The System-on-a-Chip Lock Cache

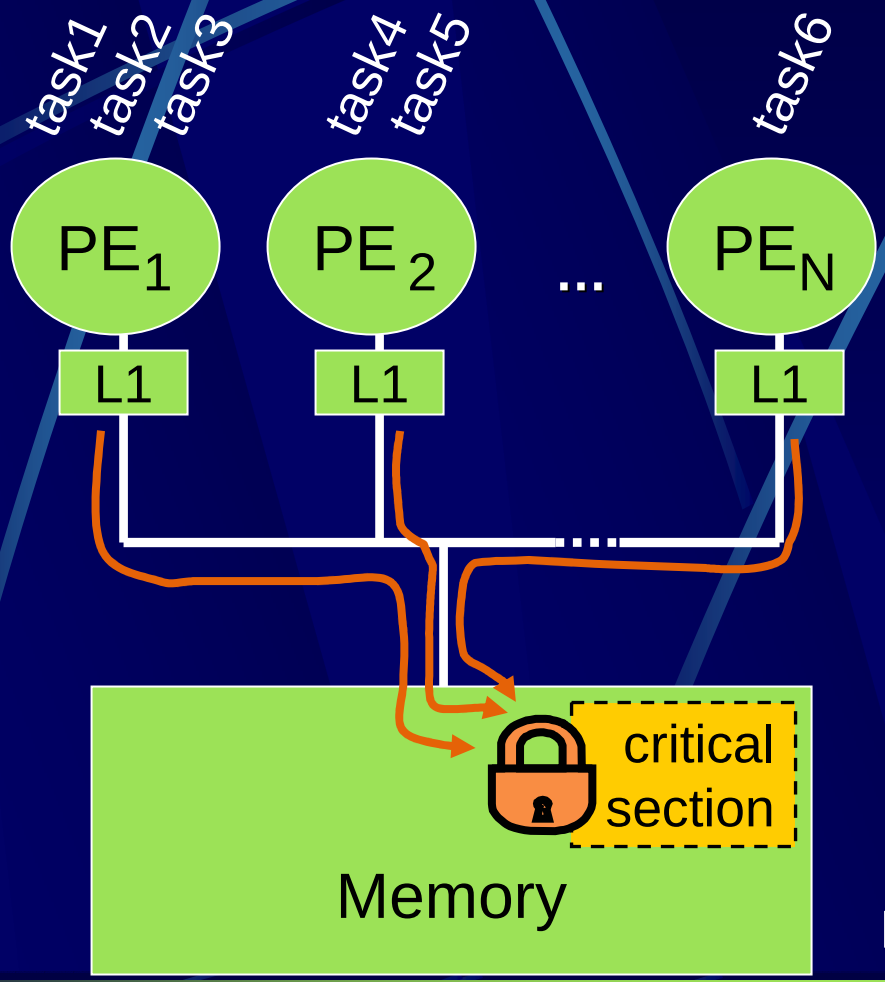
PhD Dissertation Defense
by

Bilge E. S. Akgul

Advisor: Prof. Vincent Mooney

School of Electrical and Computer Engineering, Georgia Institute of Technology
April 2004

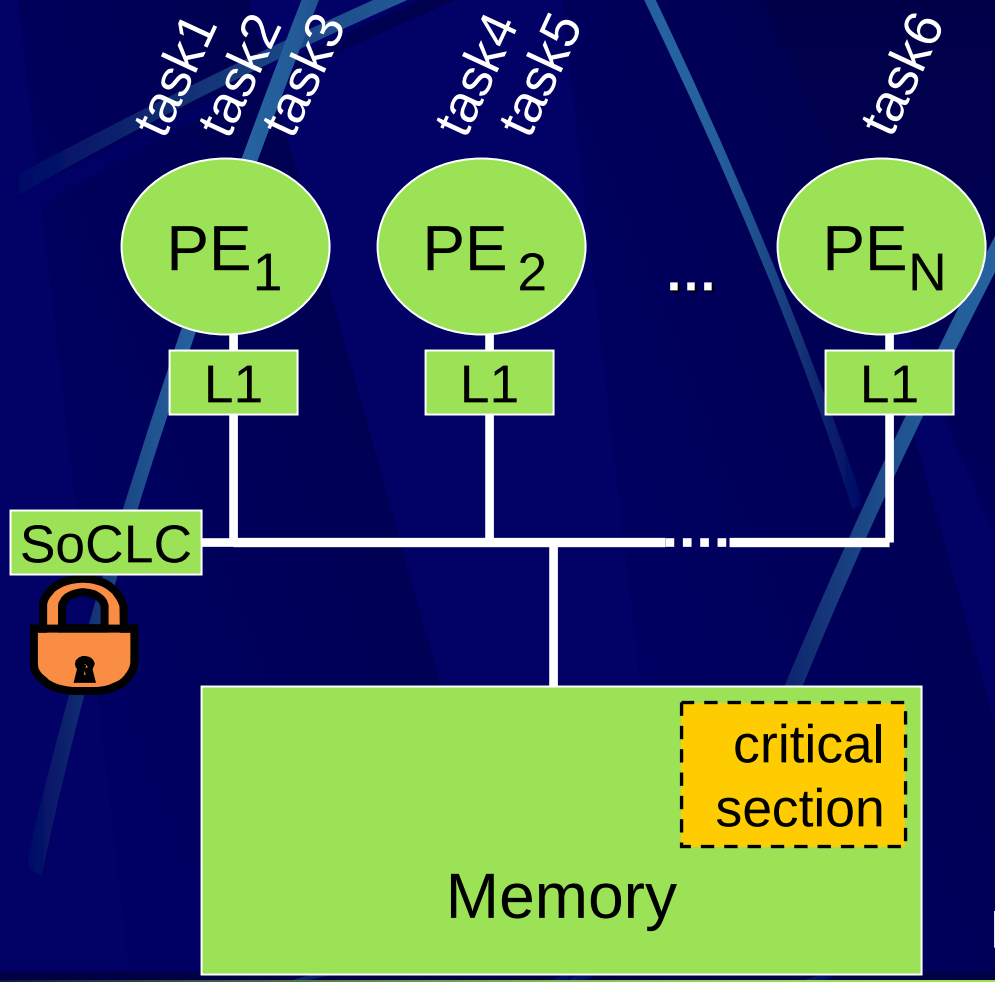
Aim: Effective Synchronization



- **A system-on-a-chip (SoC) may include**
  - Multiprocessors, on-chip shared memory, peripherals and other hardware components
  - Multi-tasking application with a real-time operating system (RTOS)
- **Many shared-data structures cause contention**

PE: processing element

Aim: Effective Synchronization



Solution: move lock variables to a specialized hardware logic

SoC Lock Cache (SoCLC)

PE: processing element

What is New in SoC?

- Previously, PEs and L2 memory on separate chips
- Specialized hardware would also be on a separate chip
  - → Have to communicate to specialized hardware via pins, i.e., at a speed similar to the PCB clock instead of processor clock
- With multiprocessor SoC, can add specialized hardware assist logic at or close to the processor speed without incurring cost of additional pins on package

Outline

- Background and Previous Work
- Basic SoCLC Approach
- Priority Inheritance Support in SoCLC
- PARLAK Tool
- Experimental Results
- Conclusion

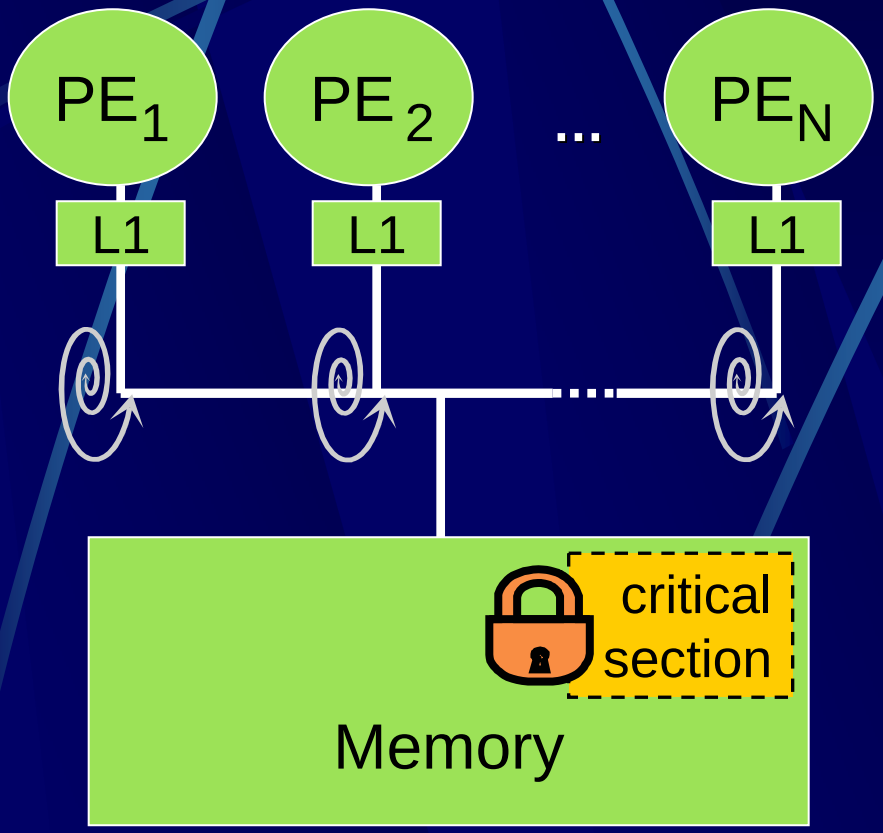
Locking/Unlocking

```
Lock(lock);

  access_critical_section();

Unlock(lock);
```

```
Lock(lock) {
  while( test_and_set(lock) )
        ;   // spin if failed
}
```

- Locking operation must be atomic
- Hardware support for atomicity
- Test-and-set atomic primitive used
  - Spin-lock

Problem



- Busy-wait problem: PEs spin on memory bus
- Cache invalidations – hold cycles
- Bandwidth consumption

Previous Work

- Hardware based synchronization mechanisms
  - Using cache, additions to processor core
- Software based synchronization algorithms
  - Make use of synchronization primitives
    - test_and_set, fetch_and_increment, compare_and_swap, etc.

Hardware Based Solutions

- Cache based synchronization
  - Cache based lock (CBL) [RL96]
  - Queue on lock bit (QOLB) [KBG97]
- Speculative approaches
  - Transactional memory [HM93]
  - Speculative lock elision [RG01]
  - Speculative synchronization unit [MT01]
  - Speculative lock reordering [RS02]

Hardware Based Solutions (cont.)

- Require special cache/cache protocol, cache to cache transfer
- Extend processor core with special hardware
- Not applicable to general purpose processors

Software Based Solutions

- Spin-on-read [RS84]
- Exponential/proportional back off inserted into the spin loop [And90]
- Ticket locks [Lam74, RK79]
- Array based locks [And90, GT90]
- Queue based locks
  - MCS locks [MCS91]
  - LH and M locks [MLH94]

MCS Locks

```
 1 type MCSnode{
 2       MCSnode *next;
 3       WORD lock;
 4 };
 5
 6 MCSnode* lock = NULL; //initialize lock to NULL
 7
 8 MCS_LockAcquire(MCSnode** lock, MCSnode* mynode)
 9 {
10       MCSnode *prevnode;
11       mynode->next = NULL;                    //mynode becomes the last in queue
12       prevnode = fetch_and_store(lock, mynode); //atomically do the following:
13                                                 //prevnode = lock
14                                                 //lock = mynode
15       IF prevnode != NULL THEN  //if lock is busy
16            mynode->lock = TRUE;
17            prevnode->next = mynode;
18            while(mynode->lock); //spin until lock is released
19       ENDIF
20 }
21
22 MCS_LockRelease(MCSnode** lock, MCSnode* mynode)
23 {
24       IF mynode->next == NULL THEN
25            IF compare_and_swap(lock,mynode,NULL) THEN
26                 return;
27            ENDIF
28            while(mynode->next==NULL);
29       ENDIF
30       mynode->next->lock = FALSE;
31 }
```

MCS Locks (cont.)

- Require fetch_and_store and compare_and_swap primitives
- Scale well for high contention but have constant software overhead (poor for low contention)

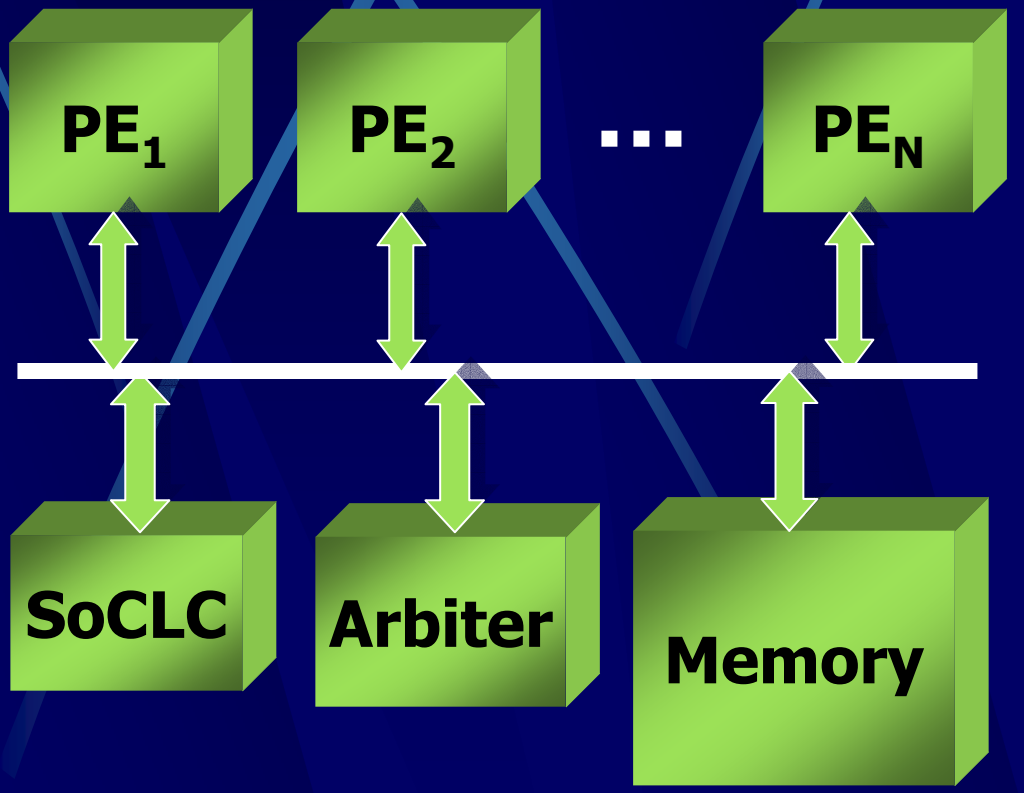
Outline

- Background and Previous Work
- Basic SoCLC Approach
- Priority Inheritance Support in SoCLC
- PARLAK Tool
- Experimental Results
- Conclusion

Our Approach

Custom hardware: SoC Lock Cache (SoCLC)

SoCLC provides synchronization among processors



Our Approach

- Simple hardware mechanism: SoCLC
- No modifications/extensions to processor core or to caches
- No special instructions or atomic primitives
- Easily integrated as an intellectual property (IP) block into the SoC
- Hardware interrupt triggered notification

Software

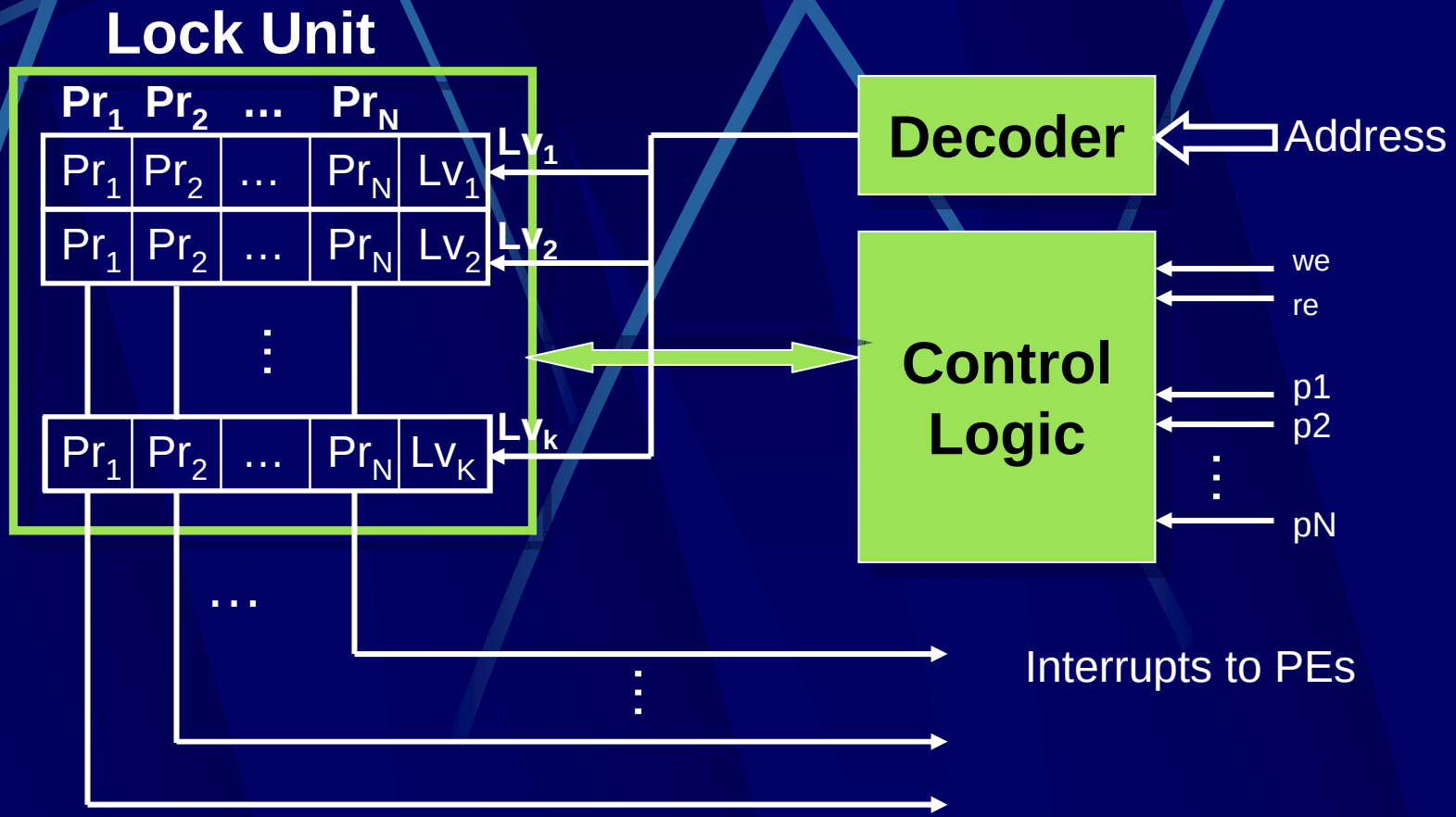
Old method

```
test:   LL    r2, (r1)      ; read lock value
        ORI   r3, r2, 1     ; r3 = 1
        BEQ   r3, r2, test  ; test again if lock = 1
        SC    r3, (r1)      ; attempt to lock
        BEQ   r3, 0, test   ; test again if failed
```

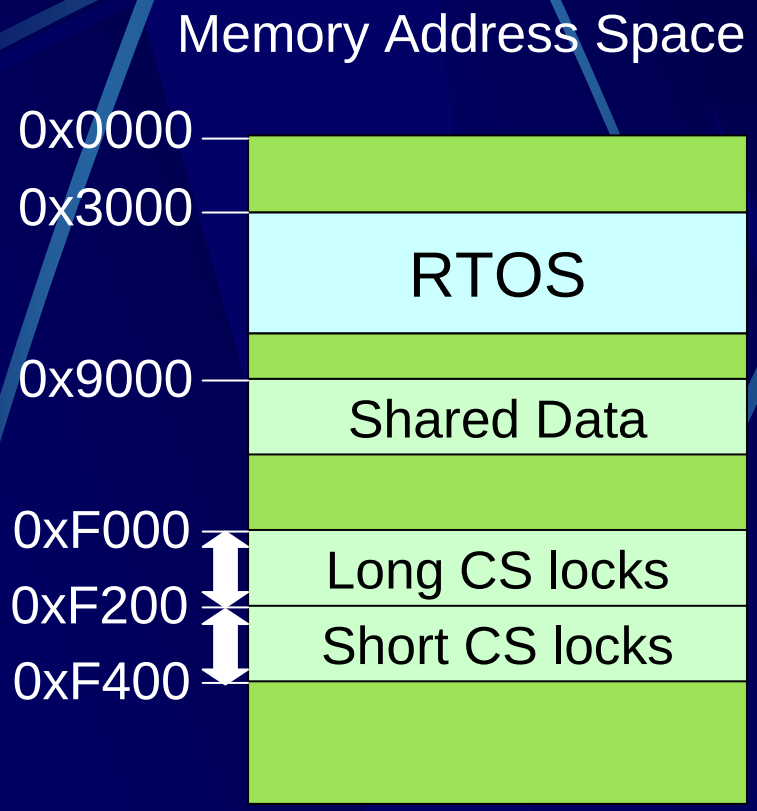
Our method

```
        LW    r2, (r1)      ; read lock value
        BEQ   r2, 1, sleep  ; test again if lock = 1

sleep:  B     sleep         ; sleep until interrupt occurs
```

Hardware


Lock Unit
Pr1 Pr2 ... PrN
Pr1 Pr2 ... PrN Lv1
Pr1 Pr2 ... PrN Lv2
Pr1 Pr2 ... PrN LvK
Lv1
Lv2
Lvk
Decoder
Address
Control Logic
we
re
p1
p2
pN
Interrupts to PEs

Long/Short CSes



- Memory mapped
- Distinguish lock variables according to the critical section lengths:
  - Long
  - Short
- Support preemption of tasks when necessary

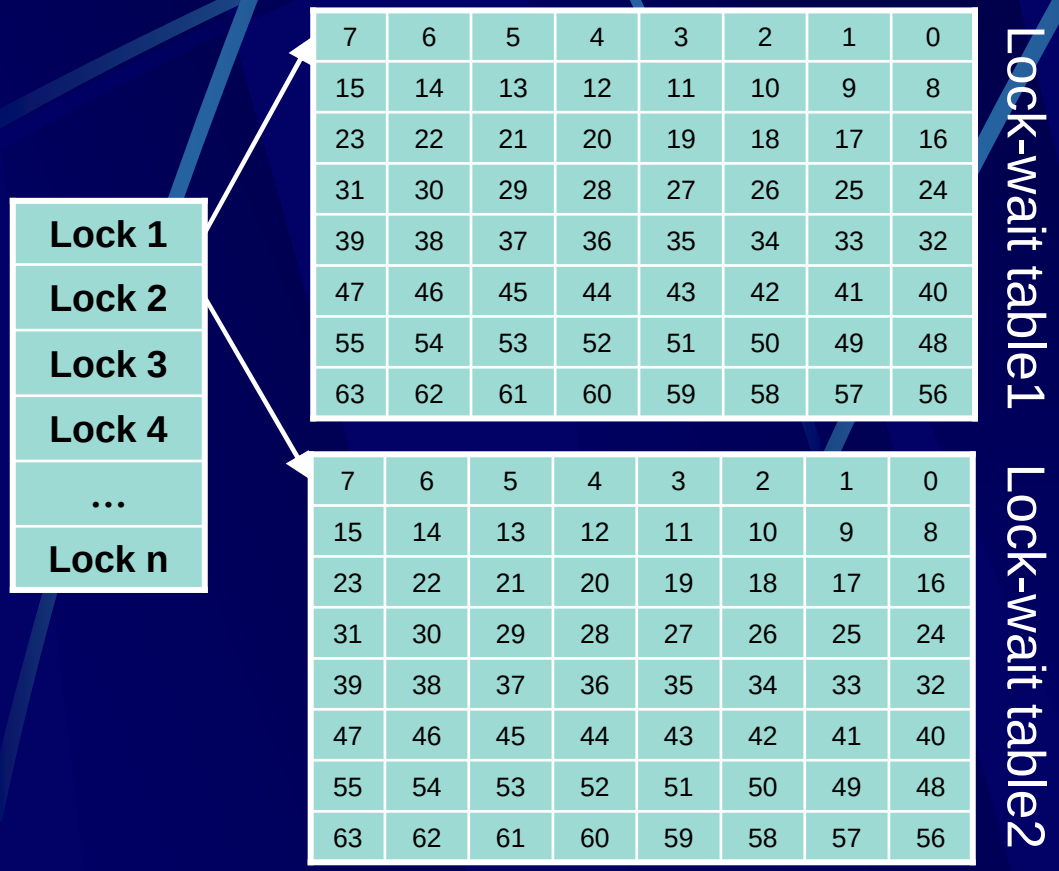
Atalanta

- Multiprocessor RTOS [DBM02]
  - Multitasking
  - Preemptive, priority based scheduler
  - Uses simple spin-lock
  - Interprocess communication/synchronization
    - Semaphores, mutexes, mailboxes, queues
- Ported to ARM and PowerPC processors

Preemption

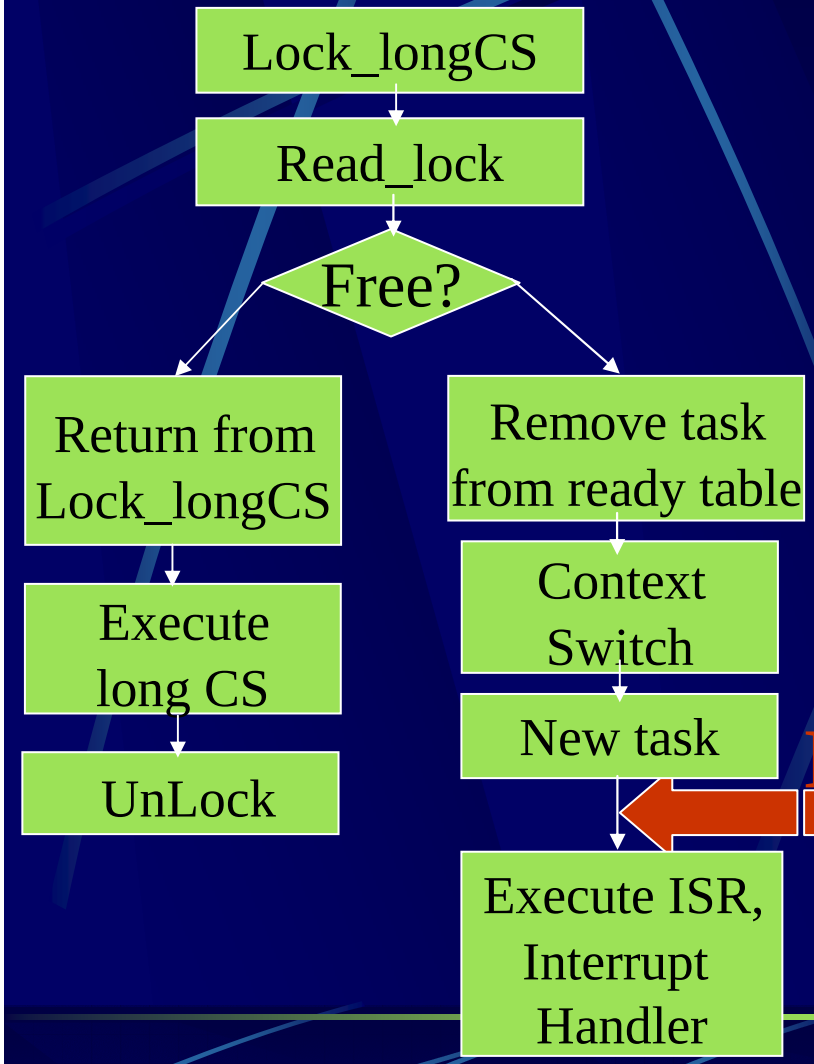
- Preemption can improve performance
  - For long critical sections
  - Yield the CPU resources, context switch
  - Enabling other tasks to do useful work instead of waiting
- Support preemption of tasks with Atalanta real-time operating system (RTOS)

Preemption

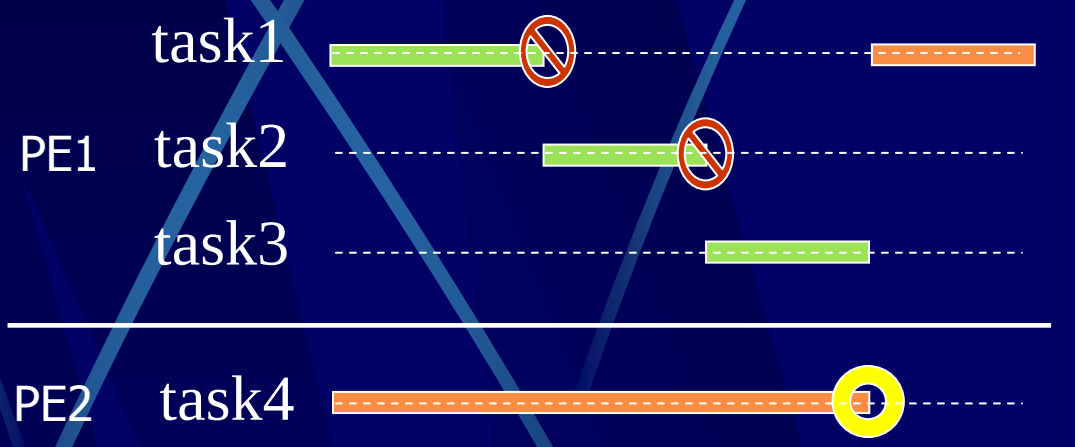


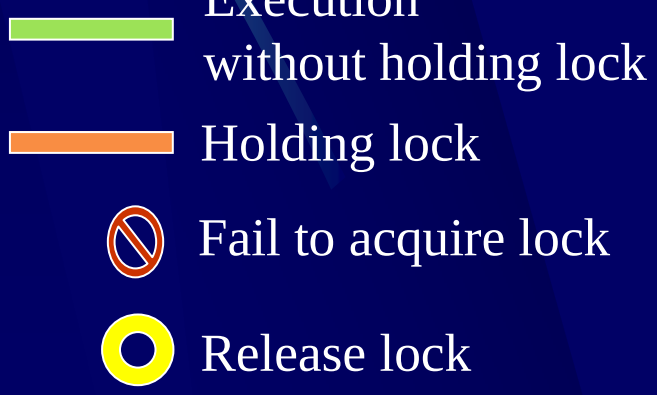
- RTOS saves the states of blocked tasks
- When interrupt occurs, the highest priority task is chosen to acquire the lock
- Context switch to new task

Long CS Locks – Preemption

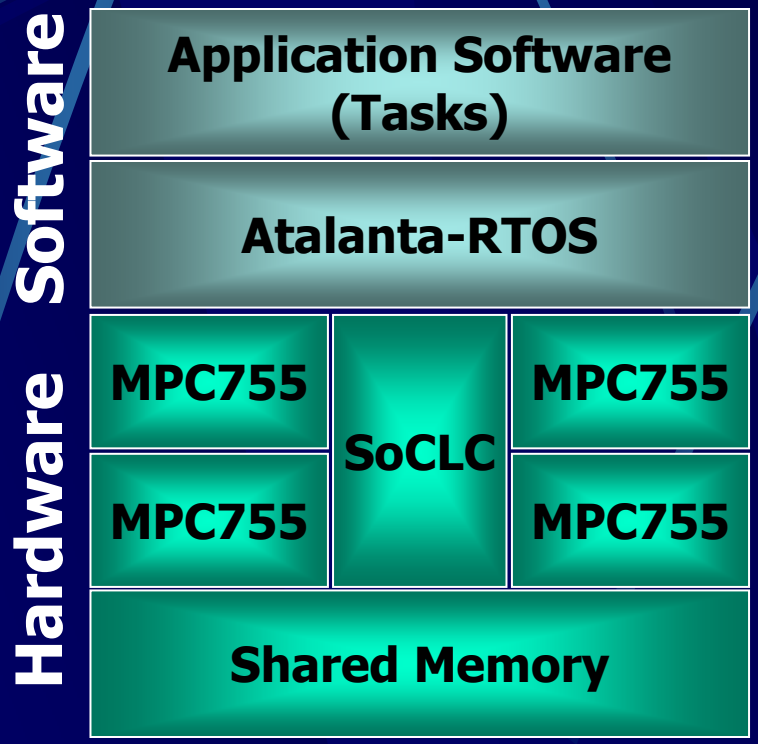


Interrupt





Architecture



- Multiple application tasks
- Atalanta-RTOS
- Multiprocessor setup with MPC750s
- Seamless CVE
- SoCLC provides lock synchronization among processing elements

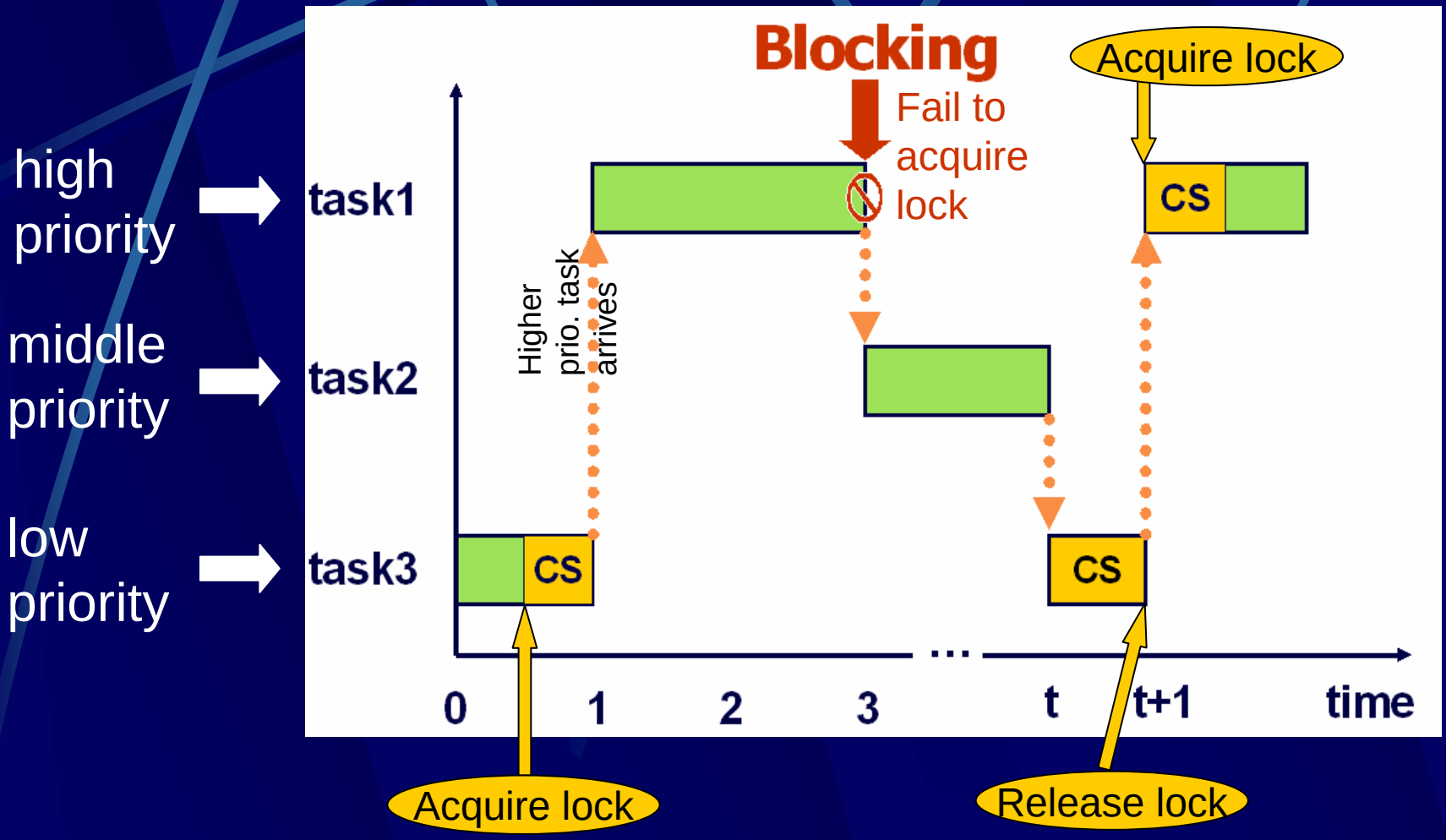
Outline

- Background and Previous Work
- Basic SoCLC Approach
- Priority Inheritance Support in SoCLC
- PARLAK Tool
- Experimental Results
- Conclusion

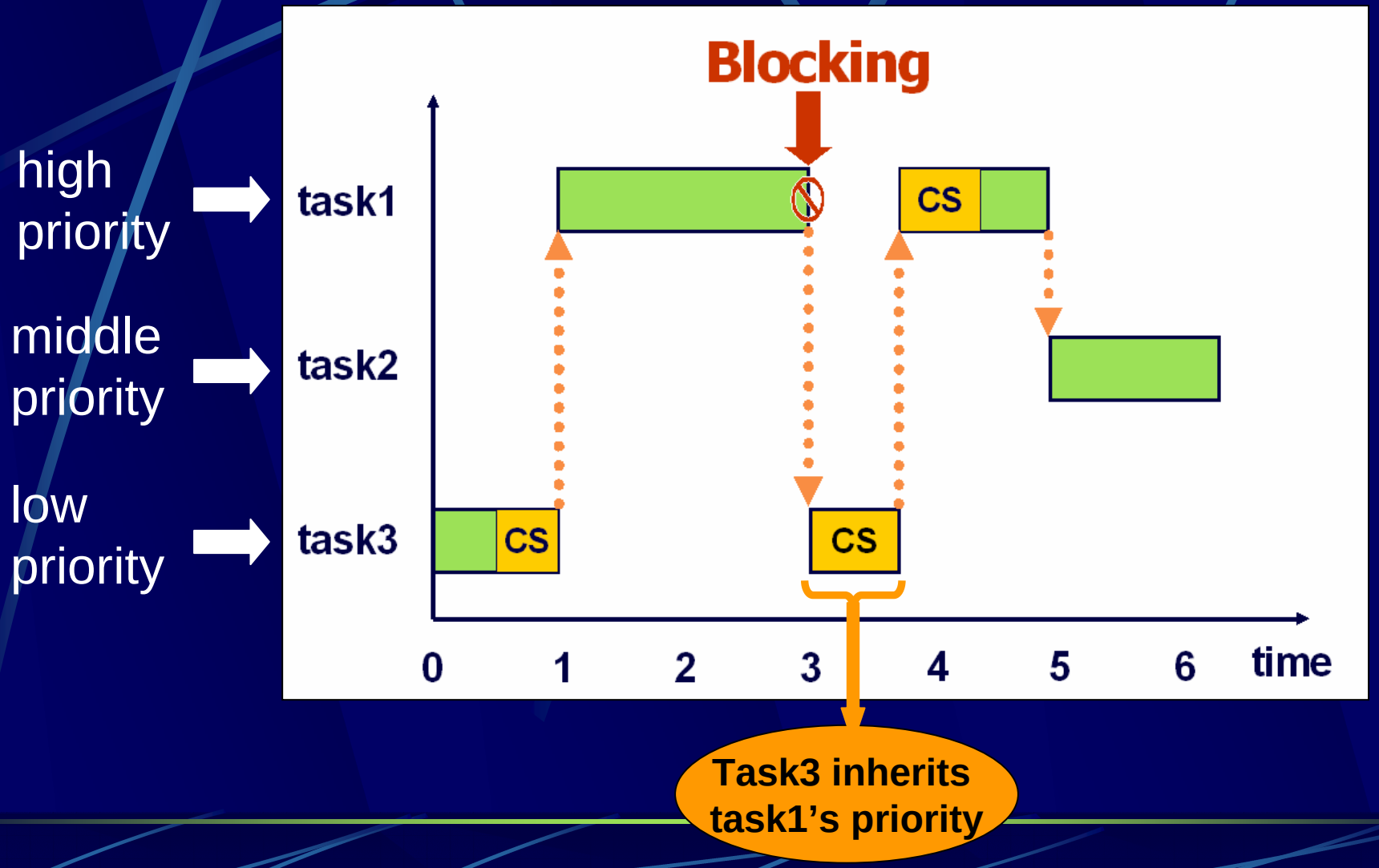
Motivation

- Real-time operating system (RTOS) with a priority-based scheduler
  - Assign a higher priority to a time-critical task with hard real-time requirement
- Problem: If tasks with different priorities share resources ➔ priority inversion may occur
  - May miss real-time deadlines

Priority Inversion



Priority Inheritance



Motivation

- Priority inheritance in RTOS
  - May affect real-time performance of application tasks
- Objective: To implement hardware support for **priority inheritance** (via SoCLC) to help RTOS be more predictive and efficient

Priority Inheritance Protocols

- Sha, Rajkumar and Lehoczky (`88)
- Prevents unbounded blocking
  - Running task inherits the highest dynamic priority of all the tasks it blocks
- List of blocked tasks must be saved in a priority queue for each CS
- Maximum blocking time (due to a lower priority task):
  - On each lock, at most once
  - Length of one CS (executed in a lower priority task)
- Still problem: deadlock, chained blockings

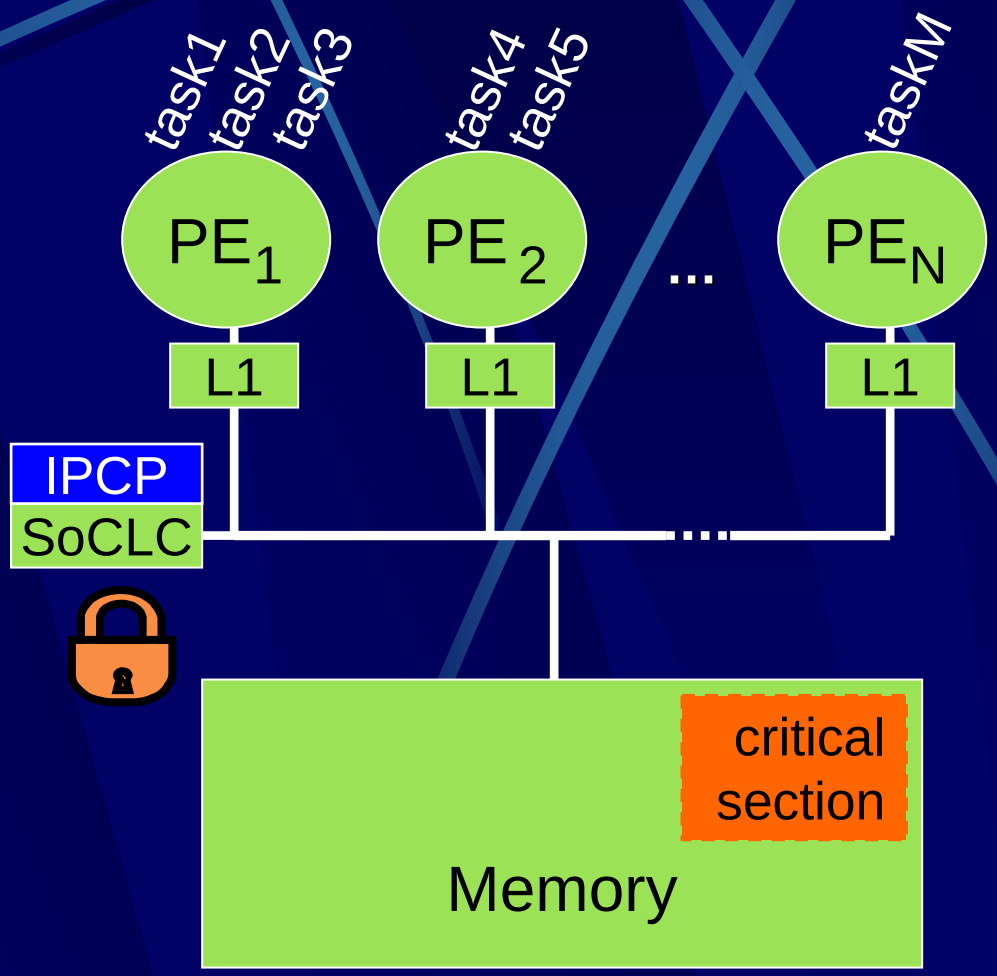
Priority Inheritance Protocols

- Sha, Rajkumar and Lehoczky (`90)
- Baker (`91)
- Klein and Ralya (`90)
- Prevent deadlocks and chained blockings
  - Implies that once a process locks its first CS, it can never be blocked by lower priority tasks
- Original priority ceiling protocol (OPCP)
- Immediate priority ceiling protocol (IPCP)
- Each task has a static (default) priority

Related Work

- Operating system coprocessors
- Implement various real-time functions in hardware
  - Real Time Unit (RTU), `96
    - Many RTOS functions in hardware
  - Ada TAsking Coprocessor (ATAC), `95
    - It has its own instruction set
    - Implements real-time part of Ada (also Ada rendezvous with basic priority inheritance) in hardware

SoCLC Priority Inheritance

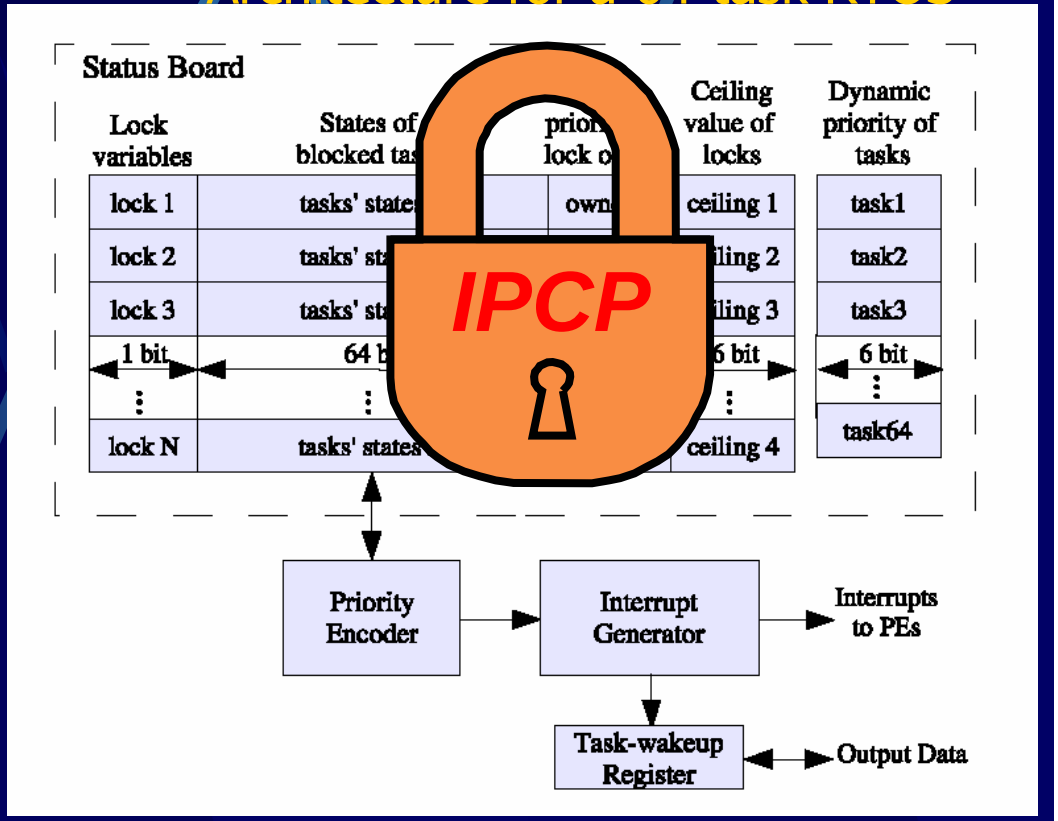


PE: processing element

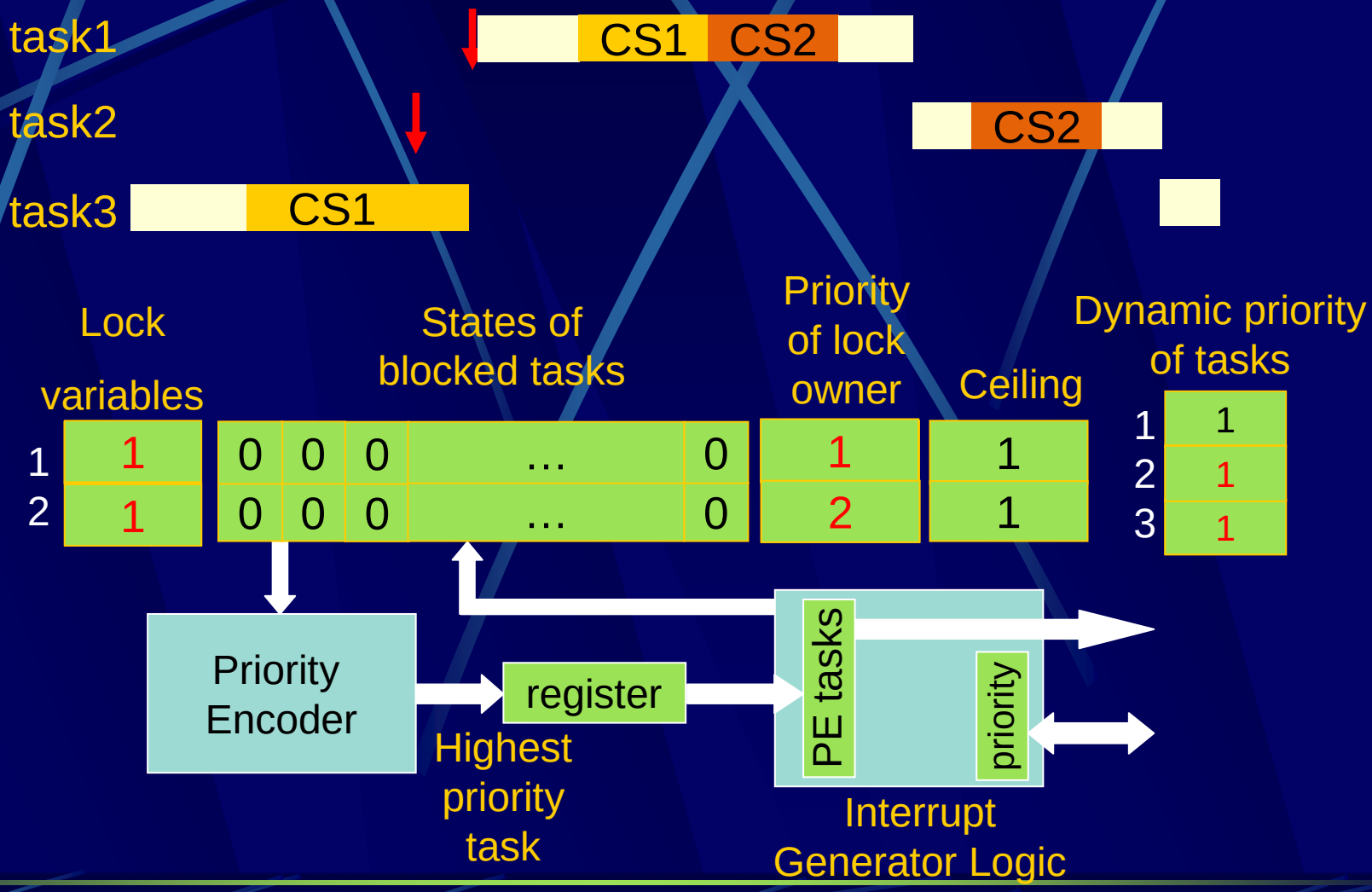
SoCLC Priority Inheritance

- SoCLC with IPCP
- Ceiling values for every CS used in each task is specified
  - SoCLC needs the ceiling values of locks
- Task priorities are updated by SoCLC in hardware
- Blocked tasks are monitored by SoCLC

Priority Inheritance Hardware Architecture for a 64-task RTOS

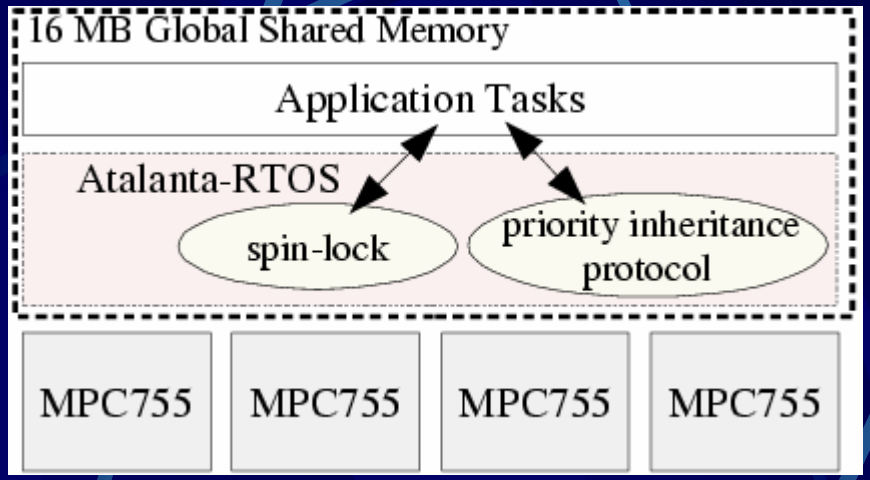


Example

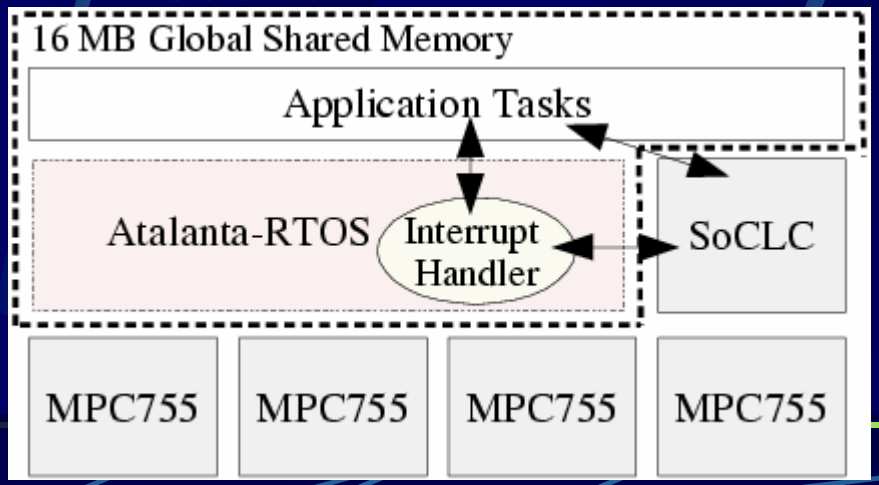

task1
CS1
CS2
task2
CS2
task3
CS1
Lock variables
States of blocked tasks
Priority of lock owner
Ceiling
Dynamic priority of tasks
1
2
1
1
0 0 0 … 0
0 0 0 … 0
1
2
1
1
1
1
2
3
1
1
1
Priority Encoder
register
Highest priority task
PE tasks
priority
Interrupt Generator Logic

Experimental HW/SW Architecture

Without SoCLC



With SoCLC

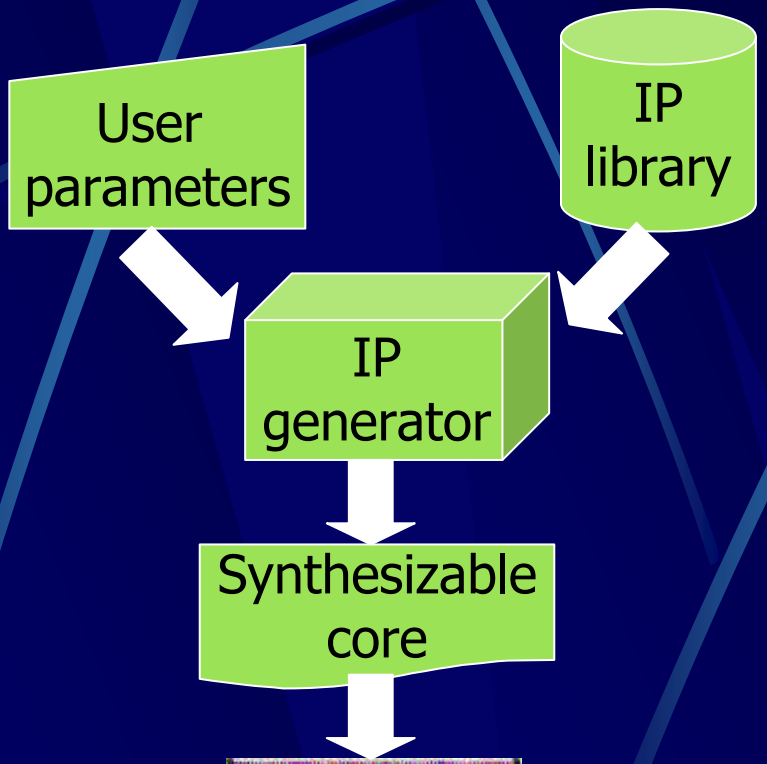


- Multiple application tasks and Atalanta-RTOS
- Multiprocessor setup with MPC750s on Seamless CVE (from Mentor Graphics)
- Atalanta-RTOS
- SoCLC provides lock synchronization among processing elements

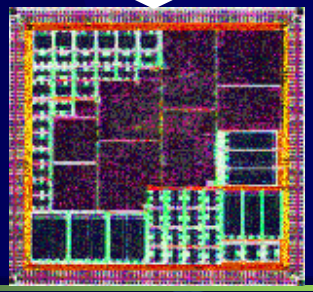
Outline

- Background and Previous Work
- Basic SoCLC Approach
- Priority Inheritance Support in SoCLC
- PARLAK Tool
- Experimental Results
- Conclusion

IP/core Based SoC Design



Many millions of gates on a single chip



- SoC includes multiple processors, memory and bus system, peripherals and specialized custom blocks
- Complex chips → complex designs
- Reusability, parametrizability, reconfigurablitiy, integration, engineering effort, time-to-market
- Solution: IP-generator tools automate generation of synthesizable IP blocks and system components

PARLAK*: Parametrized Lock Cache Generator

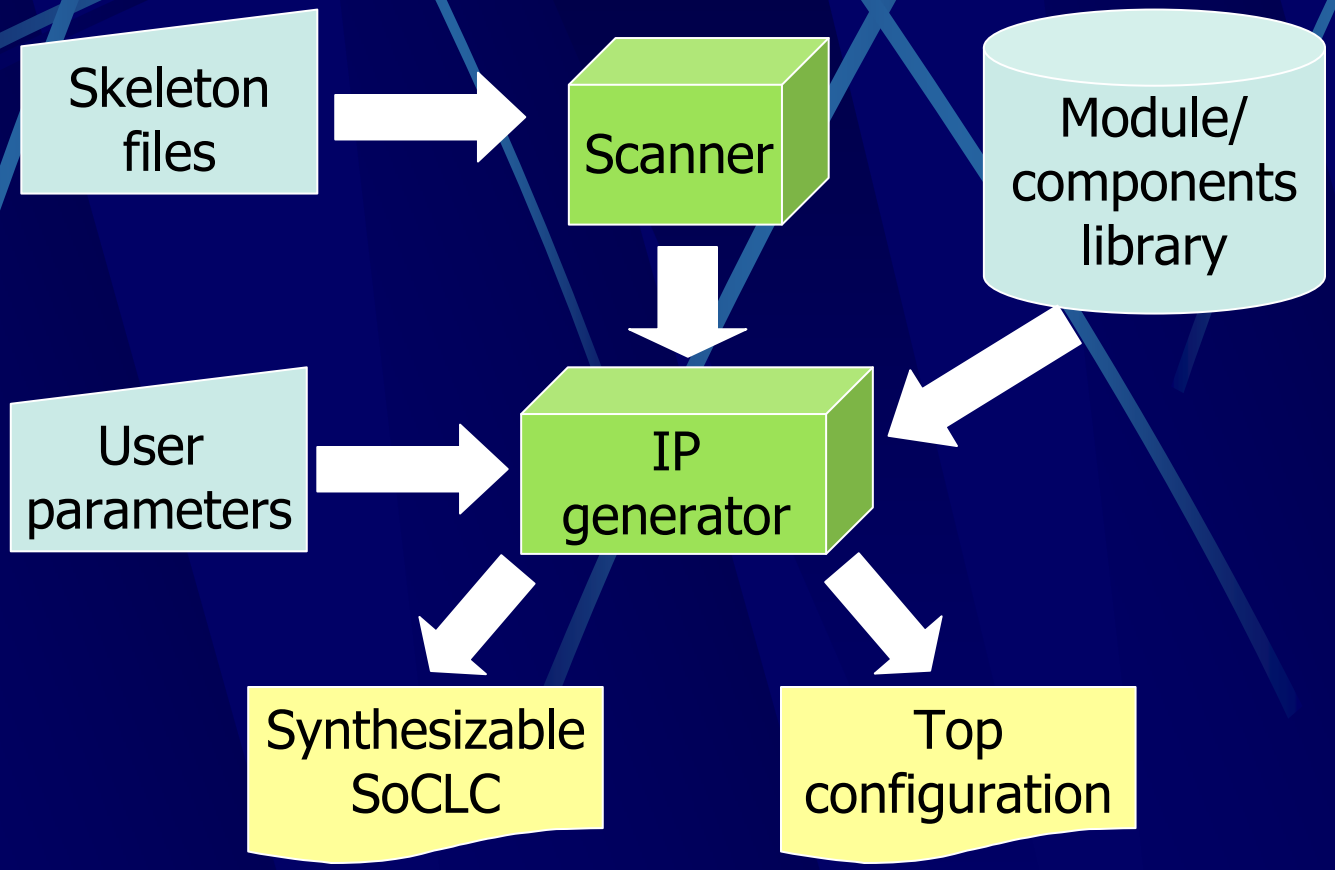
- PARLAK is an IP-generator tool for SoCLC
  - Build custom logic (prior to chip fabrication)
  - May reconfigure SoCLC (after chip fabrication) on the reconfigurable logic of the SoC
- Generates customized, user specified versions of SoCLC
- Designs generated using PARLAK were synthesized by Design Compiler (Synopsys)

* Parlak means bright in Turkish

PARLAK

- Building blocks:
  - Input parameters: number of short critical section locks, number of long critical section locks, number and types of processors (MPC755 and ARM9tdmi)
  - Skeleton files (in Verilog HDL): signal, process and module descriptions (independent from the input parameters)
  - Library of modules/components

PARLAK


Skeleton files
Scanner
Module/ components library
User parameters
IP generator
Synthesizable SoCLC
Top configuration

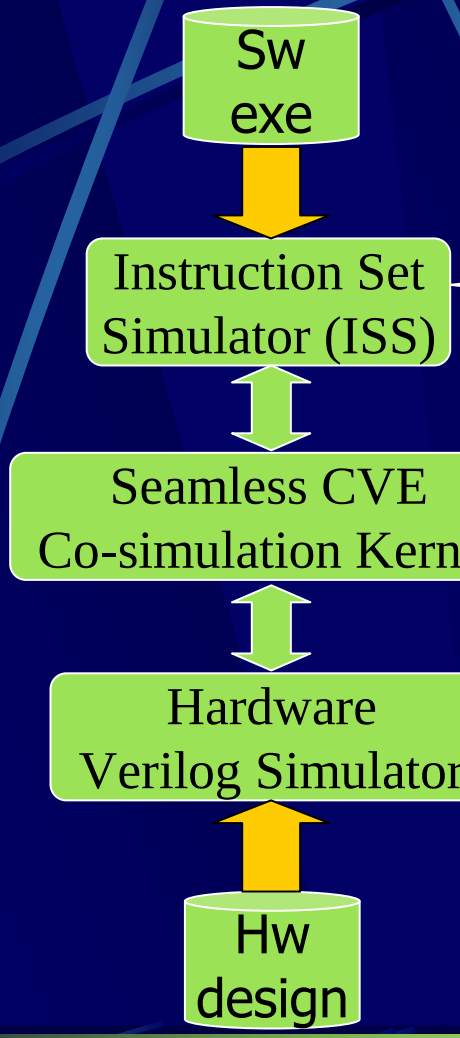
Outline

- Background and Previous Work
- Basic SoCLC Approach
- Priority Inheritance Support in SoCLC
- PARLAK Tool
- Experimental Results
- Conclusion

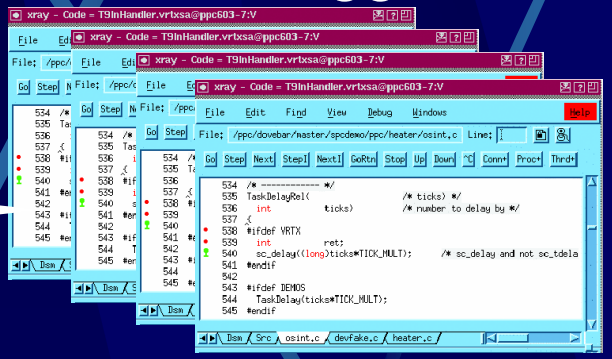
Experimental Results

- Basic SoCLC
  - Microbenchmark simulations
  - Database application
- SoCLC with priority inheritance
  - Synthetic robot application
- PARLAK synthesis results

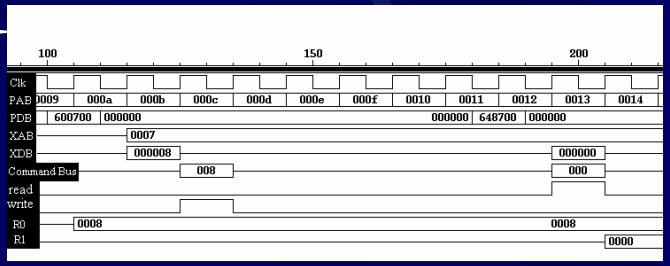
Experimental Platform



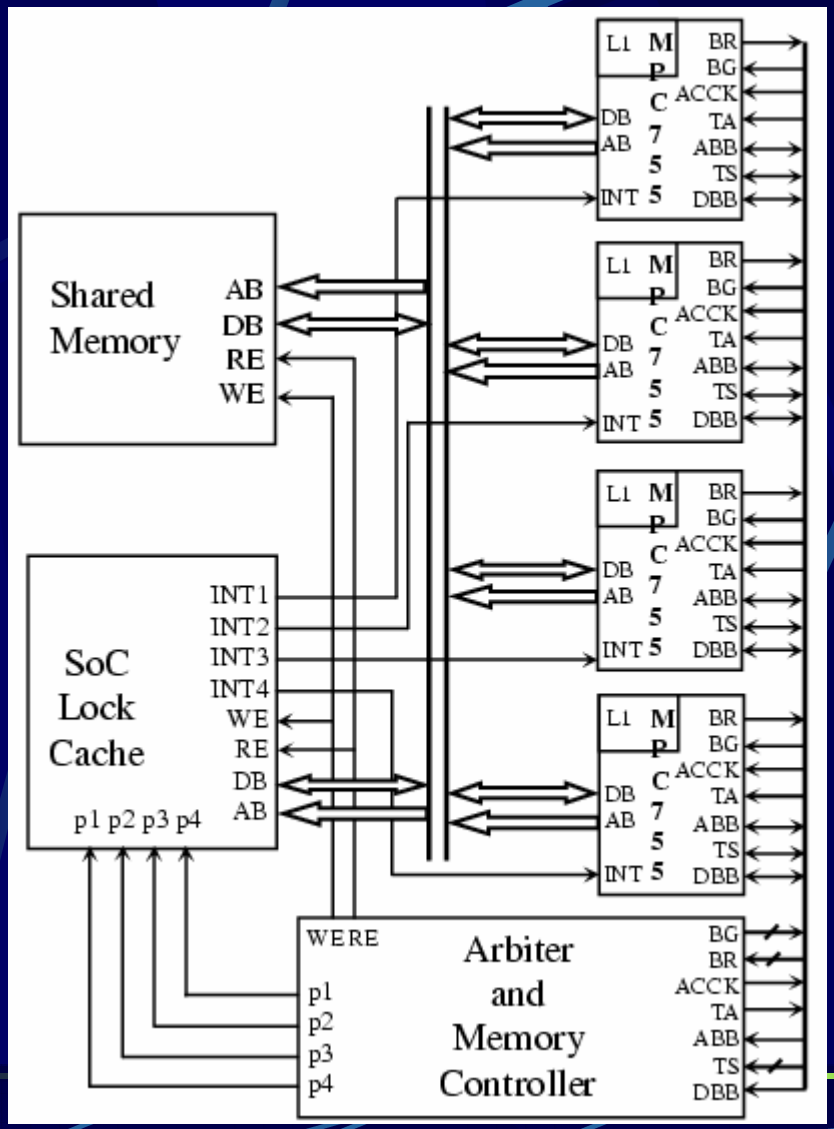
XRAY debugger interface



VCS hw simulator

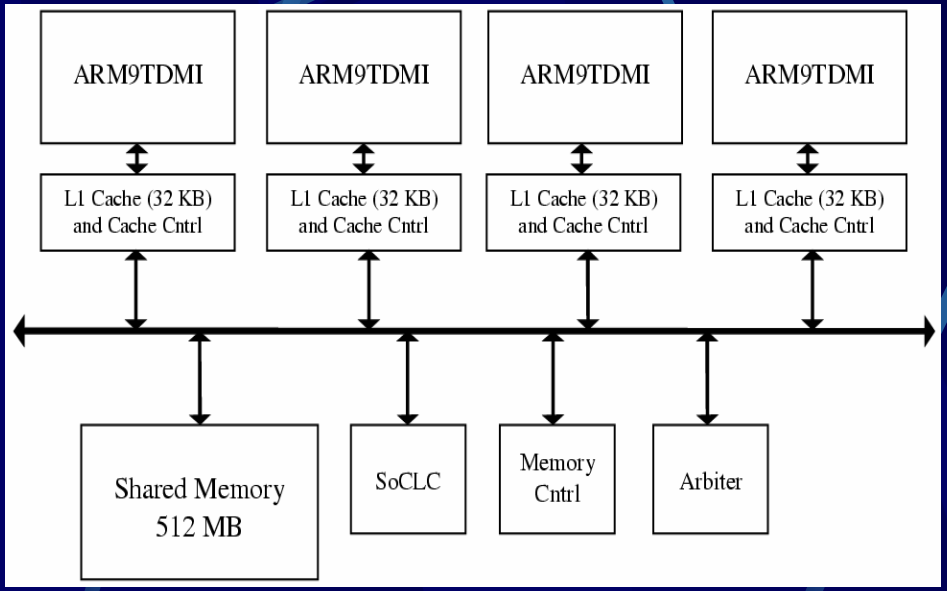


PowerPC Platform



- System bus clk: 100 MHz
- MPC755 Specifications:
  - I-$ size: 32 KB
  - D-$ size: 32 KB
  - Cache protocol/policy: MEI, write-back, insert-in-cache
  - Cache line size: 32 B
  - Global shared memory: 16MB

ARM Platform



- System bus clk: 10 MHz
- ARM Specifications:
  - Unified I-$ + D-$ size: 32 KB
  - Cache protocol/policy: MESI, write-back, insert-in-cache
  - Cache line size: 16 B
  - Global shared memory: 512MB

Experimental Results

- Basic SoCLC
  - Microbenchmark simulations
  - Database application
- SoCLC with priority inheritance
  - Synthetic robot application
- PARLAK synthesis results

Microbenchmark (1)

```
#define N 500  // iterations

for (i = 0; i < N ; i++)
{
      Lock(lock_variable);
        // Begin critical section
        Access_shared_data_here();
        // End critical section
      UnLock(lock_variable);
}
```

- PowerPC platform
- Microbenchmark program
- 24B data
- Compared SoCLC against spin-lock and MCS
- SoCLC speedup:
  - Over spin-lock: 37%
  - Over MCS: 19%

Microbenchmark (1 cont.)

Microbenchmark Performance Result

Total Elapsed Time (u sec)			SoCLC Speedup over Spin-lock	SoCLC Speedup over MCS
Without SoCLC		With SoCLC		
Spin-lock	MCS			
5521.5	4820.8	4044.5	37%	19%

Microbenchmark (2)

```
#define N 500  // iterations
for (i = 0; i < N ; i++)
{
      Lock(lock_variable1);
        // Begin critical section
        Access_shared_data_here();
        // End critical section
      UnLock(lock_variable1);
}
```

lock variable1	lock variable2	lock variable3	lock variable4

Performance Result

	Spin-lock	MCS	SoCLC	SoCLC Speedup
Elapsed time	2375.7 usec	2768.7 usec	1868.6 usec	27% over spin-lock 48% over MCS

- PowerPC platform
- Observe false sharing effect
- 24B data
- Used separate lock variables (residing in the same cache line) for each task
- No lock contention at all, but false sharing!
- SoCLC speedup:
  - Over spin-lock: 27%
  - Over MCS: 48%

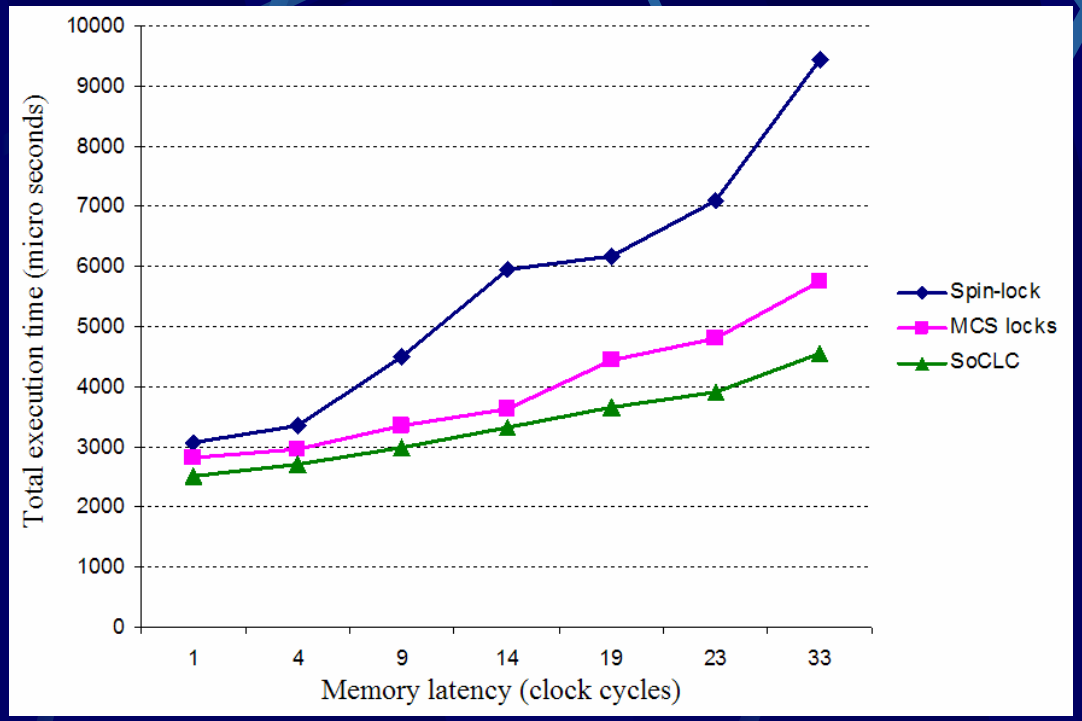
Microbenchmark (3)

- **PowerPC platform**
- **Observe CS length effect**
- **Increased CS length in the same bench**

Locking Scheme Used	Total Elapsed Time (u sec)		
	CS1 24B data, N=500	CS2 48B data, N=500	CS3 64B data, N=500
Spin-lock	5521.5	7685.4	9728.1
MCS	4820.8	7026.3	9257.0
SoCLC	4044.5	6227.4	8545.5

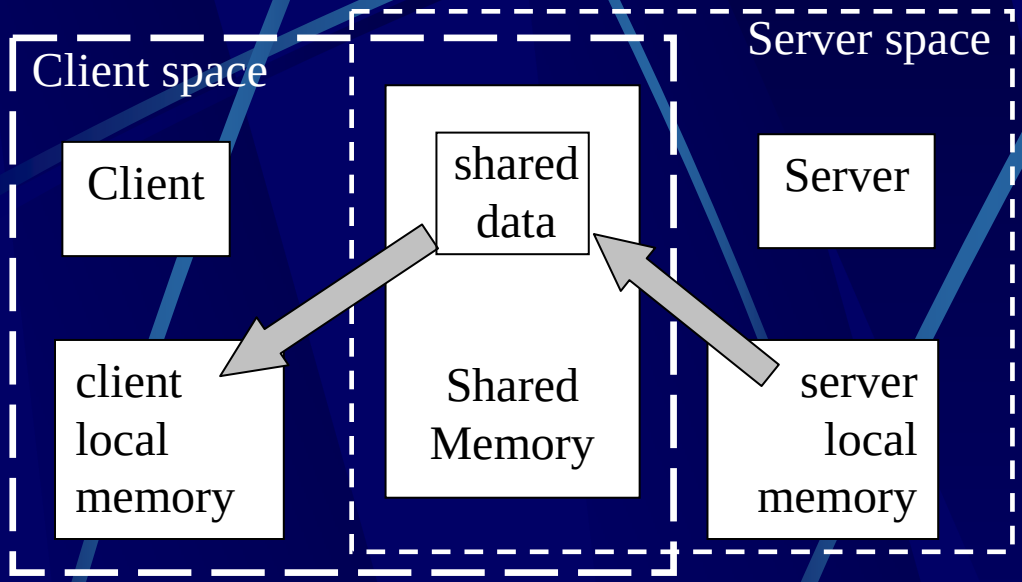
SoCLC speedup				
	Spin-lock	37%	23%	14%
	MCS	19%	13%	8%

Microbenchmark (4)

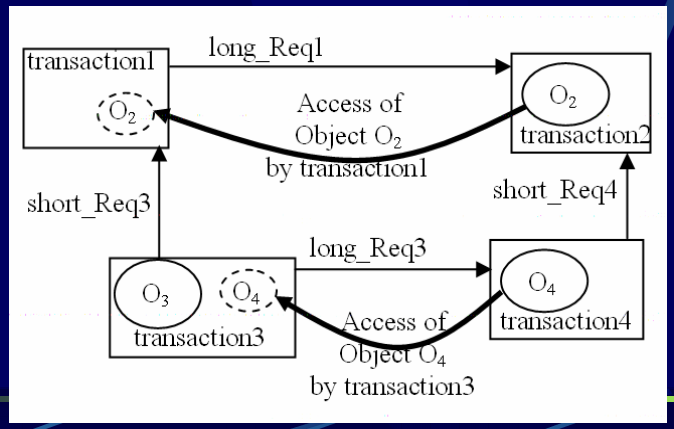


- PowerPC platform
- Observe memory latency effect
- 48B data, N=200
- Increased memory latency
  → higher cache miss penalty
  → worse performance

Database Application



- Database example application
- 40 tasks (client/server pair of tasks)
- 31% speedup



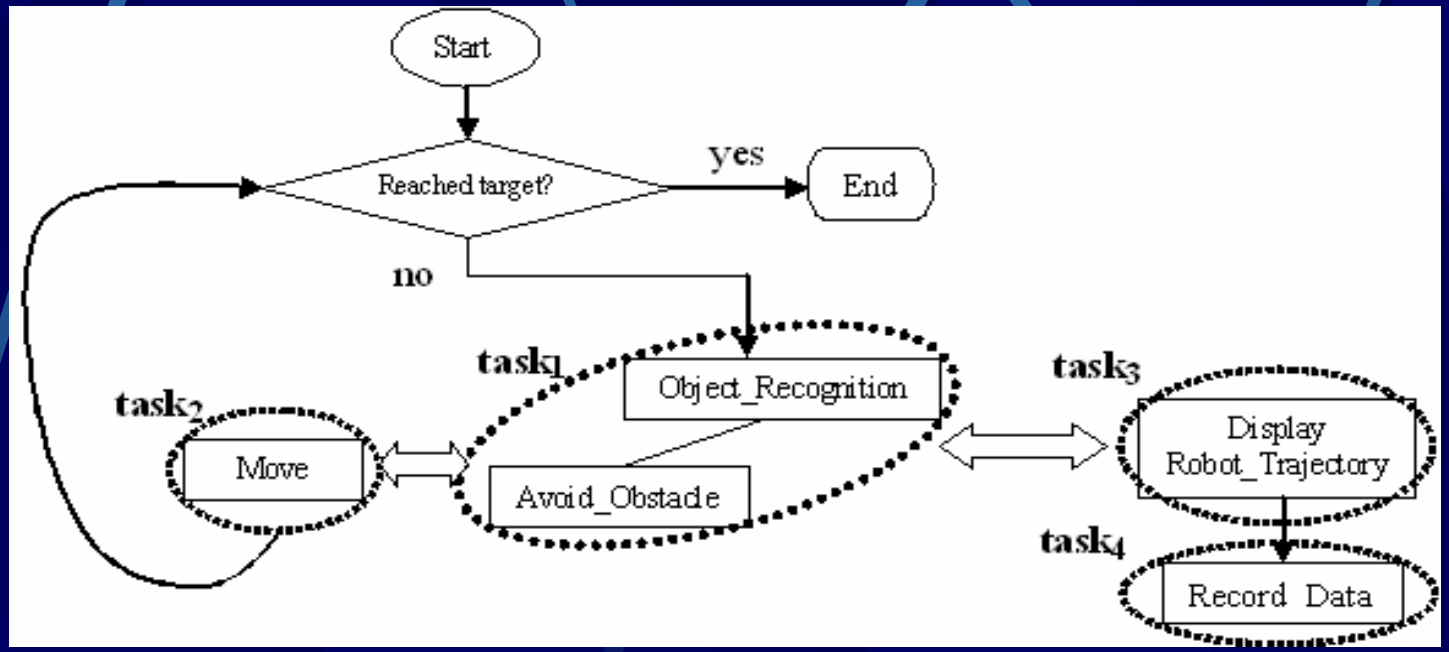
Performance Result

	Without SoCLC	With SoCLC	Speedup
Elapsed time (usec)	4955	3786	31%

Experimental Results

- Basic SoCLC
  - Microbenchmark simulations
  - Database application
- SoCLC with priority inheritance
  - Synthetic robot application
- PARLAK synthesis results

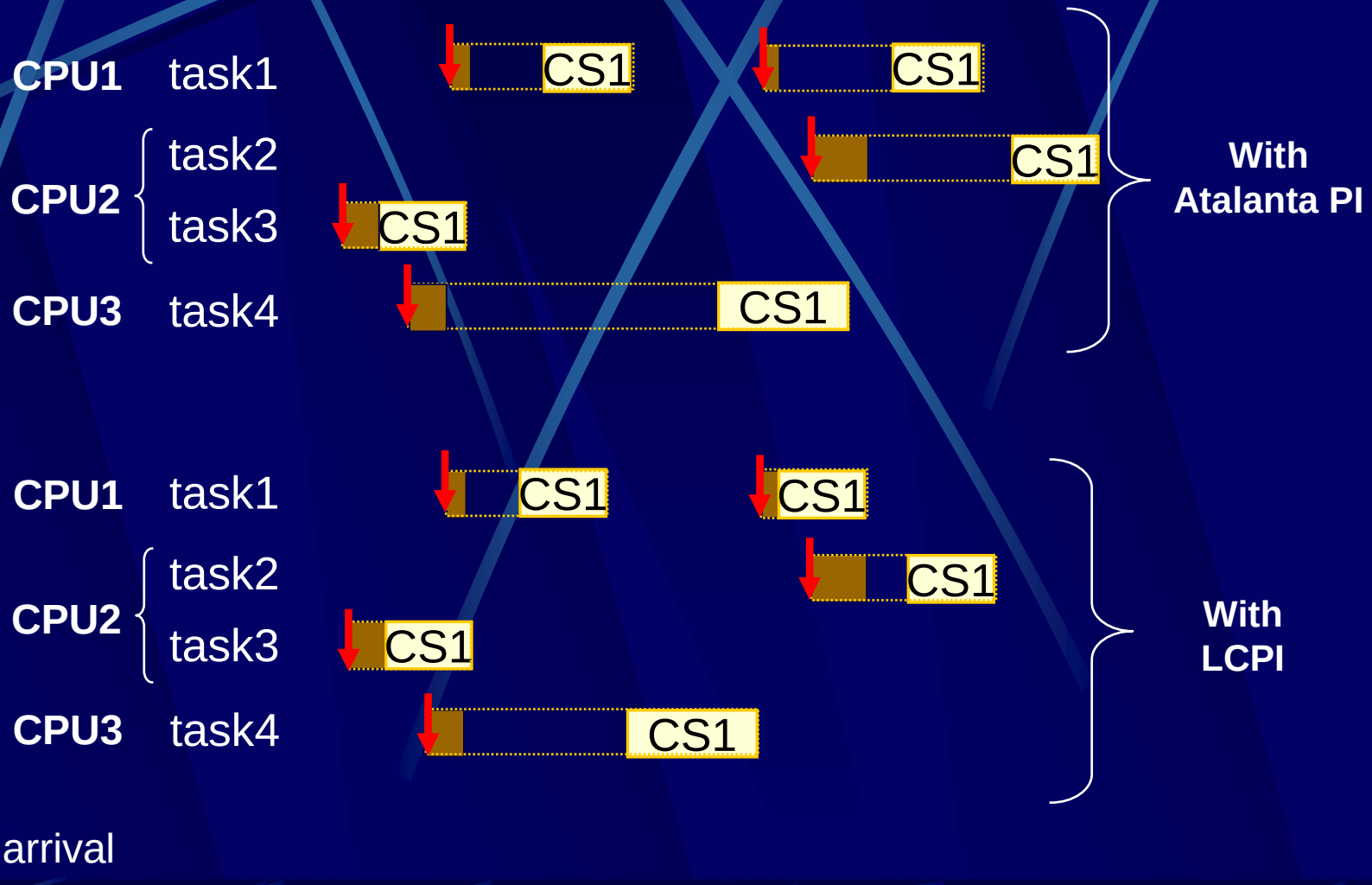
Simulation Scenario: a robot application



Task Priorities

- Task1 → highest priority task with critical hard real-time requirement (response time: 250 us)
- Task2 → second highest priority task (response time: 300 us)
- Task3 → third highest priority task (response time: 300 us)
- Task4 → lowest priority task (response time: 600 us)

Execution Trace



Measurement Results

	Task 1	Task 2	Task 3	Task 4
WCRT	250 us	300 us	300 us	600 us
Completion time for Atalanta PI Protocol	283 us	556 us	80 us	517 us
Completion time for LCPI	93 us	247 us	77 us	337 us

	Without SoCLC	With SoCLC	Speedup
Lock Latency (clk cycles)	570	318	1.79 X
Lock Delay (clk cycles)	6701	3834	1.75 X
Overall Execution (clk cycles)	112170	78226	1.43 X

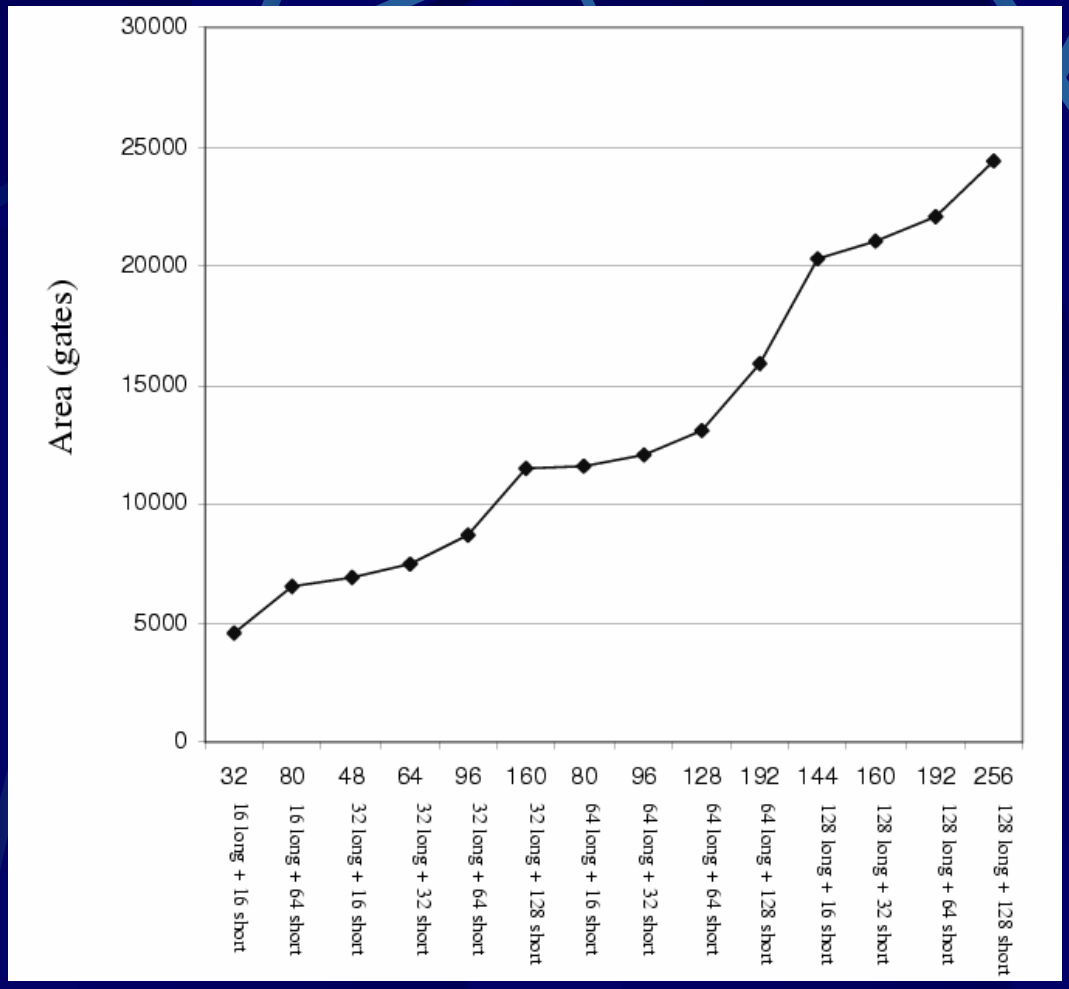
Experimental Results

- Basic SoCLC
  - Microbenchmark simulations
  - Database application
- SoCLC with priority inheritance
  - Synthetic robot application
- PARLAK synthesis results

PARLAK Synthesis Results

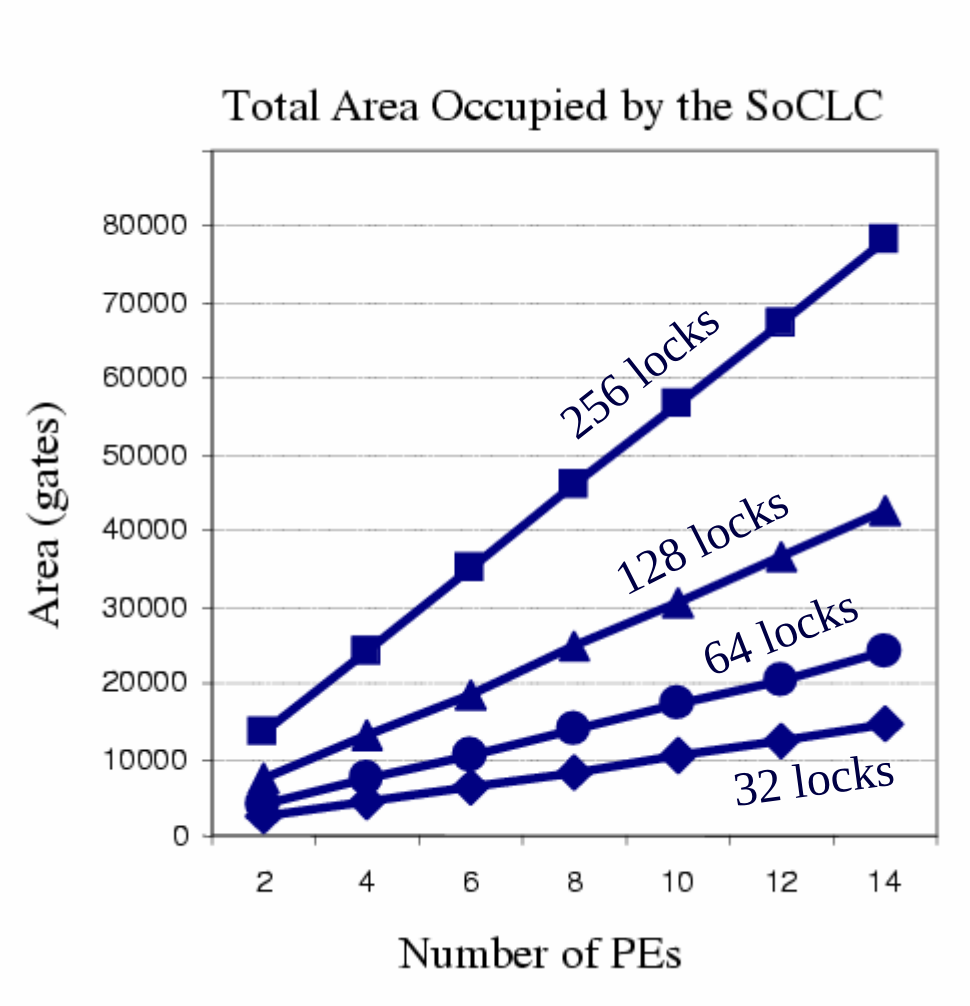
- Full range of customized SoCLCs that are generated by PARLAK have been directly synthesized using Design Compiler from Synopsys
- 0.25µ TSMC standard cell library from LEDA
- An SoCLC for two processors with 32 lock variables occupies 2,520 gates and an SoCLC for 14 processors with 256 lock variables occupies 78,240 gates
- PARLAK output SoCLC and top configurations are also simulated to test correctness in the Seamless CVE platform from Mentor Graphics

Synthesis Results



- TSMC 0.25u, LEDA
- 1 gate area = 2-input standard NAND gate area
- Use registers for lock bits
- 4 processors, 32 to 256 locks: 4,600 to 24,370 gates

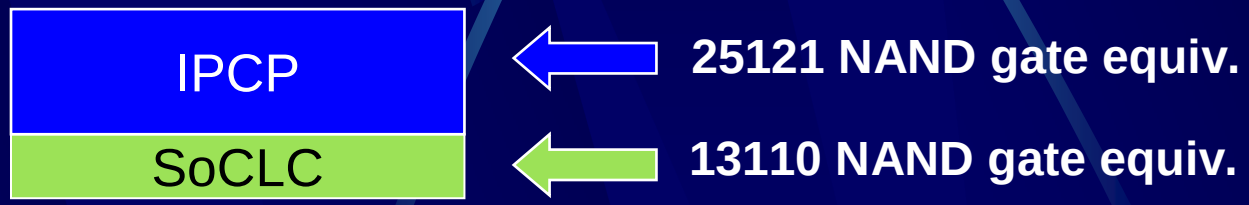
Synthesis Results



- Different SoCLC configurations for increasing number of processors
- Number of locks: 32, 64, 128, 256

Synthesis Results

Number of processors	short CS locks	long CS locks	total number of locks	SoCLC with IPCP total area	SoCLC without IPCP total area	IPCP hardware total area
4	16	16	32	13063	4605	8458
4	16	32	48	20859	6873	13986
4	32	32	64	21430	7435	13995
4	32	64	96	36877	12085	24792
4	64	64	128	38231	13110	25121



- Area in NAND gate equivalents in .25u TSMC
- Can easily fit into on-chip eFPGA

Area Estimation of an SoC

System Component	# of transistors
4 MPC755 processors including 32 KB D-$ and 32 KB I-$	4 x 6.75 M = 27 M transistors
16 MB global memory	134.217 M transistors
SoCLC logic for 128 locks + IPCP logic + memory logic	40 K gates = 160 K transistors
Total SoC area	**161.377 M**
SoCLC / SoC (%)	**160K / 161.377M = 0.1 %**

Conclusion

- SoCLC: Custom hardware logic that improves lock-based synchronization performance in a multiprocessor SoC
- Effective and low cost
- Priority inheritance support
- Paradigm shift: decision making between hw/sw
- Hardware/Software architecture
  - MPC755 and ARM processors; RTOS
- PARLAK: IP generator tool

Publications as First Author

- **B. E. S. Akgul,** V. Mooney, H. Thane and P. Kuacharoen, “Hardware Support for Priority Inheritance,” *Proceedings of the IEEE Real-Time Systems Symposium* (RTSS’03), pp. 246-254, December 2003.
- **B. E. S. Akgul** and V. Mooney, “PARLAK: Parametrized Lock Cache Generator,” *Design Automation and Test in Europe* (DATE'03), pp. 1138-1139, March 2003.
- **B. E. S. Akgul** and V. Mooney, “The System-on-a-Chip Lock Cache,” *International Journal of Design Automation for Embedded Systems*, 7(1-2), pp. 139-174, September 2002.
- **B. E. S. Akgul**, J. Lee and V. Mooney, “A System-on-a-Chip Lock Cache with Task Preemption Support,” *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (CASES'01), pp. 149-157, November 2001.
- **B. E. Saglam (Akgul)** and V. Mooney, “System-on-a-Chip Processor Synchronization Support in Hardware,” *Design Automation and Test in Europe* (DATE'01), pp. 633-639, March 2001.

Thank you!