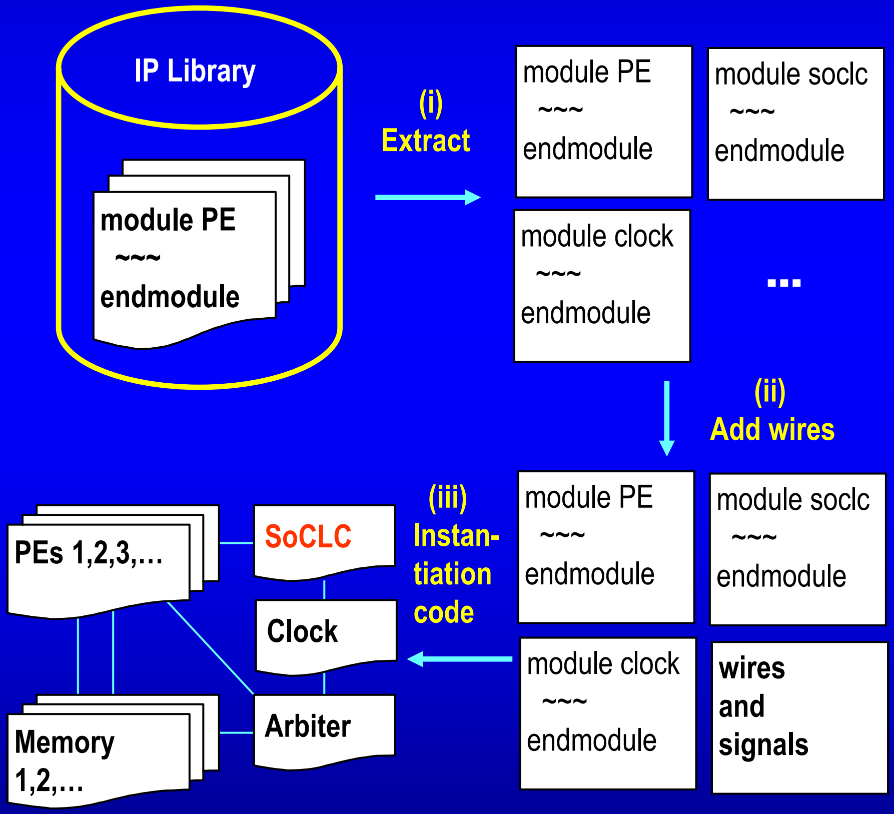
- Novel HW integration method
- Construct a Verilog header file
- Integrate user-selected HW RTOS components into the Base architecture
- Start with an SoCLC architecture description (an example with SoCLC)


MPC750-1
L1
MPC750-2
L1
MPC750-3
L1
MPC750-4
L1
SoCLC
Reconfig. Logic
Arbiter, Intr. controller, Clock
Memory controller and memory

Implementation (8/8)

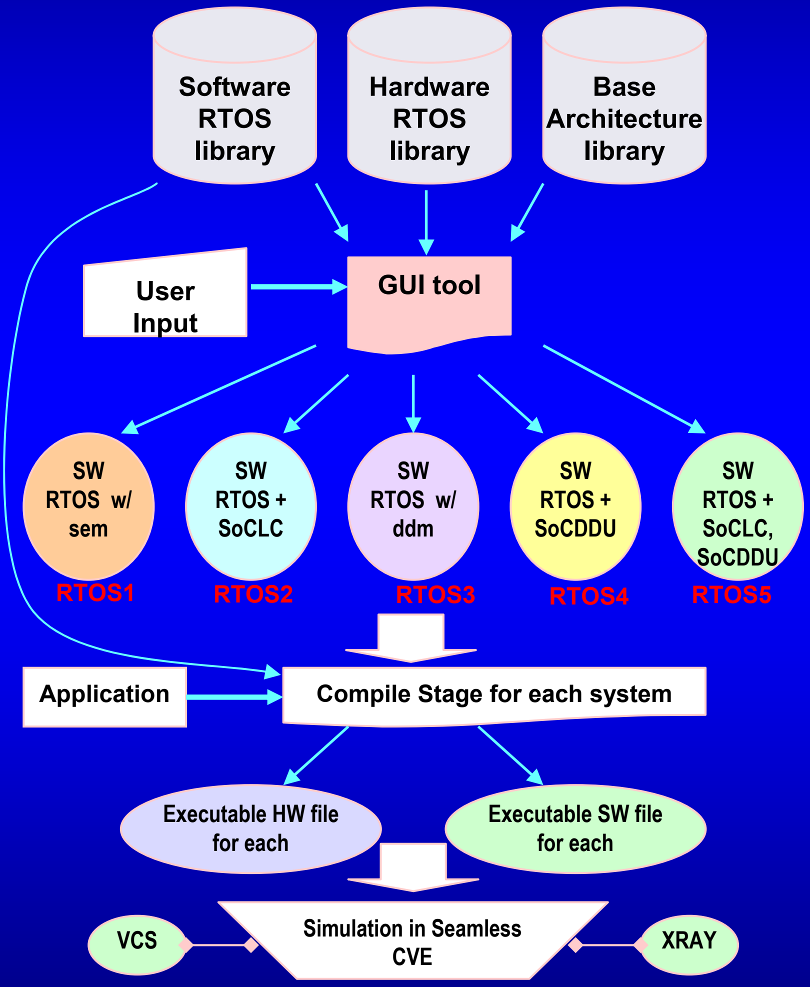
Verilog header file generation example

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


IP Library
module PE
~~~
endmodule
(i)
Extract
module PE
~~~
endmodule
module soclc
~~~
endmodule
module clock
~~~
endmodule
...
(ii)
Add wires
module PE
~~~
endmodule
module soclc
~~~
endmodule
module clock
~~~
endmodule
wires
and
signals
(iii)
Instan-
tiation
code
PEs 1,2,3,...
SoCLC
Clock
Arbiter
Memory
1,2,...
~~~

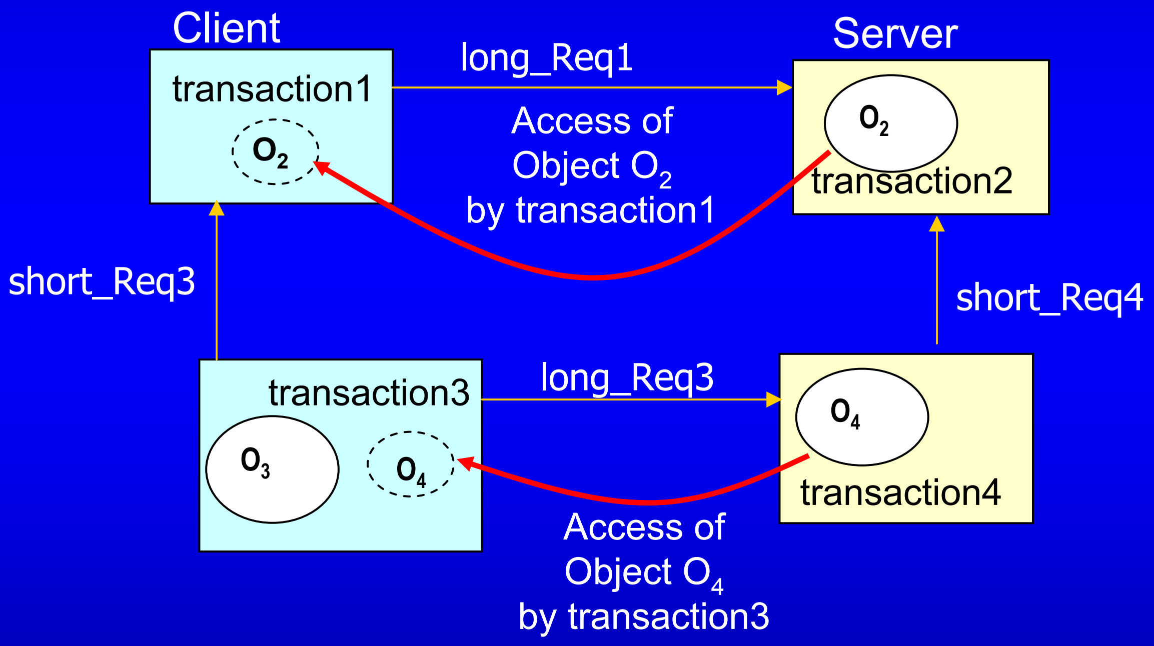
Experimental Setup

- 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



Experimental Results (1/4)

- Example 1: Database transaction application [1]



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

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

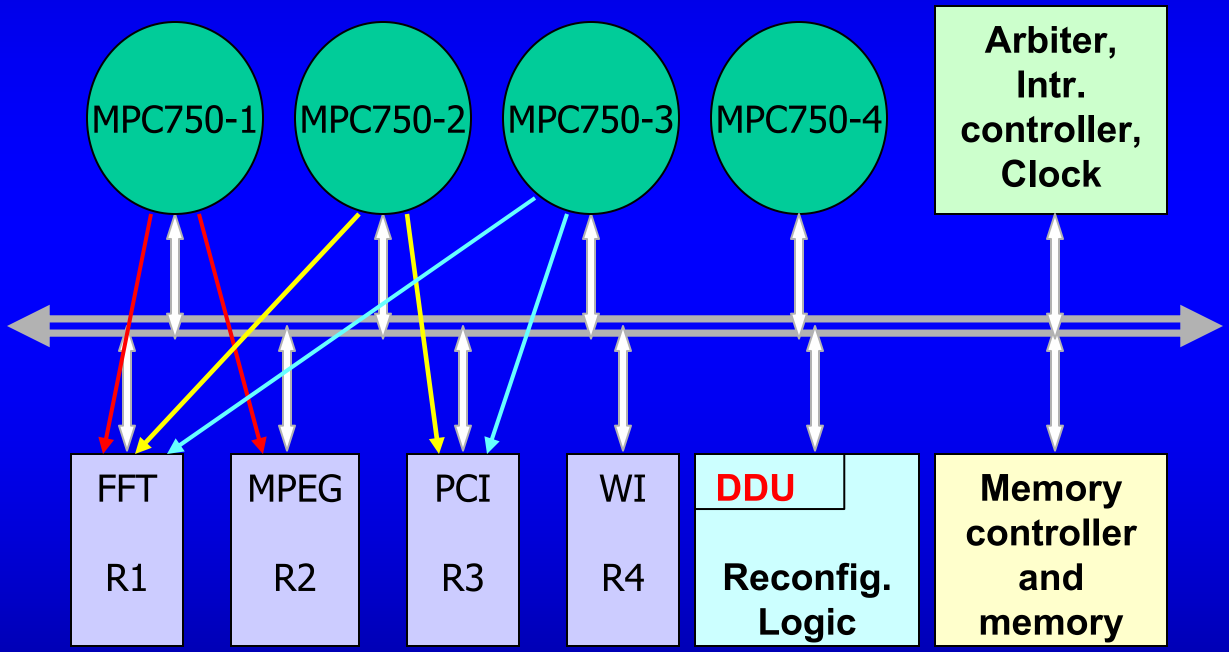
(clock cycles)	* Without SoCLC	With SoCLC	Speedup
Lock Latency	1200	908	1.32x
Lock Delay	47264	23590	2.00x
Execution Time	36.9M	29M	1.27x

* 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.

Experimental Results (3/4)

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



[3] S. Morgan, "Jini to the rescue," *IEEE Spectrum*, 37(4), pp 44-49, April 2000.

Experimental Results (4/4)

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

Method of Deadlock Detection	Software Algorithm	SoCDDU	Speedup
Detection Time Δ (clock cycles)	16928	2	8463x
Execution time up to deadlock detection	61131	44205	1.38x

[4] P. H. Shiu, Y. Tan and V. Mooney, "A novel parallel deadlock detection algorithm and architecture," *CODES '01*, pp.30-36, April 2001.

Hardware Area

Total area in	**SoCLC** (For 64 short CS locks + 64 long CS locks)		**SoCDDU** (For 5 Processors x 5 Resources)
Semi-custom VLSI	**7435** gates using TSMC 0.25μm standard cell library		**364** gates using AMI 0.3μm standard cell library
Xilinx XC4000E 4003EPC84	# Seq. logic	532	10
	# Other gates	9036	559

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