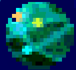
Georgia Institute of Technology

System-on-a-Chip Processor Synchronization Support in Hardware



by

Bilge E. Saglam and Vincent J. Mooney

Georgia Institute of Technology

School of Electrical and Computer Engineering

Outline

- Background
- Motivation
- Methodology
  - SoCSU Lock Cache Hardware Mechanism
  - Software Support for SoCSU
- Experimental Set-up
  - Hardware and Software Architectures
  - Database Example Simulation
- Results
- Conclusion

Background

- Critical Section
  - Code section where shared data between multiple execution units is accessed
  - E.g., multiple readers and multiple writers
  - A lock is necessary to guarantee the consistency of shared data (e.g., global variables)
- Lock Delay
  - Time between release and acquisition of a lock
- Lock Latency
  - Time to acquire a lock in the absence of contention

Background (Continued)

- Atomic Locking
  - Special load/store instructions
    - 'LL' – load linked and 'SC' – store conditional (MIPS)
    - 'lwarx' and 'stwcx.' (MPC750)
    - Paired instructions
    - Breakable link for Effective Address (EA)
  - Synchronization Primitives
    - Test-and-Set, Compare-and-Swap, Fetch-and-Increment primitives
- Ensuring mutual exclusiveness and consistency

Background (Continued)

- **test-and-set primitive**

```
TRY:  LL    r2, (r1)          ; load lock variable
      ORI   r3, r2, 1         ; set r3 = 1
      BEQ   r3, r2, TRY       ; unlocked?
      SC    r3, (r1)          ; try locking
      BEQ   r3, 0, TRY        ; succeed?

       …./* critical section here */….

      ANDI r2, r2, 0          ; set r2 = 0
      SW r2, (r1)             ; unlock lock variable
```

- **Problem: busy-wait !**

Motivation

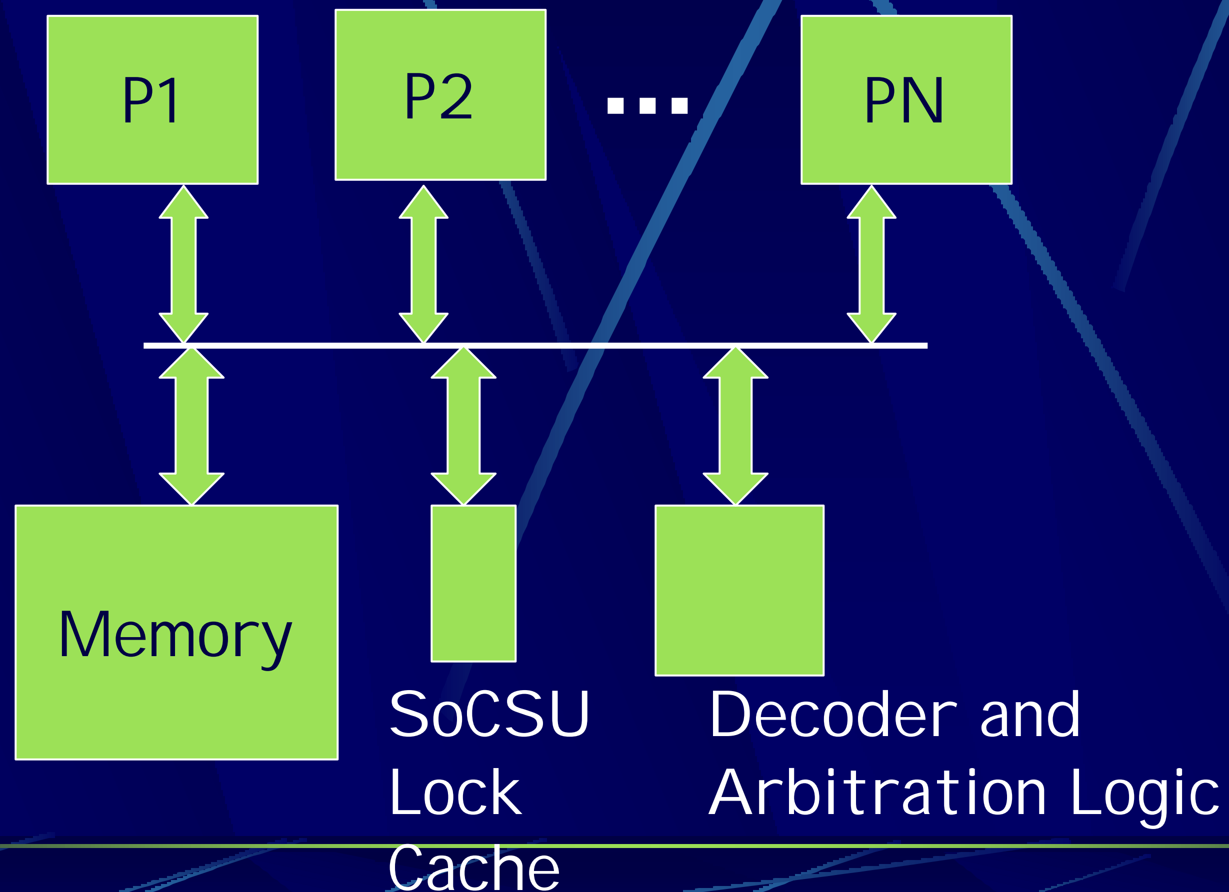
- **Previous Software Solutions**
  - spin-on-test-and-set,spin-on-read, static/adaptive delay in loops,queue algorithms (Anderson'90, etc.)
  - poor in terms of bandwidth consumption, lock delay, lock latency
  - cache invalidations → hold cycles
- **Previous Hardware Solutions**
  - special cache schemes, each processor has a private cache directory for locks (Ramachandran'96, etc.)
  - dependent on memory hierarchy (special consistency model)

Motivation (Continued)

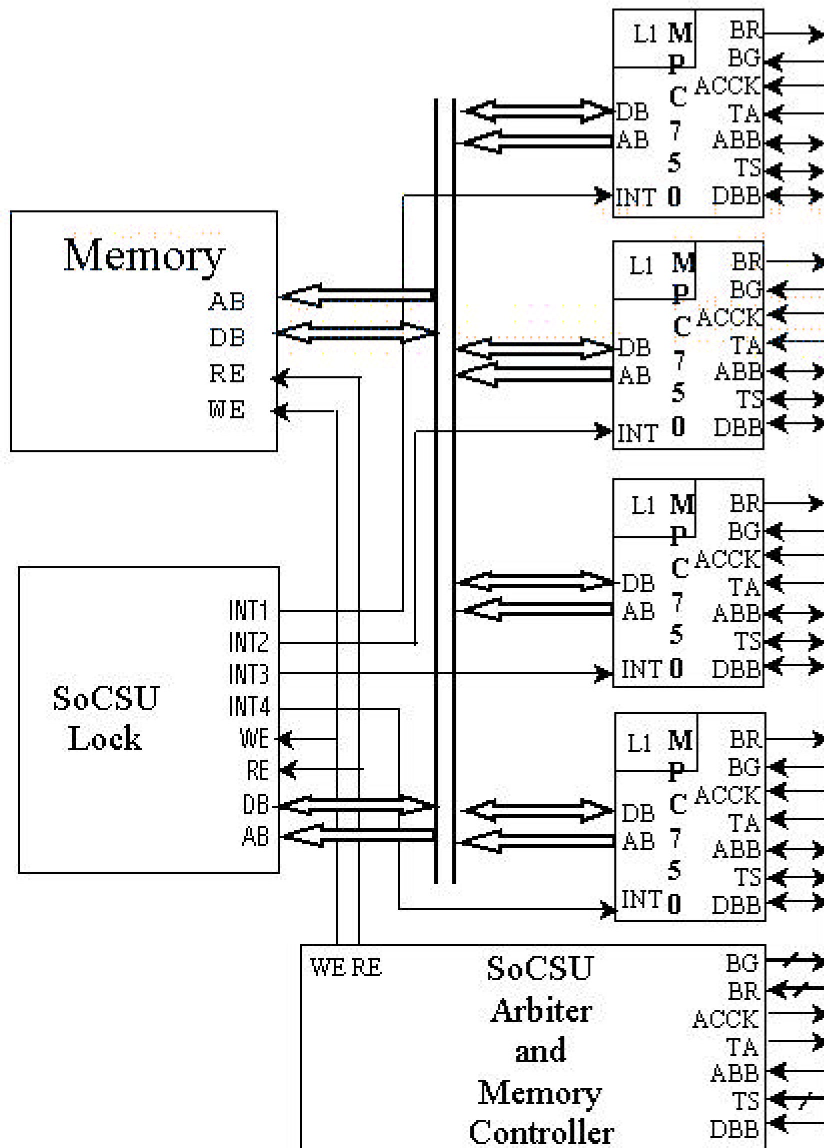
- Solution in hardware: SoCSU Lock Cache
- Deterministic and much faster access to lock variables
- Better performance in terms of lock delay, lock latency and bandwidth consumption
- Higher scalability for multi-processor SoC designs
- RTOS support

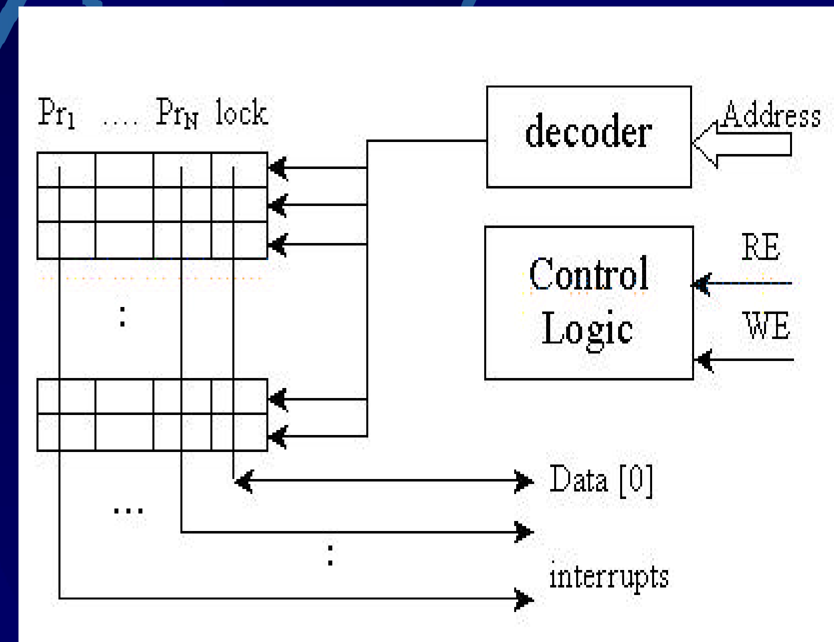
Methodology

SoCSU Lock Cache Hardware Mechanism



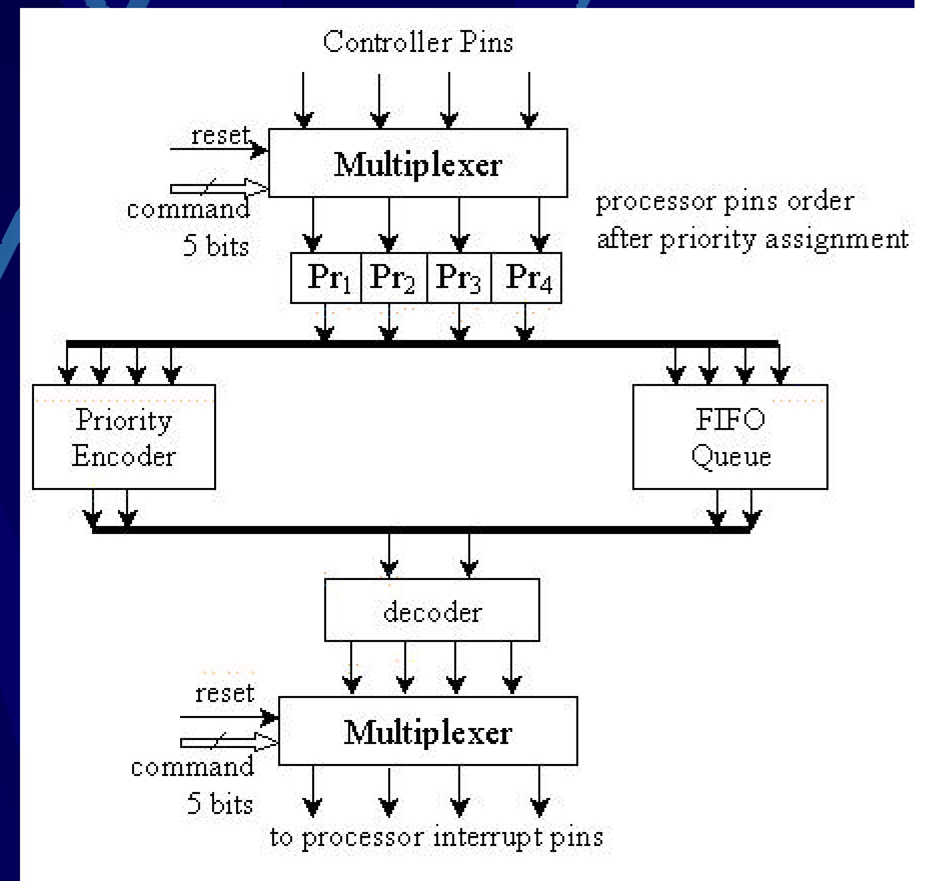
Methodology (Continued)
SoCSU Lock Cache Hardware





Methodology (Continued) SoCSU Lock Cache Hardware Mechanism

- **Interrupt Generation**
  - Programmable priority assignment during system reset
  - Notify one processor at a time → preventing unnecessary signaling
  - Priority or FIFO



Methodology
Software Example for SoCSU

- **Traditional code for spin-lock**

C:
```
Lock (  lock variable );
…/*critical section*/…
UnLock ( lock variable );
```

ASM:
```
try:  LL   R2,(R1)       ;read the lock
      ORI  R3,R2,1
      BEQ  R3,R2,try     ;spin if lock is busy
      SC   R3,(R1)       ;acquire the lock
      BEQ  R3,0,try      ;spin if store fails
      …/*critical section*/…
      SW   R2, (R1)      ;release lock
```

Methodology (Continued)
Software Example for SoCSU

- **New code with SoCSU Lock Cache HW support**

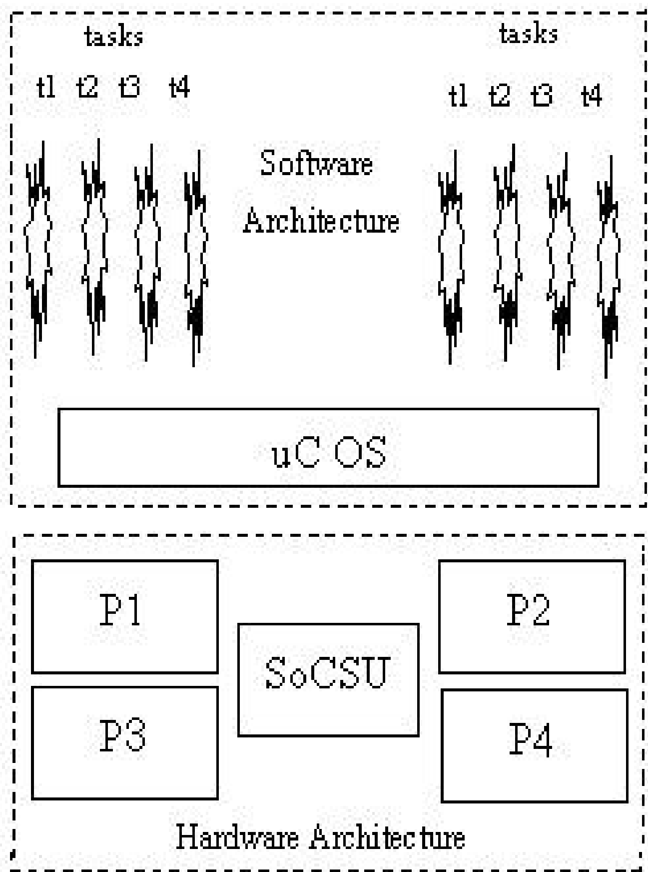
C:

```
Lock ( lock variable );
…/*critical section*/…
UnLock ( lock variable );
```

ASM:

```
try:  LW   R2,(R1)       ;read the lock
      BEQ  R2,1,sleep    ;succeed?
      …/*critical section*/…
      SW   R2, (R1)      ;release lock
```

SoCSU Lock Cache vs. Traditional Implementation



	SoCSU	Traditional
P1	P2	P2
Task1	Task2	Task2
Lock(); Succeed	Lock(); Fail	Lock(); Fail
C.S.	Sleep();	
Unlock(); →	Interrupt	Contend
	Lock();	Lock();
	Succeed	Succeed?
	C.S.	C.S.
	Unlock();	Unlock();

Methodology (Continued)
SoCSU Software Implementation

- **Special load (LL) and store (SC) instructions removed**
- **Latency reduced**
- **Assumption: only small critical sections**
  - **sleep instead of context switch**
- **ISR enables the sleeping task to return back to its original program flow**

```
ISR:    mflr   %r0
        mtspr  %SRR0, %r0
        rfi
```

- **No need to save context; high responsiveness**

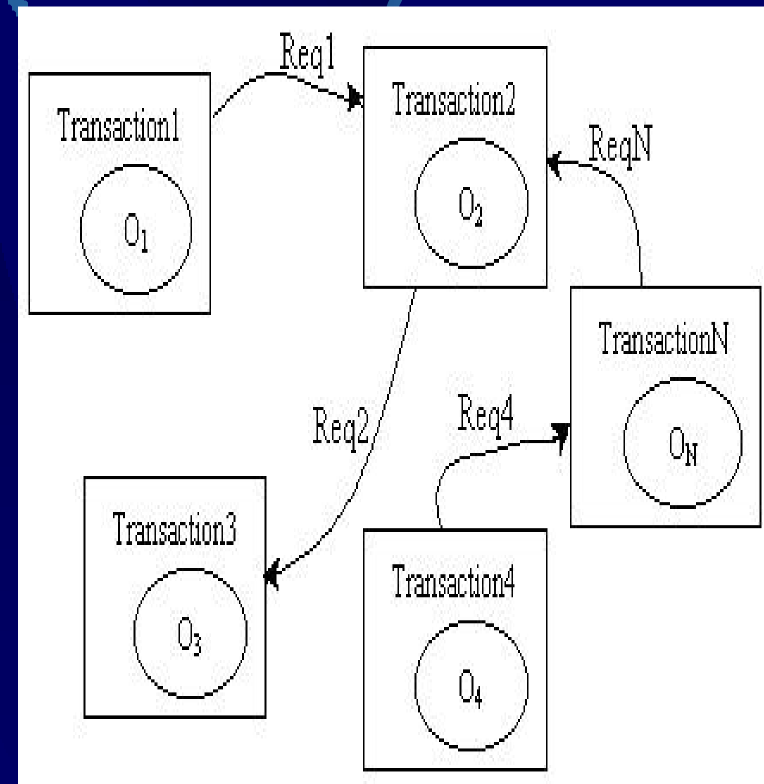
Experimental Set-up

- Seamless Co-Verification Environment (Seamless CVE)
- Seamless processor support packages for PPC family (we are using MPC750)
- Instruction set simulators
- Synopsys VCS verilog simulator
- RTOS – using uC/OS-II

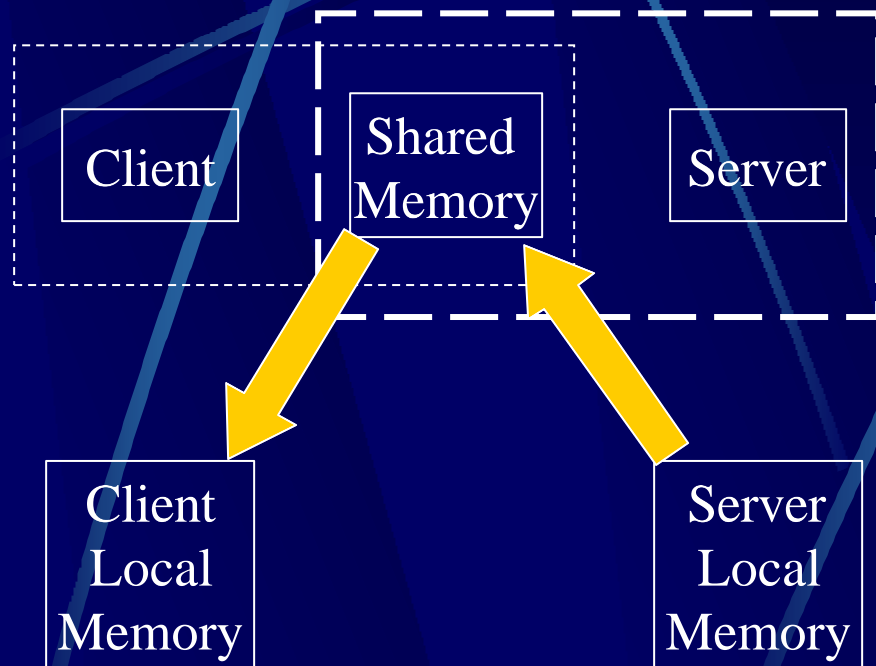
Experimental Set-up (Continued)

Database Example Simulation

- **Four MPC750 processors**
- **Database example application combined with client/server pair execution model**
- **Thread-level synchronization**
  - each thread acquires a lock
  - a transaction = accessing a database (critical section)
  - SoCSU provides synchronization



Database Example Simulation (Continued)



- Server accesses to shared memory object after acquiring lock from SoCSU Lock Cache
- Server reads from its own local memory into the shared memory object
- Server notifies client by releasing the lock (interrupt sent from SoCSU Lock Cache)
- Client acquires the lock and copies the data from shared memory into its own local memory

Results

- Simulation with 10 server tasks on one processor and 30 client tasks on the other 3 processors
- Worst case experimental results for 4-processor simulation (comparing SoCSU approach with the traditional spin-lock method):

	Spin-Lock	SoCSU Lock Cache
Lock Latency (#clk cycles)	17	3.5
Lock Delay (#clk cycles)	15578	34.5
Total Execution time (#clk cycles)	1326311	1040714

- Total execution time → 27% speedup
- Lock delay → 451 times, Lock latency → 4.8 times

Conclusion & Future Work

- **A hardware mechanism for multi-processor SoC Synchronization: SoCSU Lock Cache**
- **Reduction in lock latency, lock delay**
- **Constant traffic contention complexity**
- **27% overall speedup in an example database application**

- **Note: patent pending**

- **Future Work**
  - Support *both* long Critical Sections and short Critical Sections
  - Allow context-switching of tasks instead of sleeping
  - RTOS modifications
  - Hardware Modifications