Automated Generation of Round-robin Arbitration and Crossbar Switch Logic

Eung S. Shin

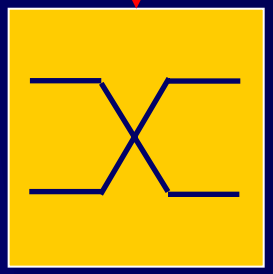
Advisor: Professor Vincent J Mooney III

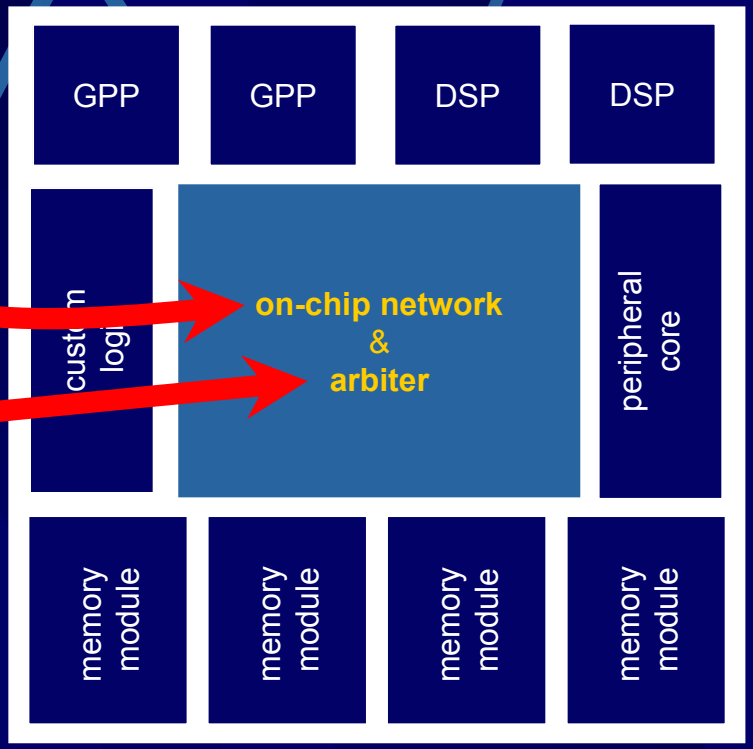
School of Electrical and Computer Engineering, Georgia Institute of Technology

Overview

Crossbar (Xbar)

round-robin arbiter

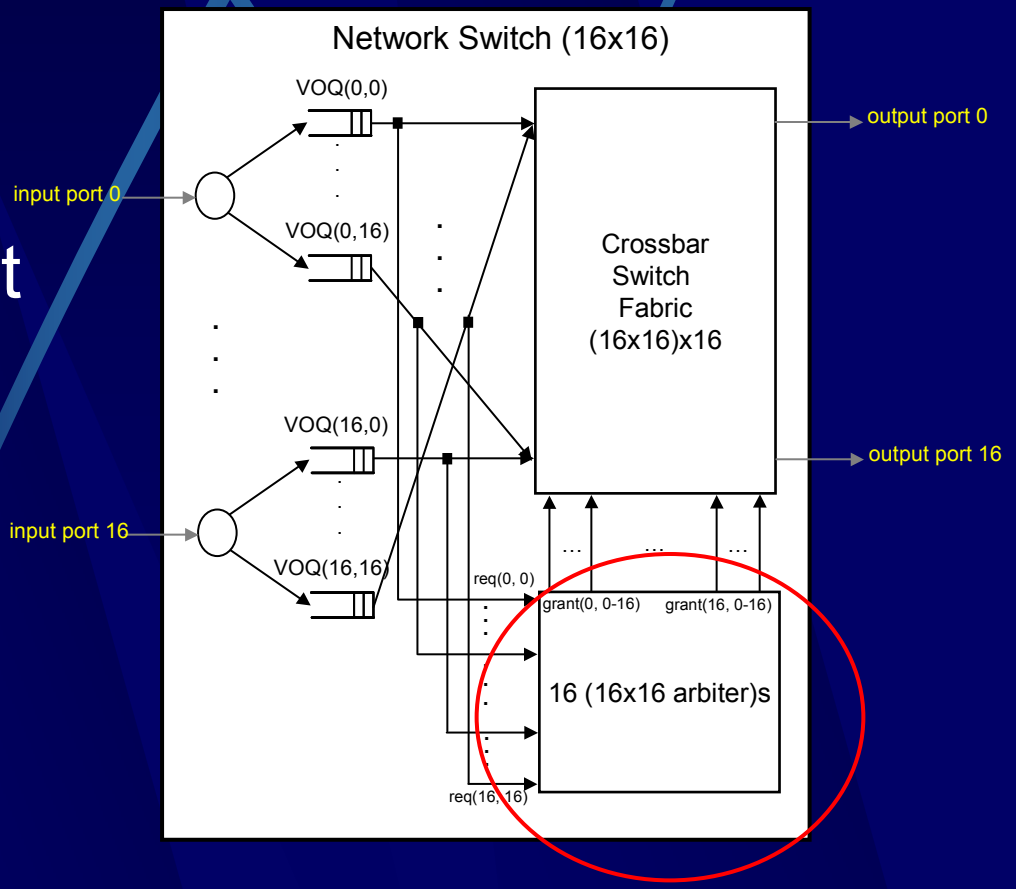




Multiprocessor System-on-a Chip (SoC)

Arbiter Problems

- A fast and powerful arbiter for an SoC
- A fast arbiter for terabit switching speeds
- A tedious and error-prone task



Xbar Problems

- Multiple communication channels demanded in a multiprocessor SoC
- Challenge: reducing productivity gap
- Productivity gap reduction techniques:
  - Enhancing IP core reusability
  - Developing a CAD tool

Objective

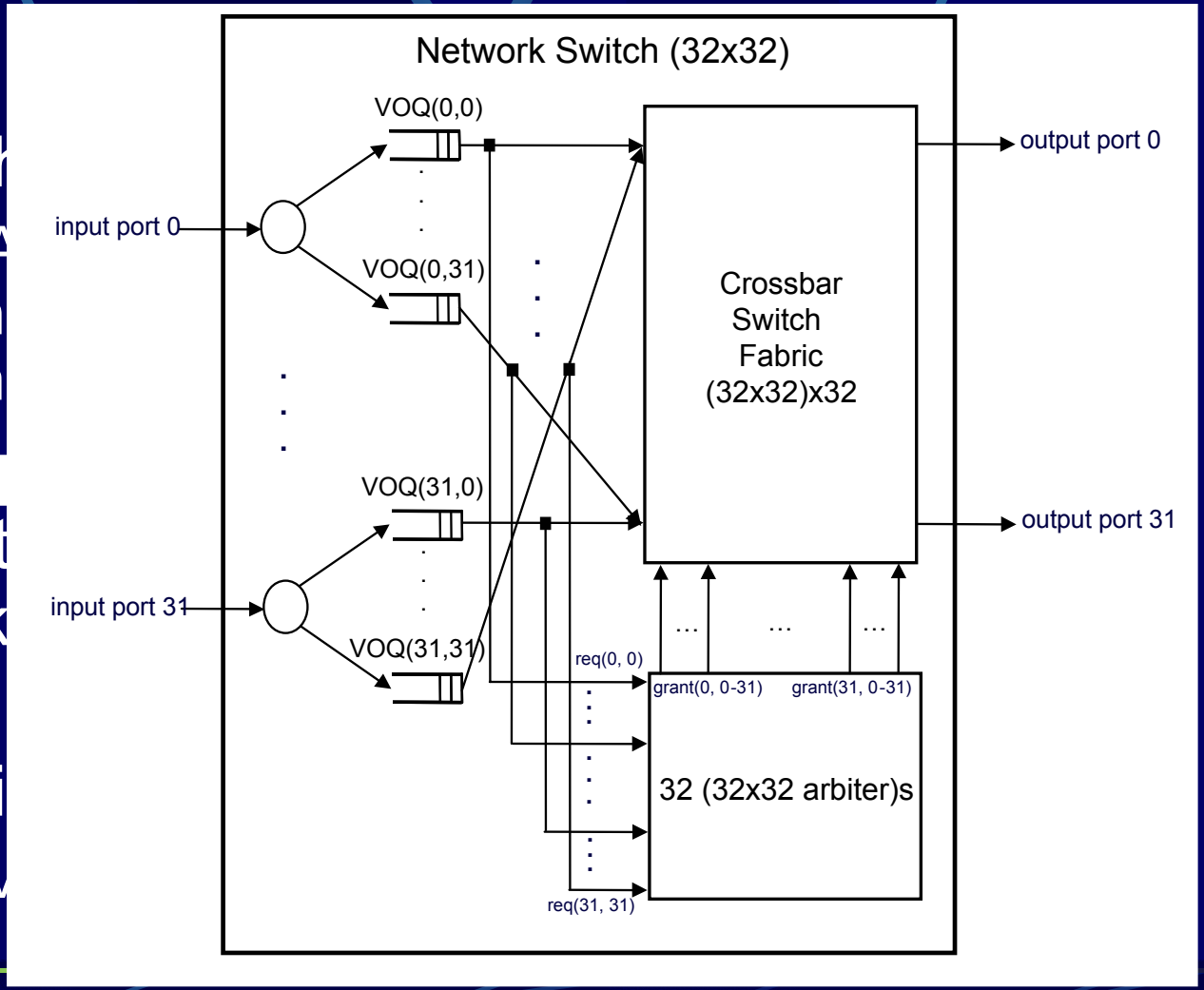
- To design and automate a fast round-robin arbiter logic generation for a bus or a network switch
  - The generated arbiter employed to crossbar (Xbar) switch arbitration logic
- To automate Xbar generation providing multiple communication paths among masters
  - The generated Xbar customized according to user specifications

Outline

- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

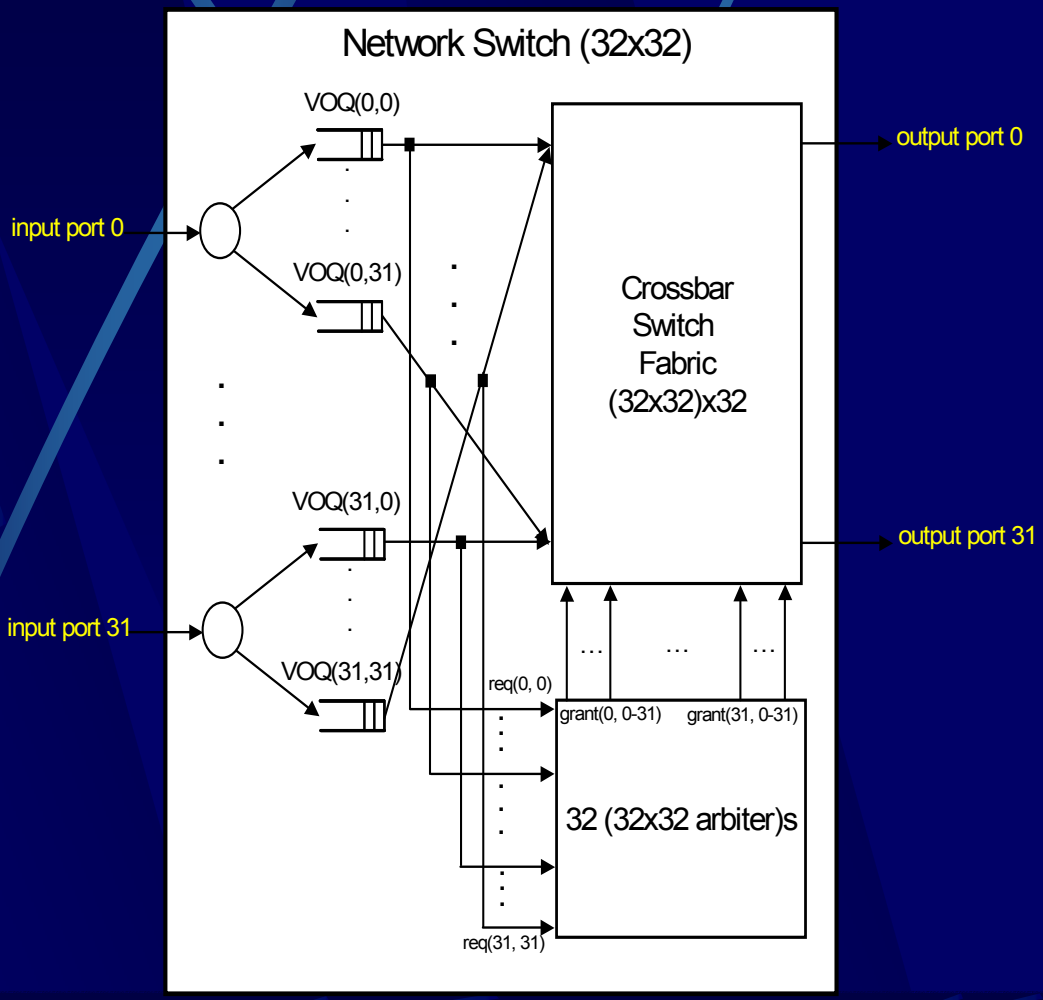
Terminology

- **MxN Switch**
  - Example: A switch with between in
- **Virtual Outp** possible out (HOL) block
- **VOQ (*m*, *n*)** output port i
  - Example: V



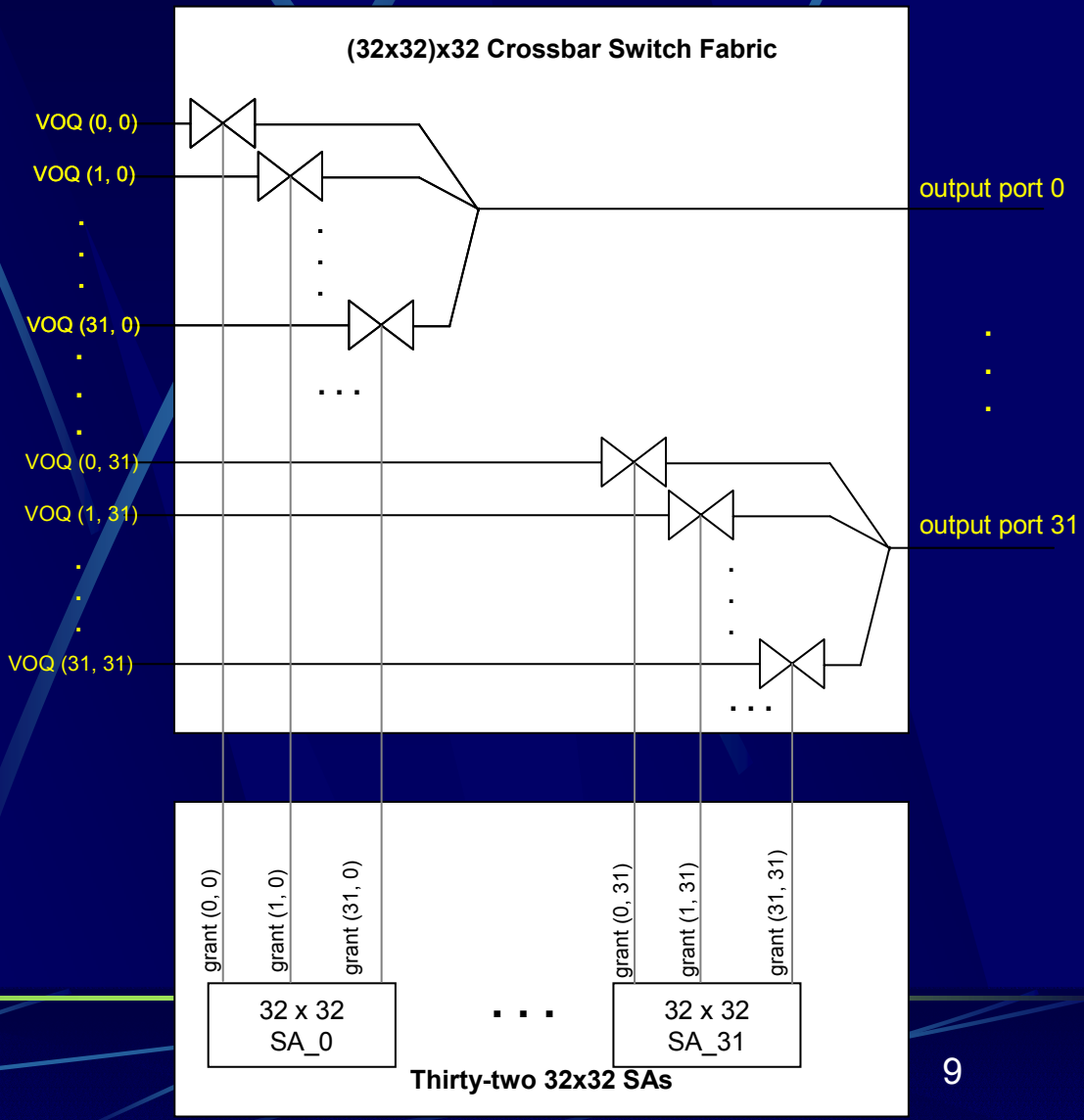
Terminology (Continued)

- **(MxV)xN Switch**:
  - M – the number of input ports of an MxN switch
  - V – the number of VOQs per input port
  - N – the number of output ports of an MxN switch
  - Typically, V = N
  - The total number of VOQs in an MxN switch – M∗N



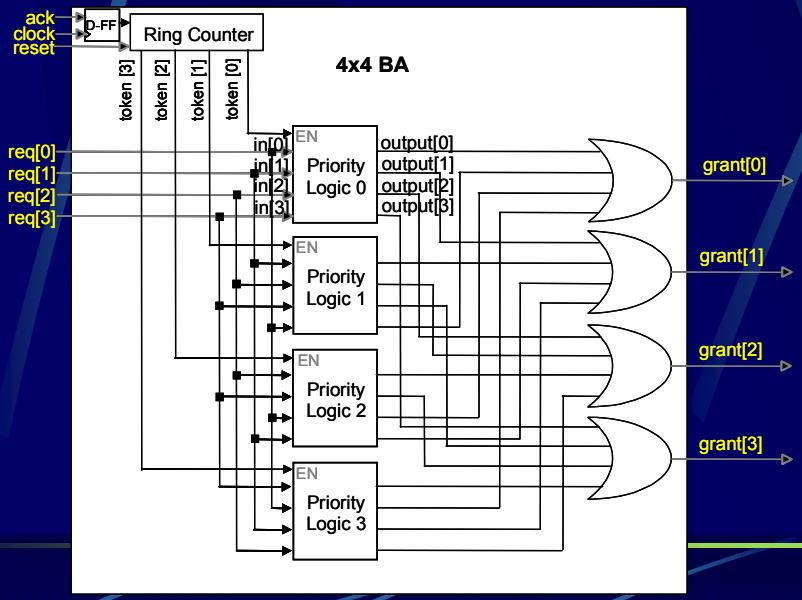
Terminology (Continued)

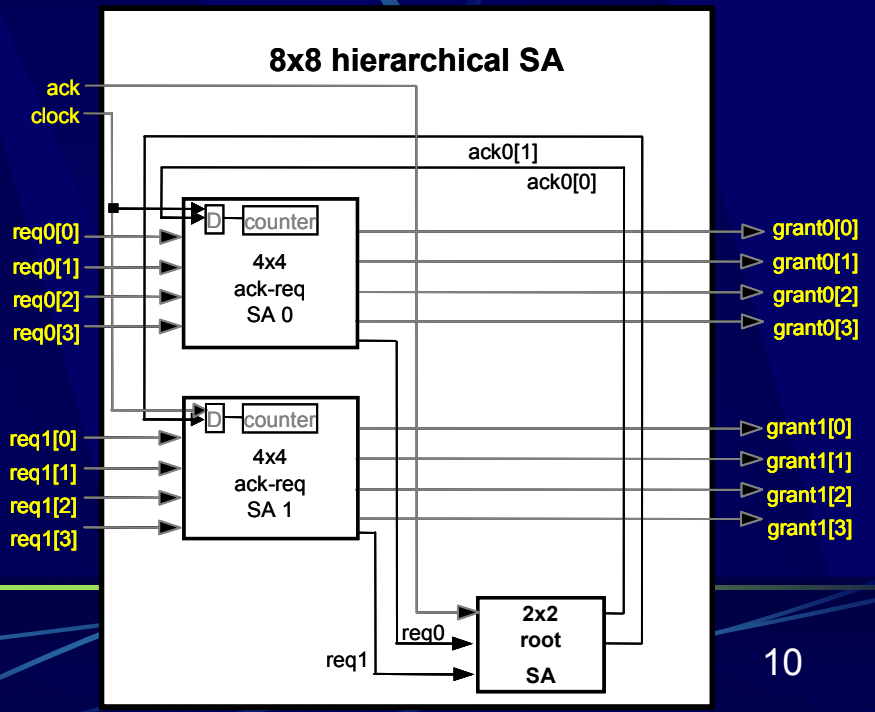
- **(MxV)xN crossbar switch fabric**:
  - Connections between (MxV) inputs and N outputs
- **MxM Switch Arbiter (SA)**:
  - Controlling M specific transmission gates between M VOQs and a particular output port
  - N MxM SAs in an MxN switch



Terminology (Continued)

- **MxM distributed SA (MxM hierarchical SA)**:
  - Equivalent to an MxM SA
  - Consisting of smaller switch arbiter in the form of a hierarchical tree structure
- **Bus Arbiter (BA)**: resolving bus conflicts





Requirements for a Terabit Switch Arbiter

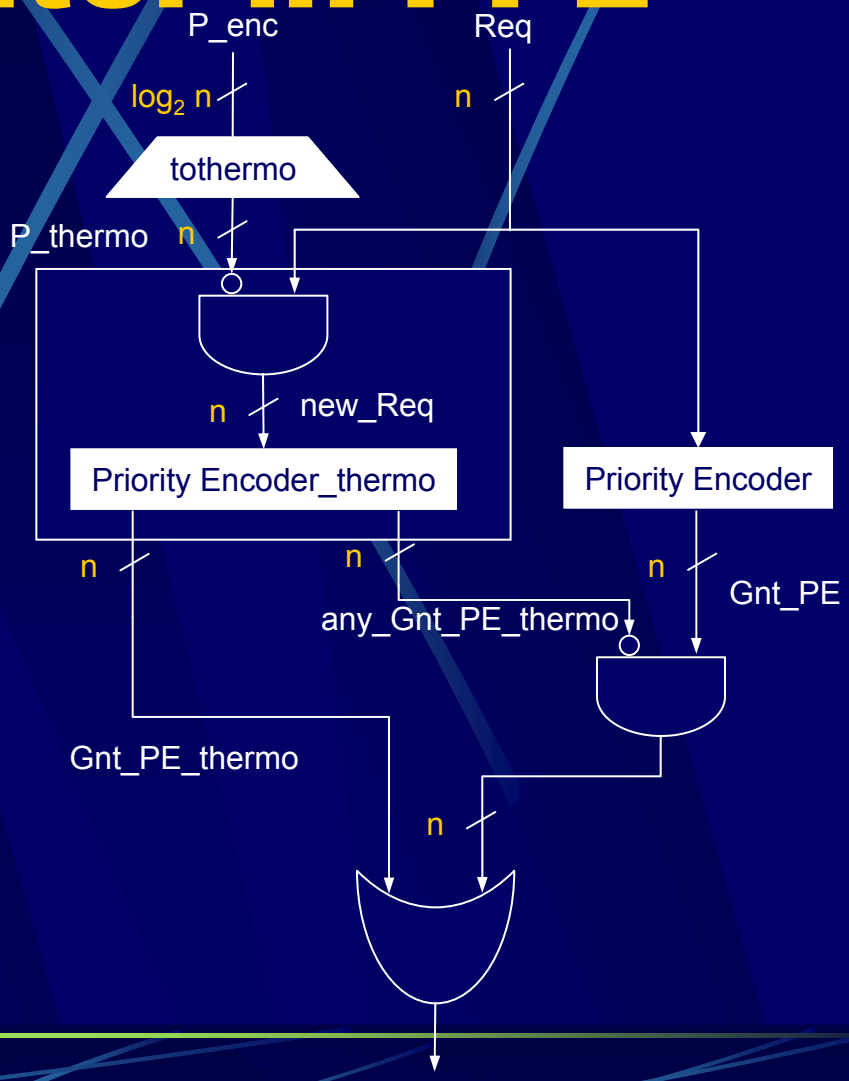
- Starvation free
- Fast Arbitration
- Simplicity to implement
- Low power:
  - Power budget of single rack router ~ 10kW

Outline

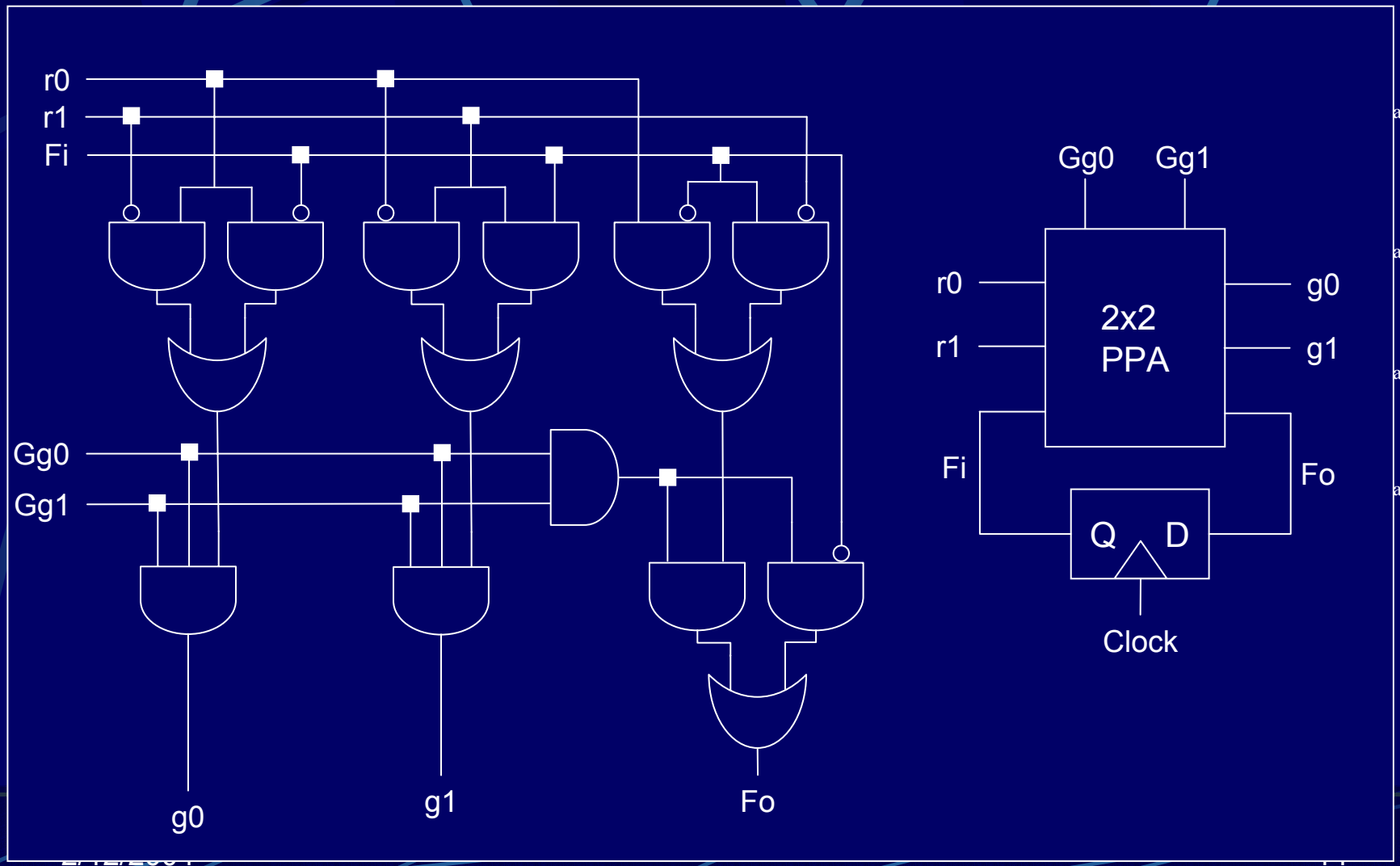
- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

History: Arbiter in PPE

- Centralized Switch Arbiters:
  - Programmable Priority Encoder (PPE) implementing iterative round-robin algorithm (iSLIP)
    - P. Gupta and N. Mckeown, "Designing and Implementing a Fast Crossbar Scheduler," *IEEE Micro*, 1999, pp. 20-28.
    - N. Mckeown, P. Varaiya, and J. Warland, "The iSLIP Scheduling Algorithm for Input-Queued Switch," *IEEE Transaction on Networks*, 1999, pp. 188-201.



History: Arbiter in PPA



ayer 4

ayer 3

ayer 2

ayer 1

Why do we need an arbiter for an SoC?

- Arbitration required by all buses
- Our arbiter applicable to anywhere requiring arbitration
- The generated arbiter utilized in our Xbar

History: Crossbar Switch

- Smart memory:
  - Reconfigurable crossbar between bus masters (2 integer clusters and 1 floating point cluster) and slaves (SRAMs)
    - K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally and M. Horowitz, “Smart Memories: A Modular Reconfigurable Architecture,” *Proceedings of International Symposium on Computer Architecture (ISCA)*, June 2000, pp. 161-171.

Outline

- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

Bus Arbiter Design

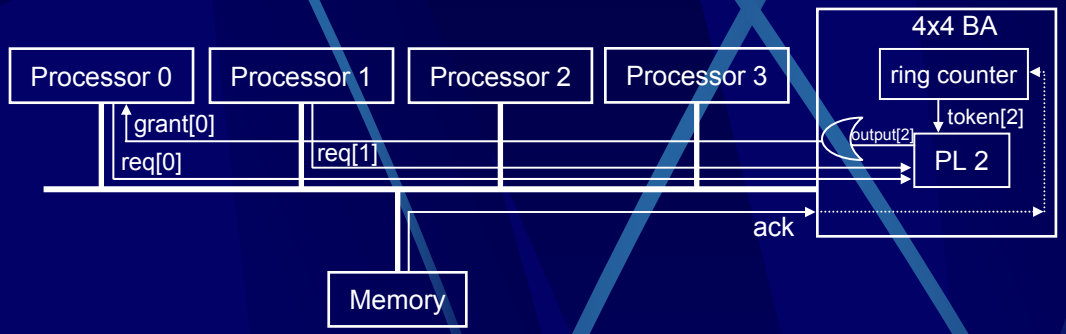
- Implemented based on ring counter for a token and “priority logic”
- Priority Logic for 4 inputs:
  - output[0] = EN•in[0]
  - output[1] = EN•in[0]'•in[1]
  - output[2] = EN•in[0]'•in[1]'•in[2]
  - output[3] = EN•in[0]'•in[1]'•in[2]'•in[3]

EN	in [0]	in [1]	in [2]	in [3]	output [0]	output [1]	output [2]	output [3]
0	X	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0	0
1	1	X	X	X	1	0	0	0
1	0	1	X	X	0	1	0	0
1	0	0	1	X	0	0	1	0
1	0	0	0	1	0	0	0	1

Previous Bus Arbiter Approach

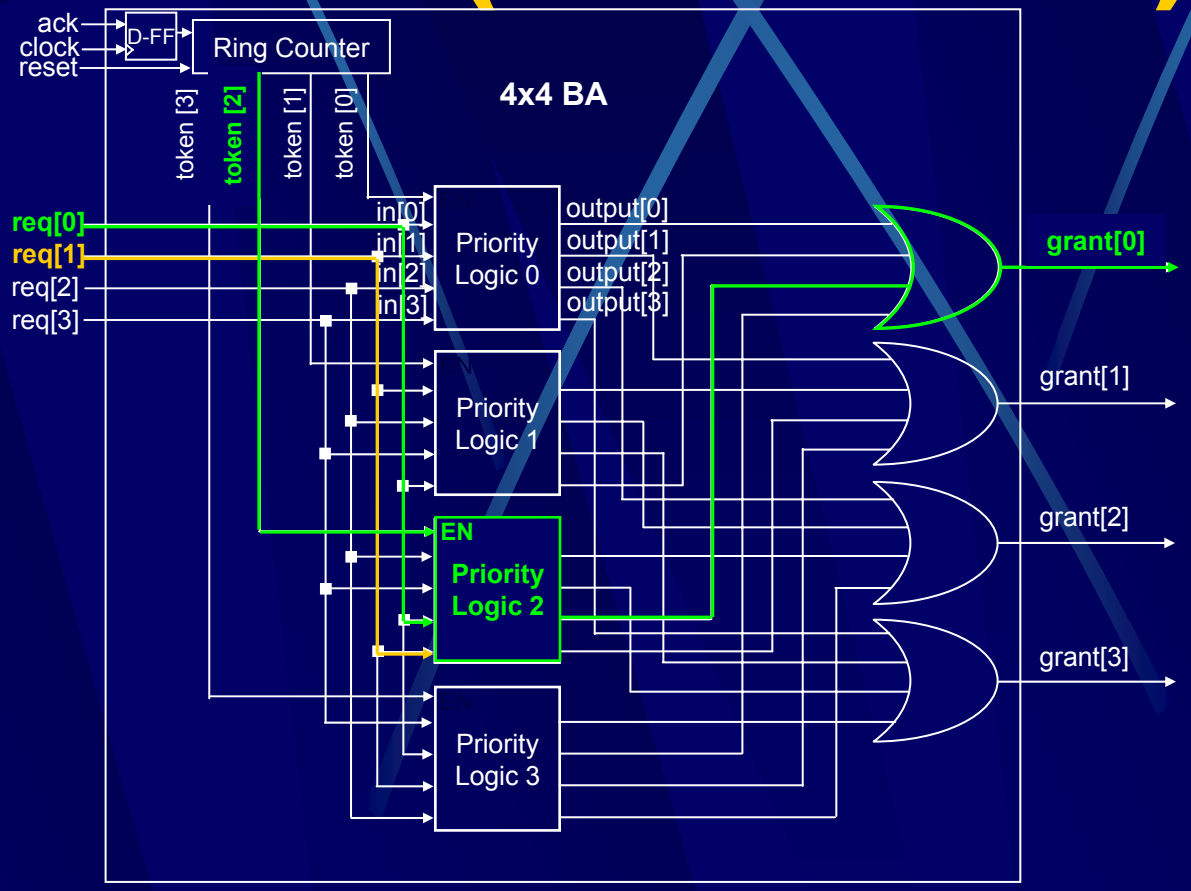
- FIFO Arbiter
- Round-robin arbiter with fixed priorities
- Round-robin arbiter with priority rotation

Example: Our Bus Arbiter



- Condition:
  - Token=4'b0100 → Processor 2 with the highest priority
  - Processors 0 and 1 requesting a bus
- Result:
  - Only Priority Logic 2 enabled
  - Processor 0 granted due to negated request signals of the higher priority parties (processor 2 and processor 3)
  - Token rotated to 4'b1000 after the ring counter receiving ack signal

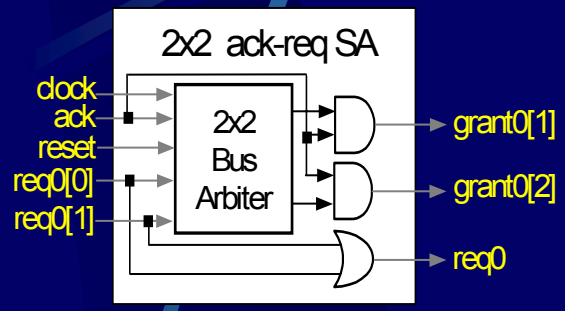
Example: Our Bus Arbiter (Continued)

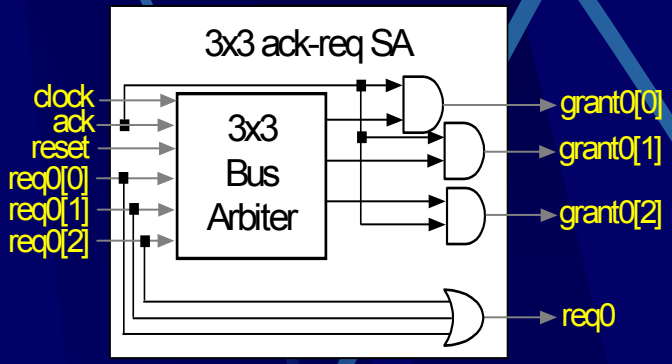


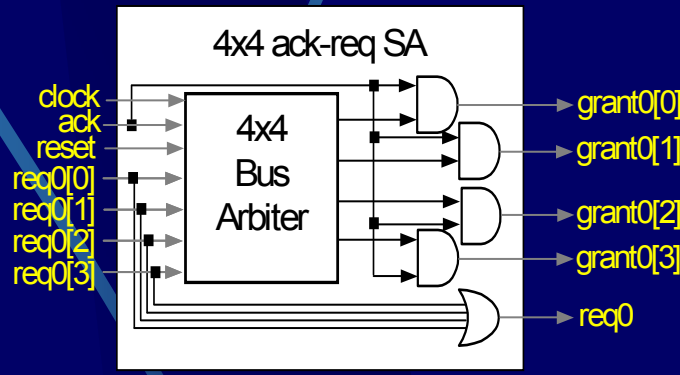
Hierarchical SA Design

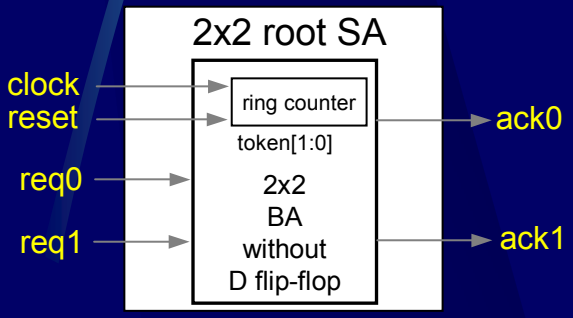
- A hierarchical SA consisting of small switch arbiter blocks
- Six types of switch arbiter blocks:
  - 2x2 ack-req SA
  - 3x3 ack-req SA
  - 4x4 ack-req SA
  - 2x2 root SA
  - 3x3 root SA
  - 4x4 root SA
- A root SA: placed on the top of a hierarchy

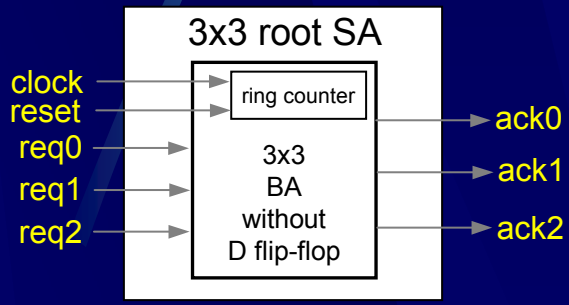
Switch Arbiter Blocks

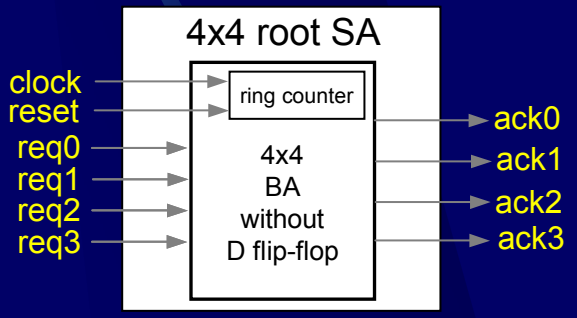




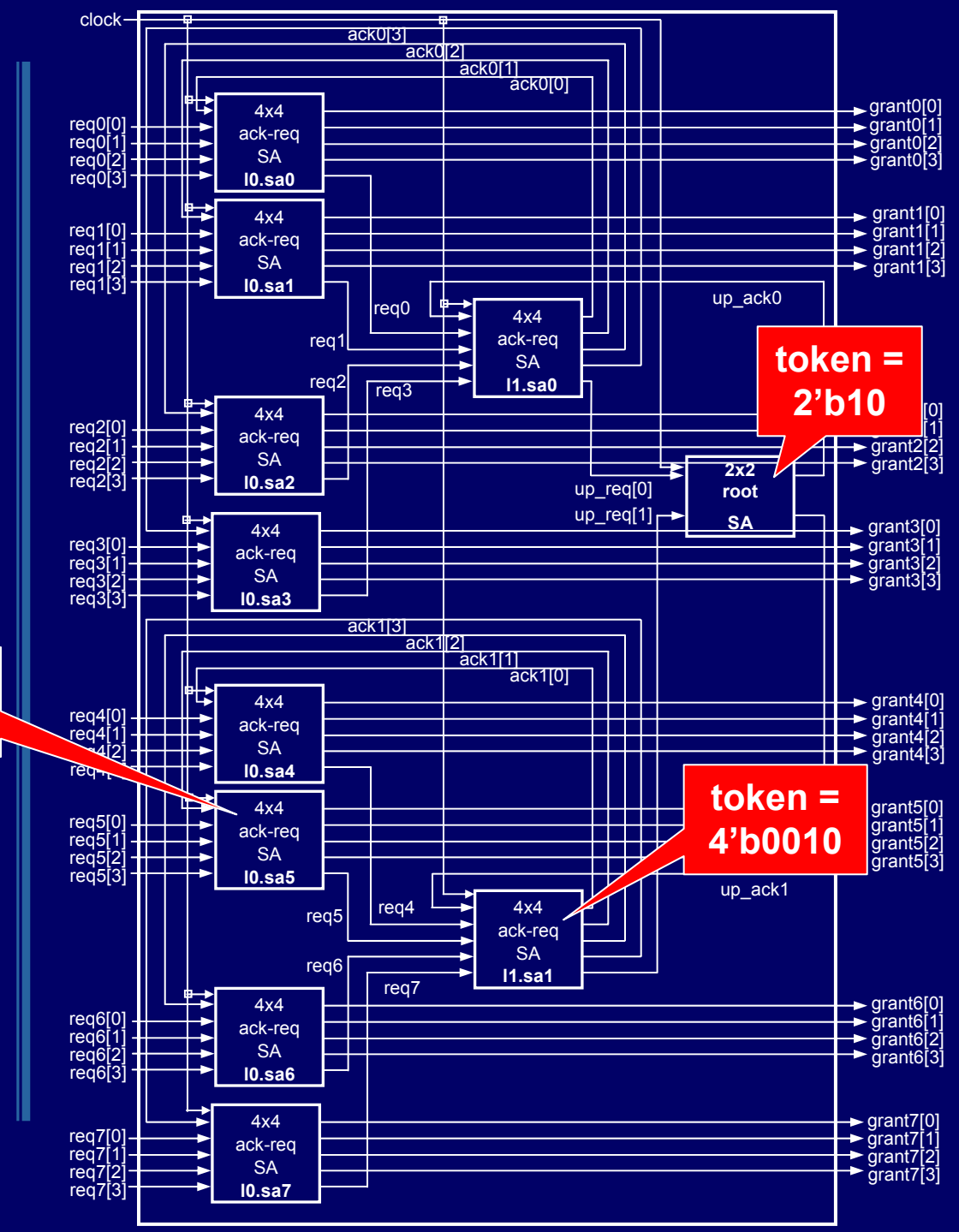








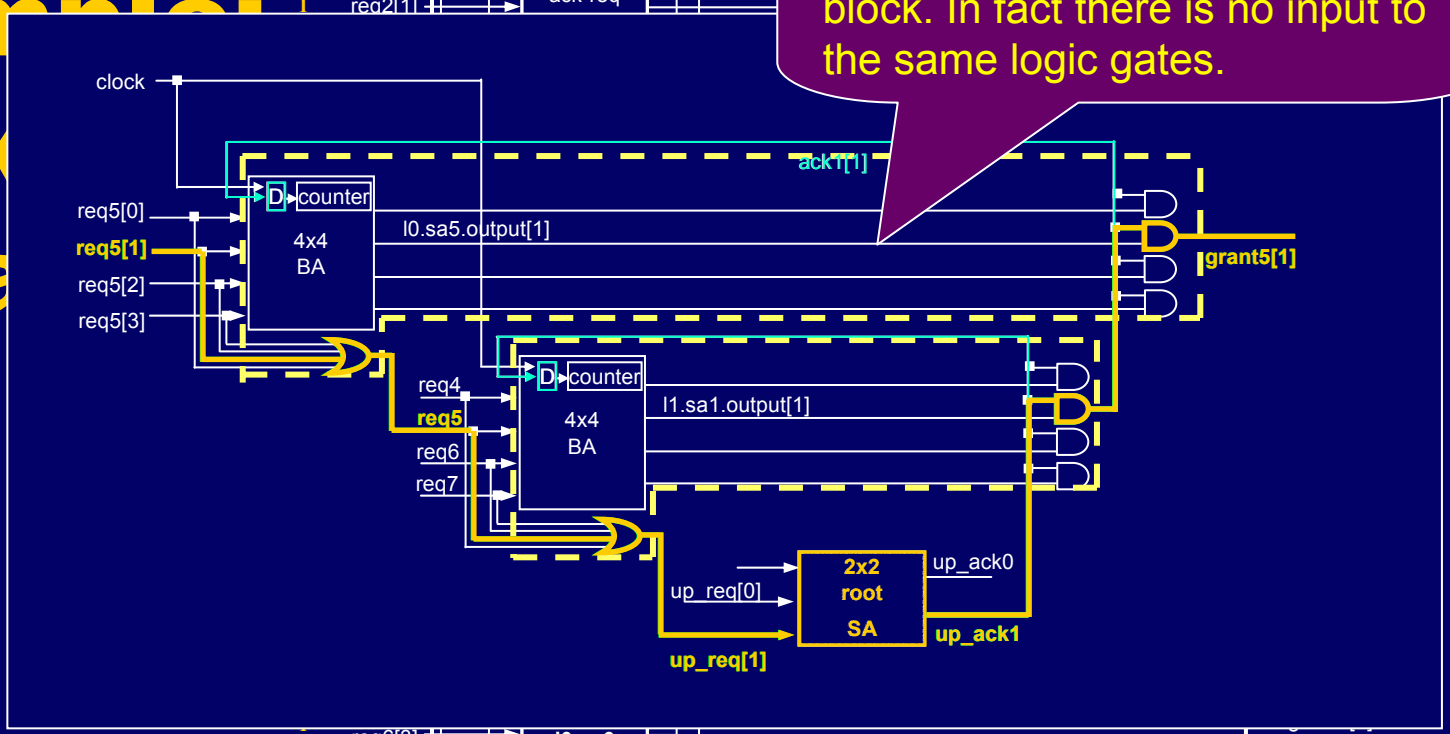
Example: 32x32 hierarchical SA



- 32 x 32 SA Critical Path:
  - •Travels through two 4-input OR gates in series
  - •Then through a 2x2 root SA
  - •Finally through two 2-input AND gates in series
  - •Results in 0.94ns delay using a TSMC
    from LEDA Systems.

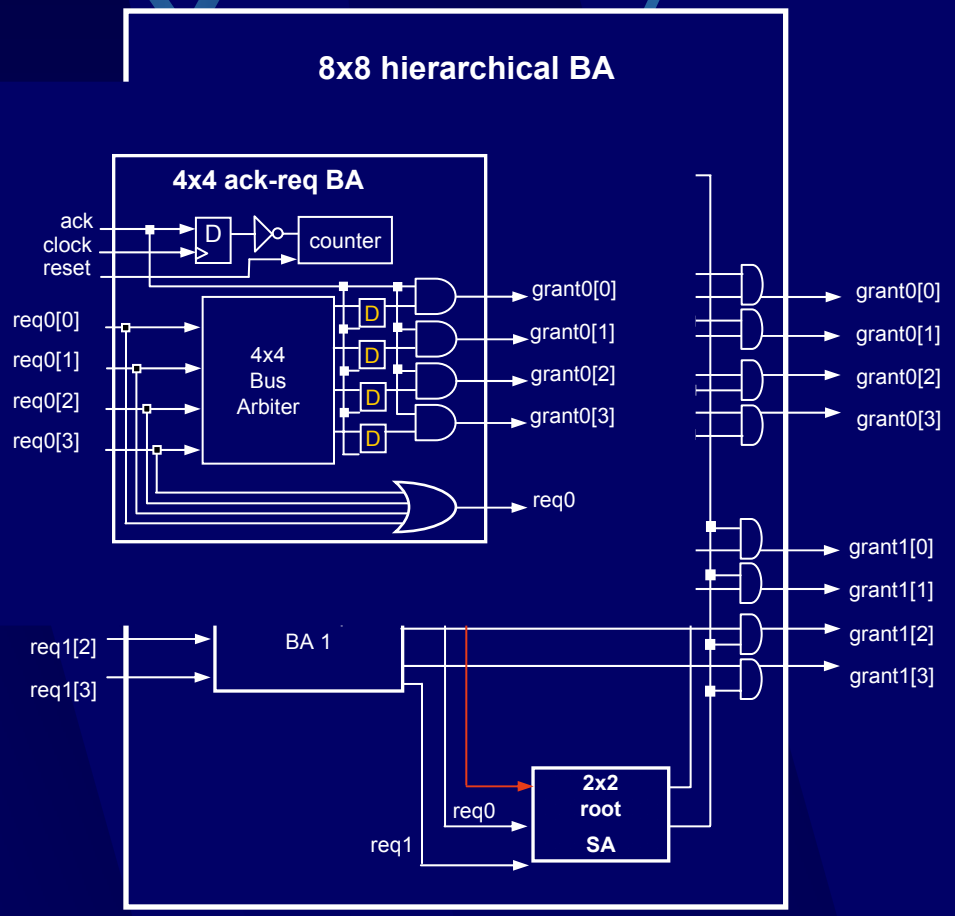
Exam
32x
hiera
ical

ack signals look like feedback path through the same logic block. In fact there is no input to the same logic gates.



Hierarchical BA

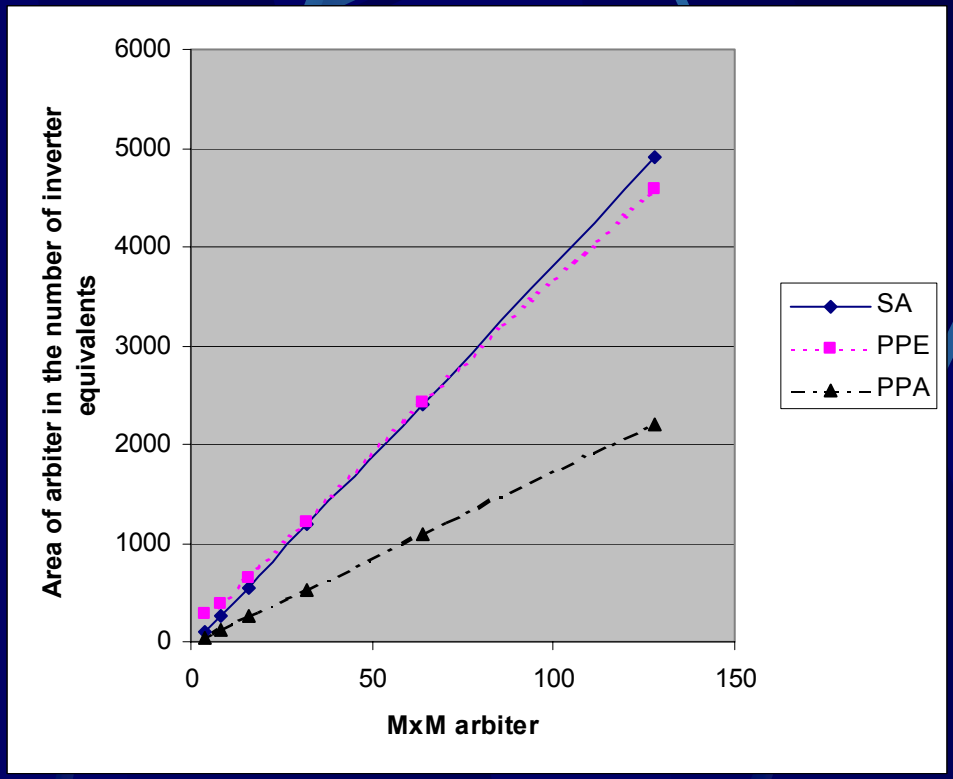
- Extra ‘ack’ input: indicating the completion of a single use of the bus
- Root token rotation:
  - By an ‘ack’ signal for a hierarchical BA
  - By clock signal for a hierarchical SA
- Possession of a bus for multiple cycles
- D latches triggered the positive edge of ack

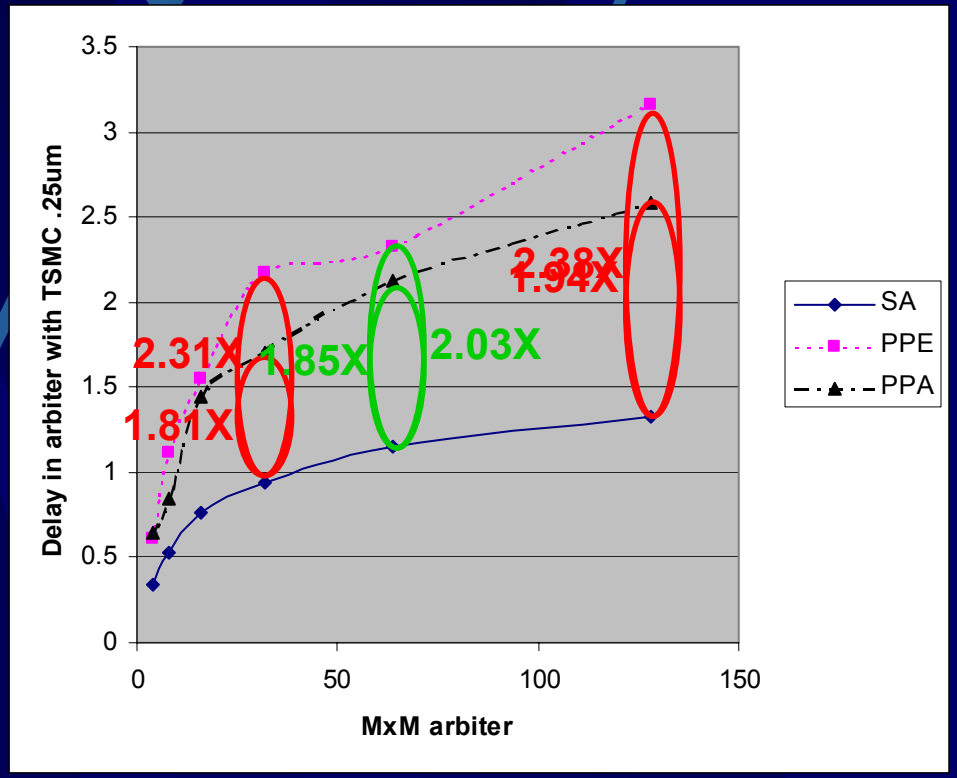


Outline

- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

Comparisons with PPE and PPA





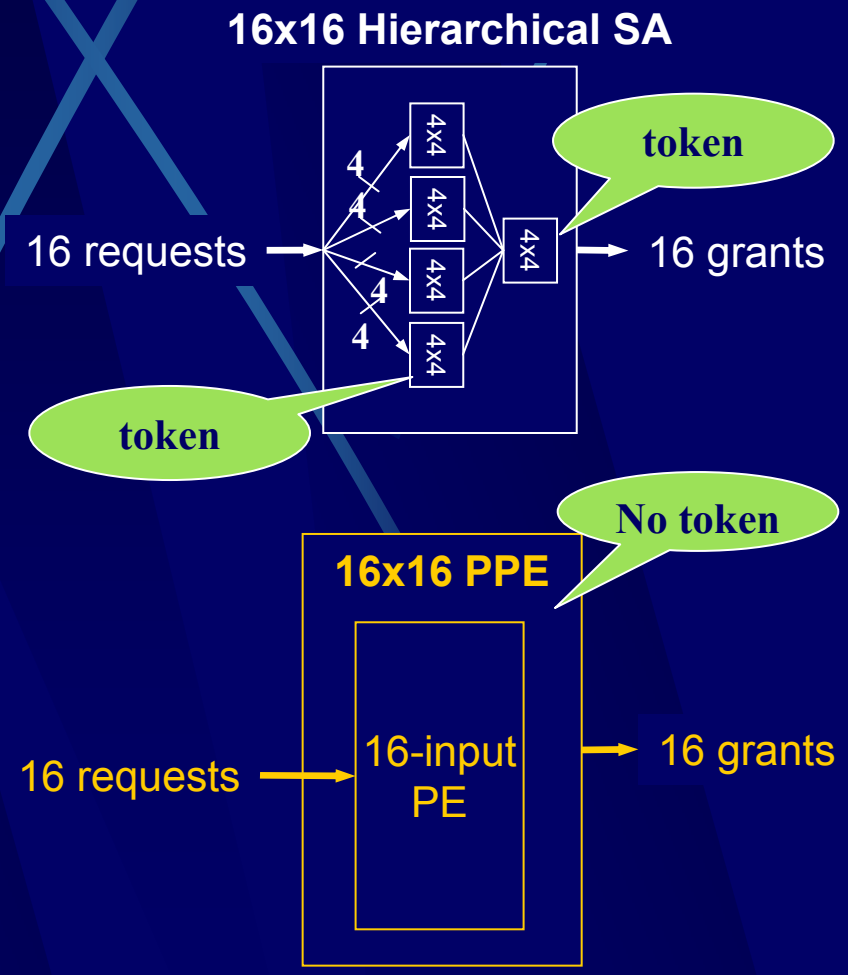
- Using TSMC 0.25μ std. cell library from LEDA Systems
- Synthesized by Design Compiler from Synopsys

Comparisons (Continued)

- The shortest delay results from
  - Limiting the size of switch arbiter blocks to 2x2, 3x3 and 4x4 to reduce the critical path delay
  - Due to the expansion of priority logic blocks compared with PPE
  - Reducing the number of levels in a hierarchy by preferring to use more 4x4 switch arbiter blocks compared with PPA

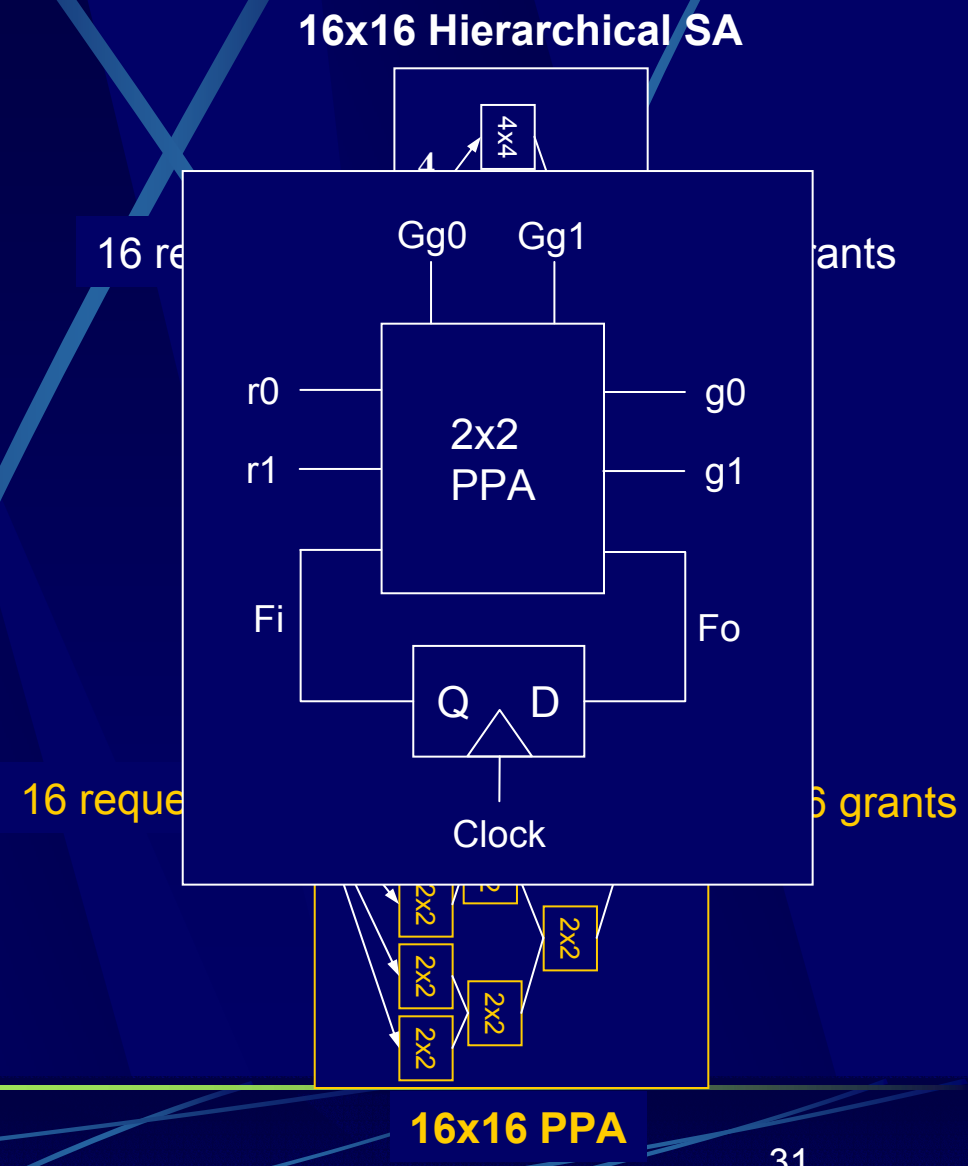
Key Insights

- Our hierarchical SA vs. PPE:
  - Much larger logic equations of PPE
  - More constrained to a specific structure in our SA
  - A distributed “state” kept by token



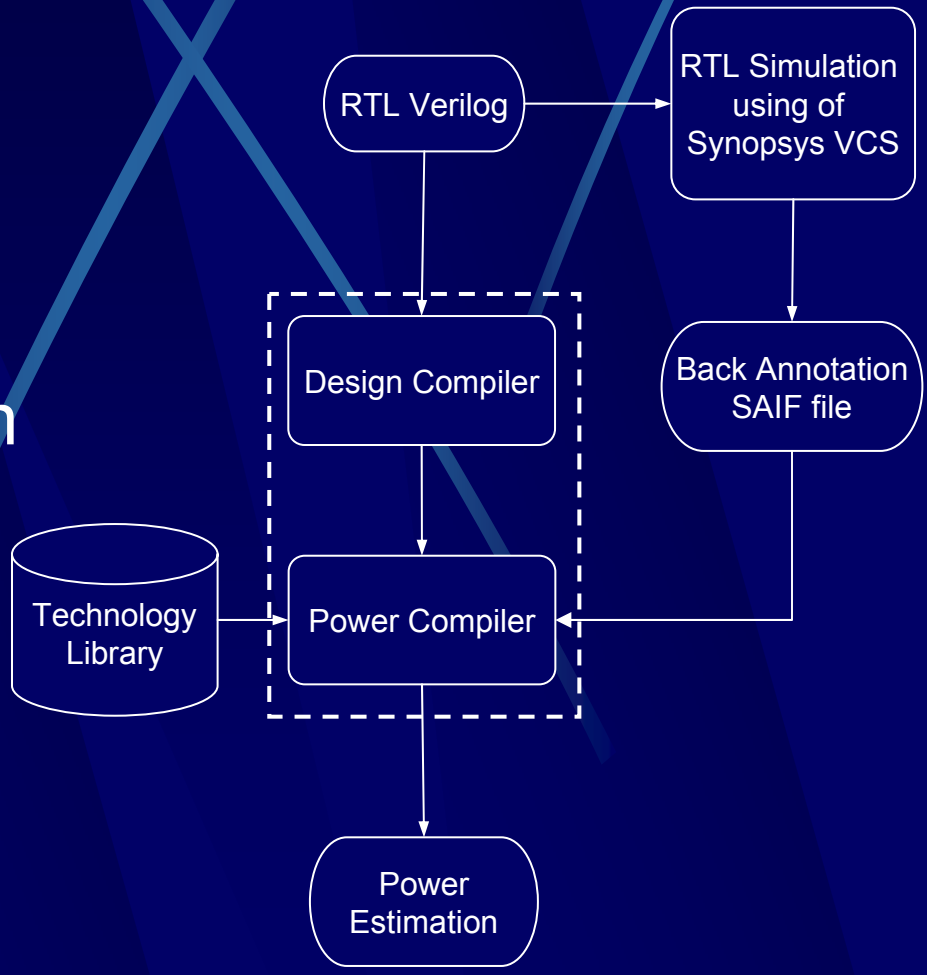
Key Insights (Continued)

- Our hierarchical SA vs. PPA:
  - PPA with token passing approach.
  - MxM PPA implemented by 2-input basic arbiter
  - Our hierarchical SA preferring utilizes 4-input switch arbiter blocks.

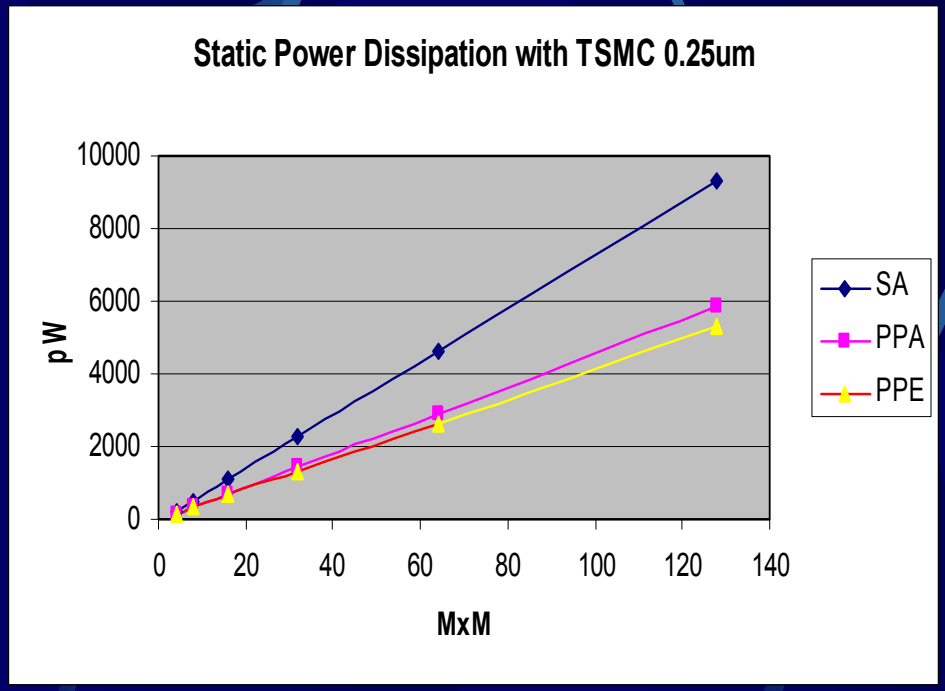


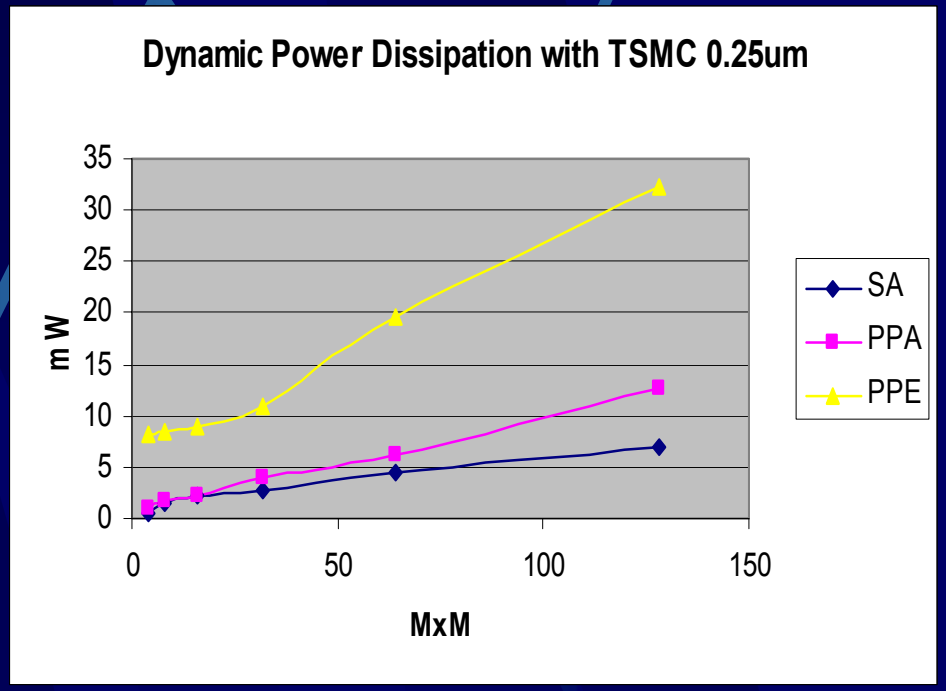
Power Comparison

- For dynamic power estimation, a Switching Activity Information File (SAIF) is extracted from Synopsys VCS.
- Assume congested network: all input requests are asserted.



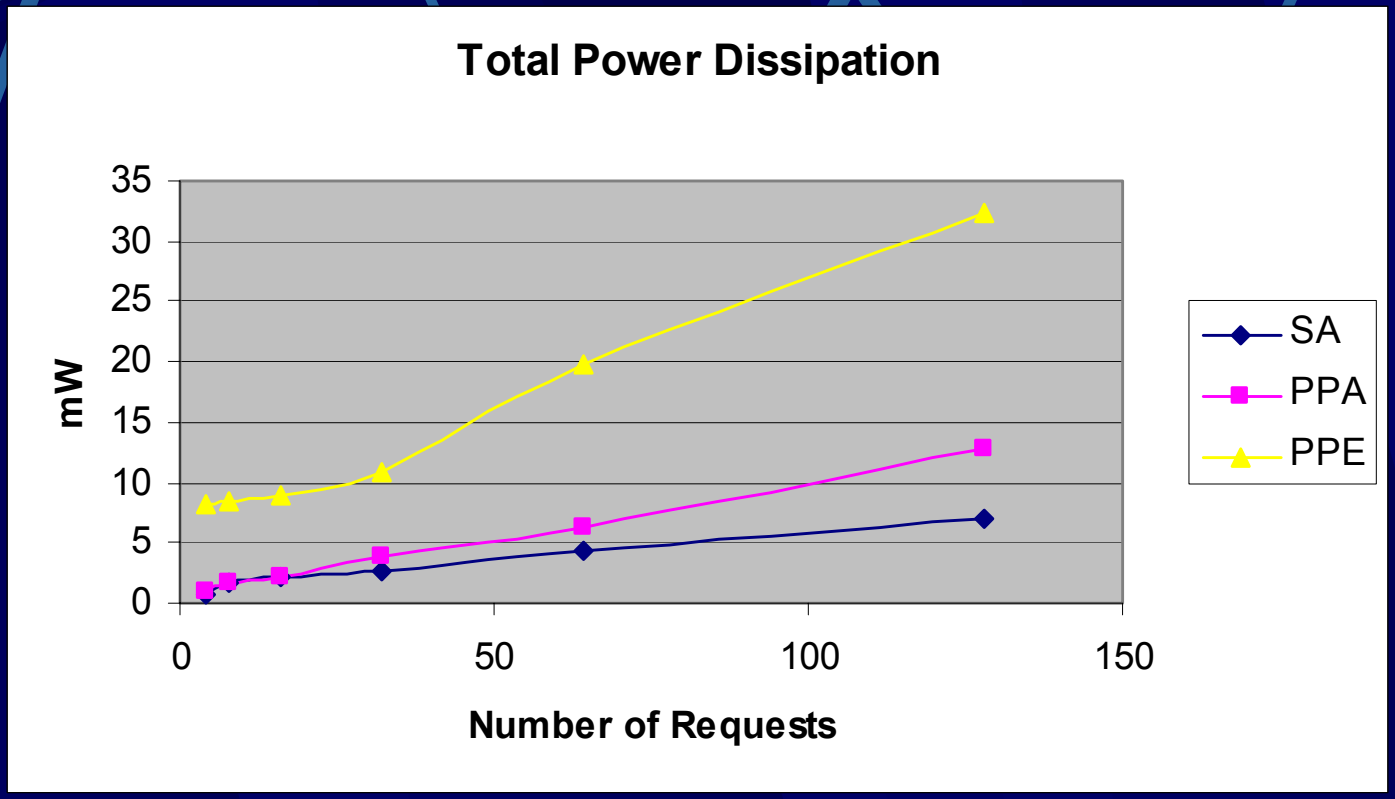
Power Comparison





- Using TSMC 0.25μ std. cell library from LEDA Systems
- Estimated by Power Compiler from Synopsys

Power Comparison

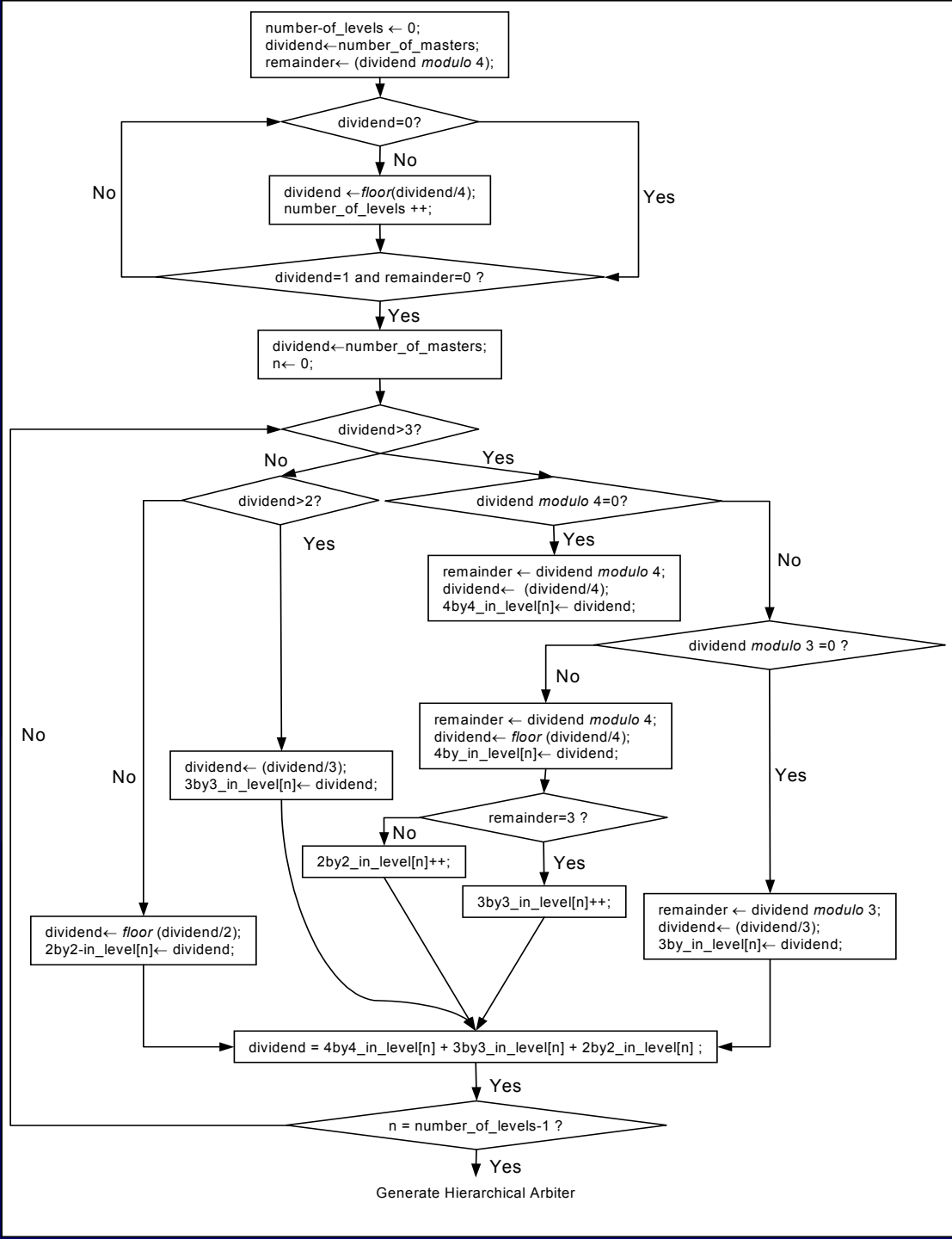


Outline

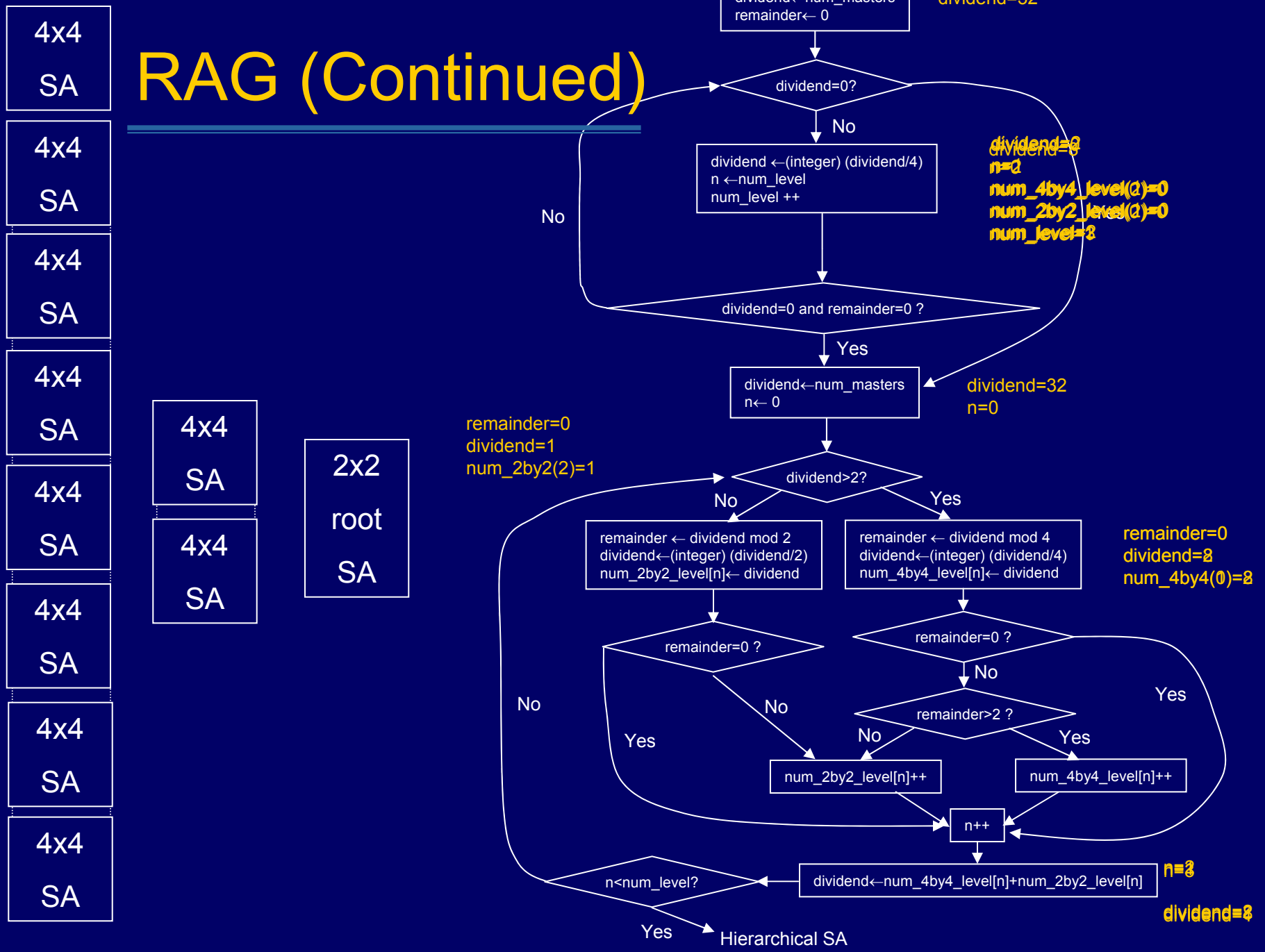
- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

RAG

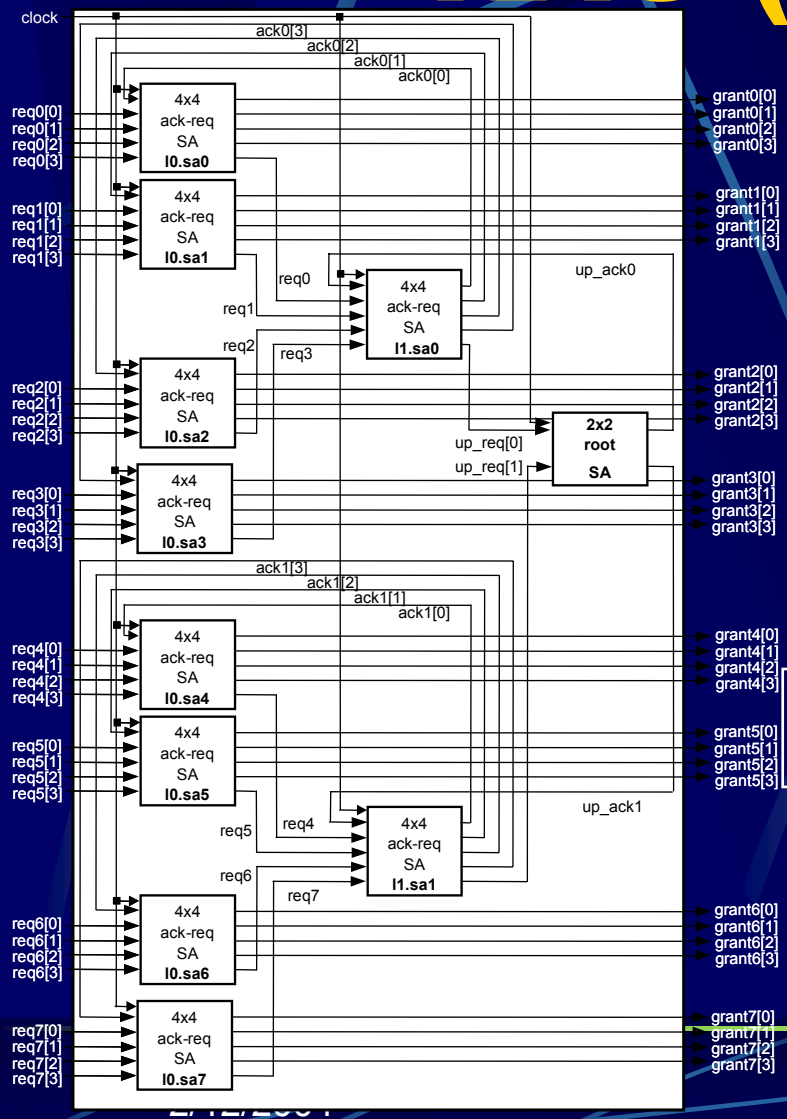
- A user specification: an arbiter type
- A user specification: the number of masters (M) to be arbitrated
- Output of RAG: a synthesizable Verilog code for a Bus Arbiter or a Switch Arbiter at the RTL

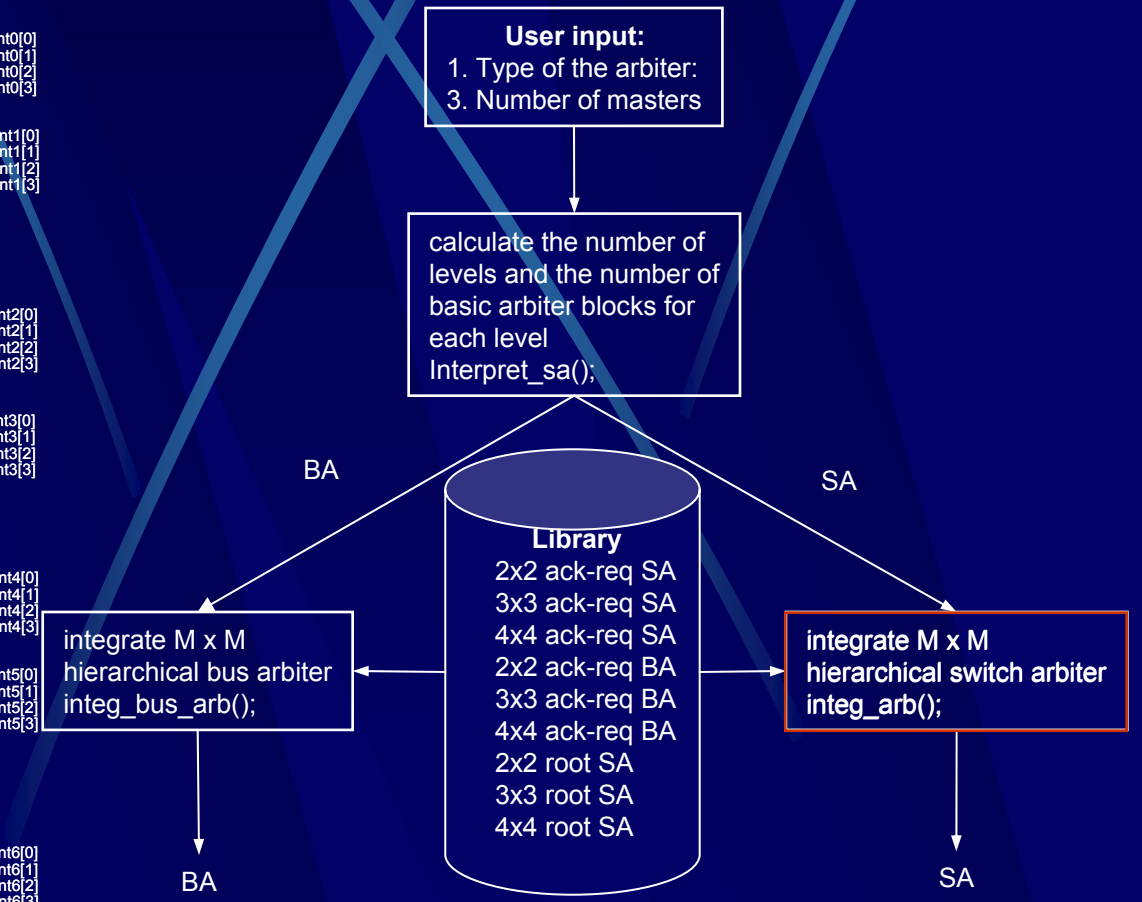


RAG (Continued)



RAG (Continued)

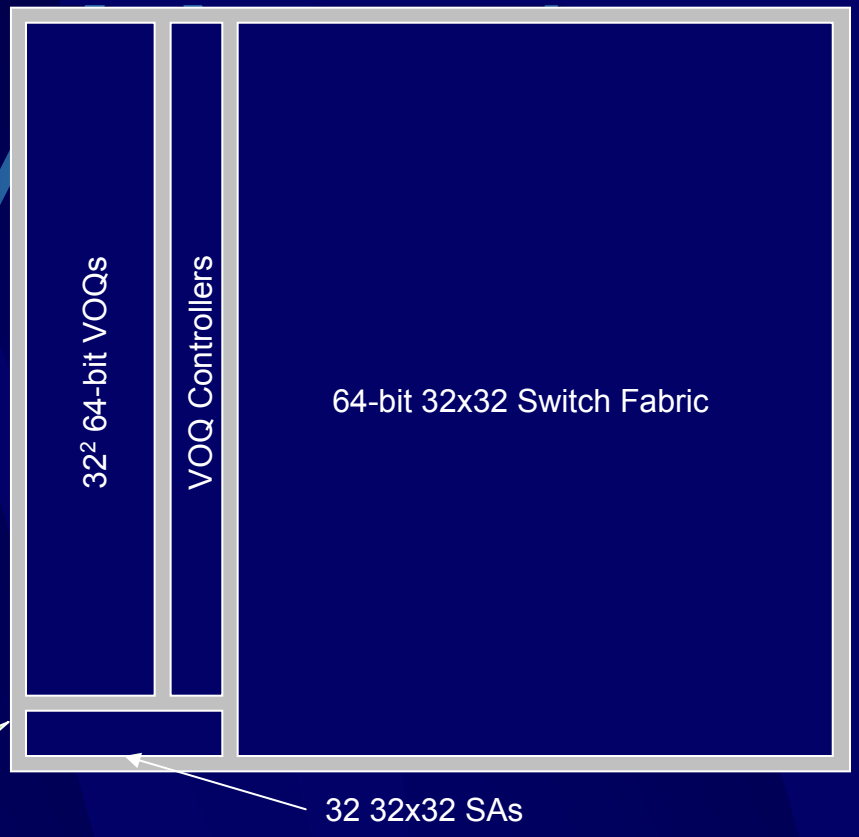




Speedup

- Throughput comparison for 64-bit switching 32x32 network switch
  - The longest delay of 32x32 switch = 0.63ns in .25μ TSMC → the maximum throughput of switch determined by arbitration delay
- Experimental setup:
  - Replacing SA in 32x32 network switch with our hierarchical SA, PPE and PPA

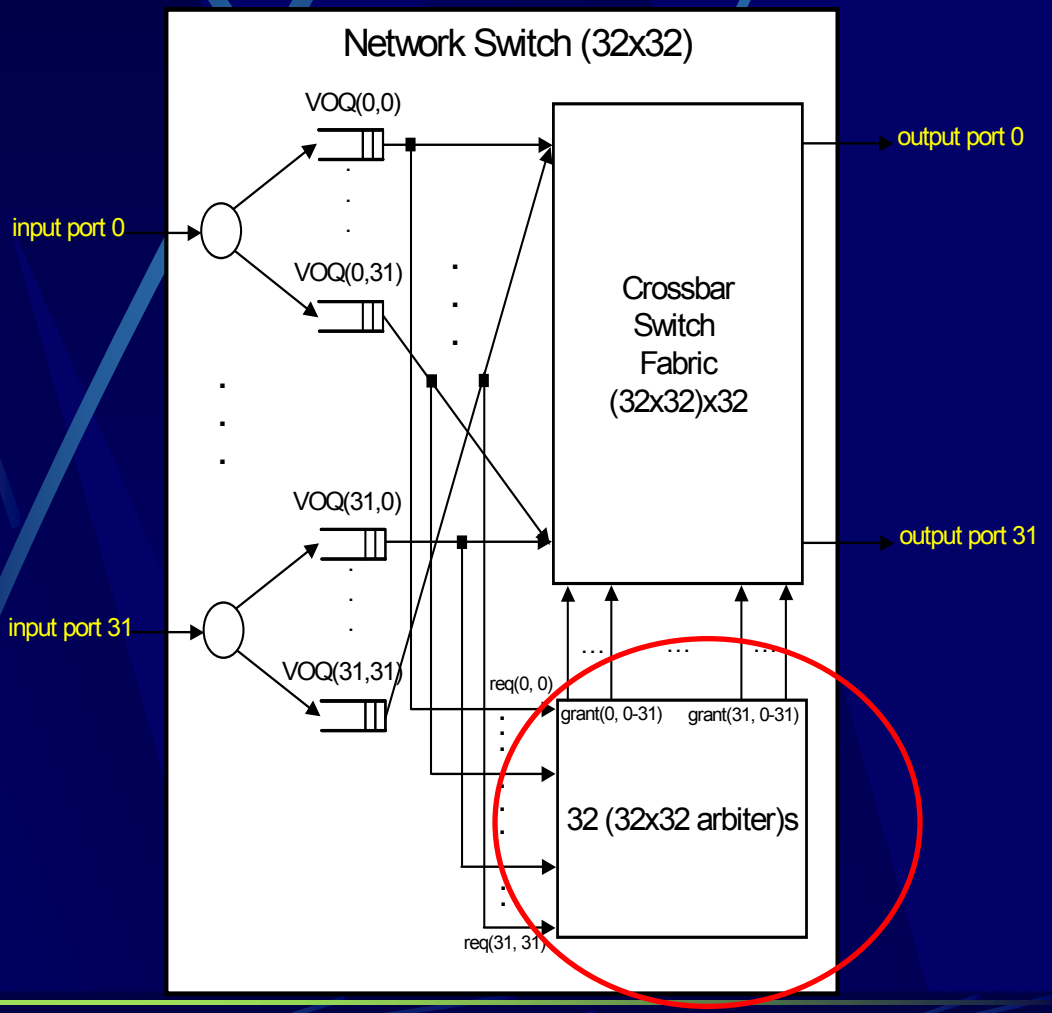
The floorplan of the 64-bit 32x32 switch fabric, VOQs, controllers and SAs



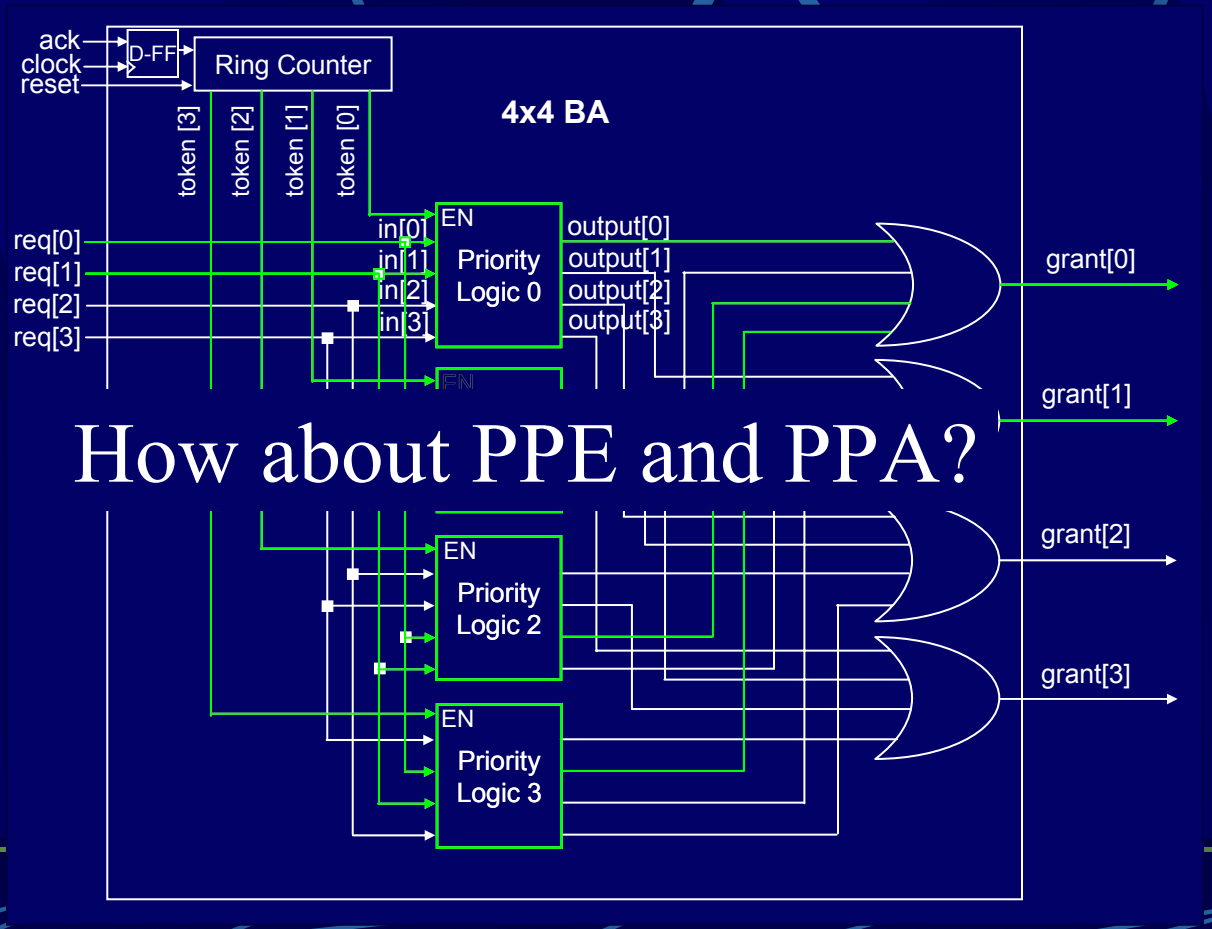
Area: 125.64mm²

Speedups (Continued)

- Speedup in .25µ TSMC
  - Our hierarchical 32x32 SA: 64 bits@0.94ns delay → 2.18Tbps
  - 32x32 PPA: 64bits@1.70ns delay → 1.20Tbps
  - 32x32 PPE: 64bits@2.17ns delay → 0.94Tbps
  - Results: The throughput achieved by our SA > 1.8X than PPA and > 2.3X than PPE



Perfectly Fair?



req #	# of grants
0	3
1	1

Fairness Simulation

- Joint work with Dr. Riley and Tom Cheng
- Simulation with the 32x32 hierarchical SA
- Several input patterns applied for uniform traffic simulation
- Bursty and TCP traffics applied
- Traffic intensity, ρ: a measure of the demand on the switch for bursty and TCP traffics.

Simulation Results

- Uniform traffic: metrics are the number of grants

	Run 1	Run 2	Run 3
4x4 ack-req input	Grant, all asserted	Grants, 0 , 1 asserted	Grants, 0, 1, 2 asserted
0	250000	749000	499999
1	250000	250000	250000
2	250000	0	250000
3	249999	0	0

Simulation Results (Continued)

- Bursty traffic: metrics are average delay

4x4 ack-req input	Delay, ρ = 2.0	Delay, ρ = 1.0	Delay, ρ = 0.9	Delay, ρ = 0.5	Delay, ρ = 0.1
0	31.43	16.65	4.46	1.98	1.05
1	31.23	15.59	4.60	1.93	1.10
2	30.95	17.14	4.61	1.82	1.01
3	31.11	16.44	4.63	2.00	1.10

Simulation Results (Continued)

- TCP traffic: metrics are average delay

4x4 ack-req input	Delay, $\rho = 2.0$	Delay, $\rho = 1.0$	Delay, $\rho = 0.9$	Delay, $\rho = 0.5$	Delay, $\rho = 0.1$
0	18.12	16.75	11.68	1.95	1.10
1	19.91	19.48	11.29	1.95	1.10
2	20.44	19.71	10.69	1.93	1.10
3	19.22	17.54	12.13	1.94	1.10

Outline

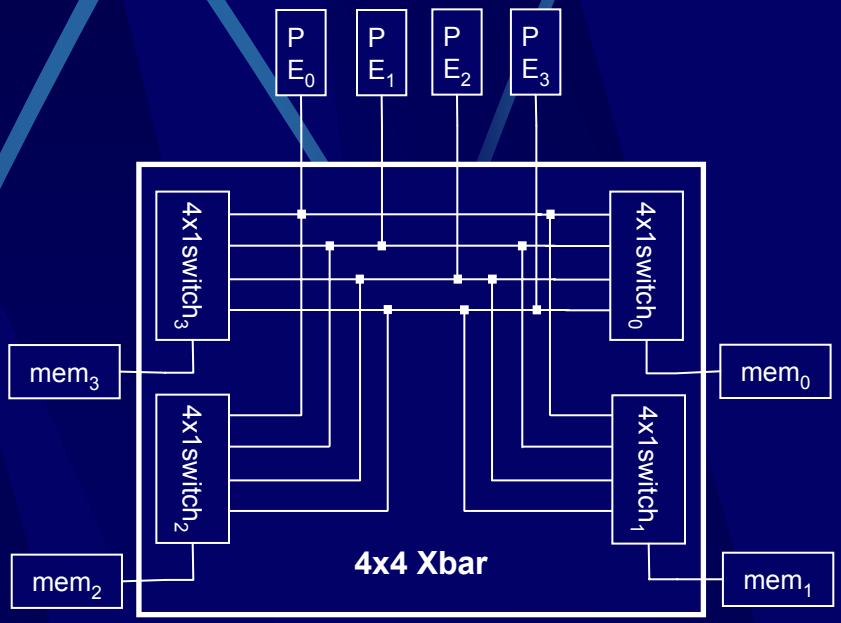
- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

Why Xbar and X-Gt?

- Xbar:
  - Demand on multiple communication channels
  - Concurrent accesses to resources
- X-Gt:
  - Generation customized Xbar on-the-fly
  - The generated Xbar in RTL Verilog

Xbar Switch

- One configuration for 4x4 case
- Each 4x1 switches: Comparing physical addresses from PEs and judging if addresses belong to the address space of the attached memory block
- Maximum of 4 concurrent memory access in one cycle

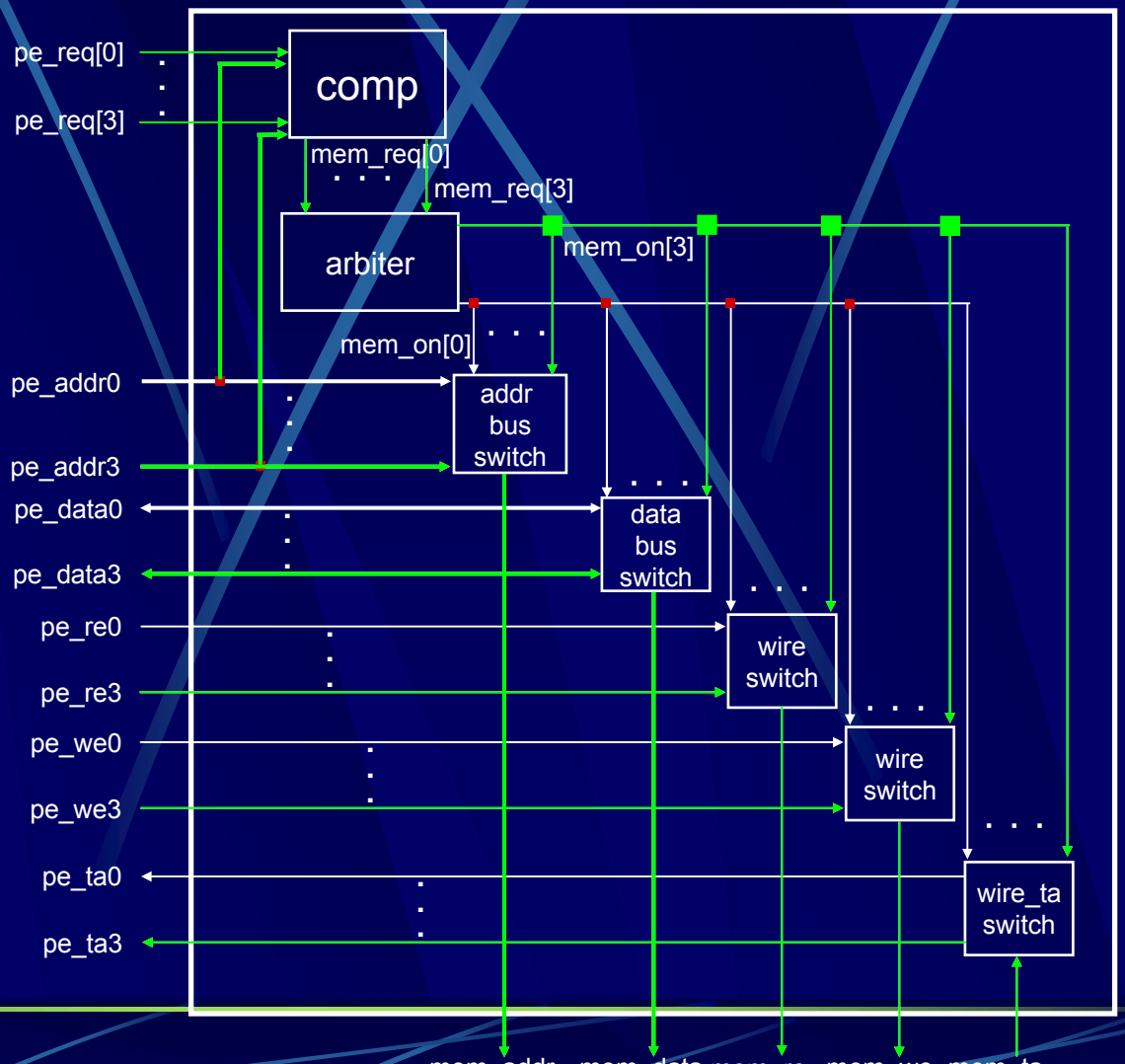


Xbar (Continued)

- An MxN switch consisting of N Mx1 switches, M = # of PEs & N = # of memory blocks
- An assertion of ‘mem_req’ inside a Mx1 switch: depending on an ‘address’ input from a PE
- Grant: given by an arbiter by the assertion of ‘mem_on’ signals
- Arbitration in round-robin order: handling M requests from M processors

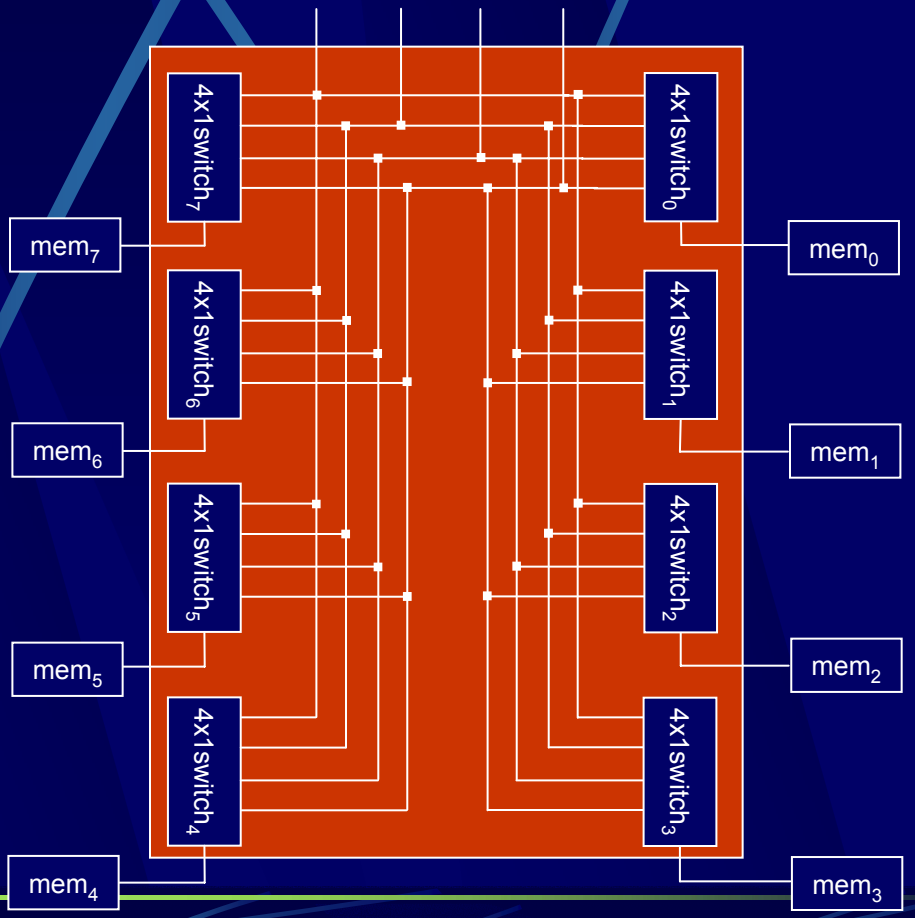
4x1 Switch Example

- Scenario:
  - Request from PE 0 and PE 3
  - PE 3 granted



Example of Xbar Configuration

- 4x8 Xbar:
  - Supporting 4 PEs and 8 memory modules
  - Connecting a particular PE signals to a specific memory module by a 4x1 switch according to a physical address from a PE



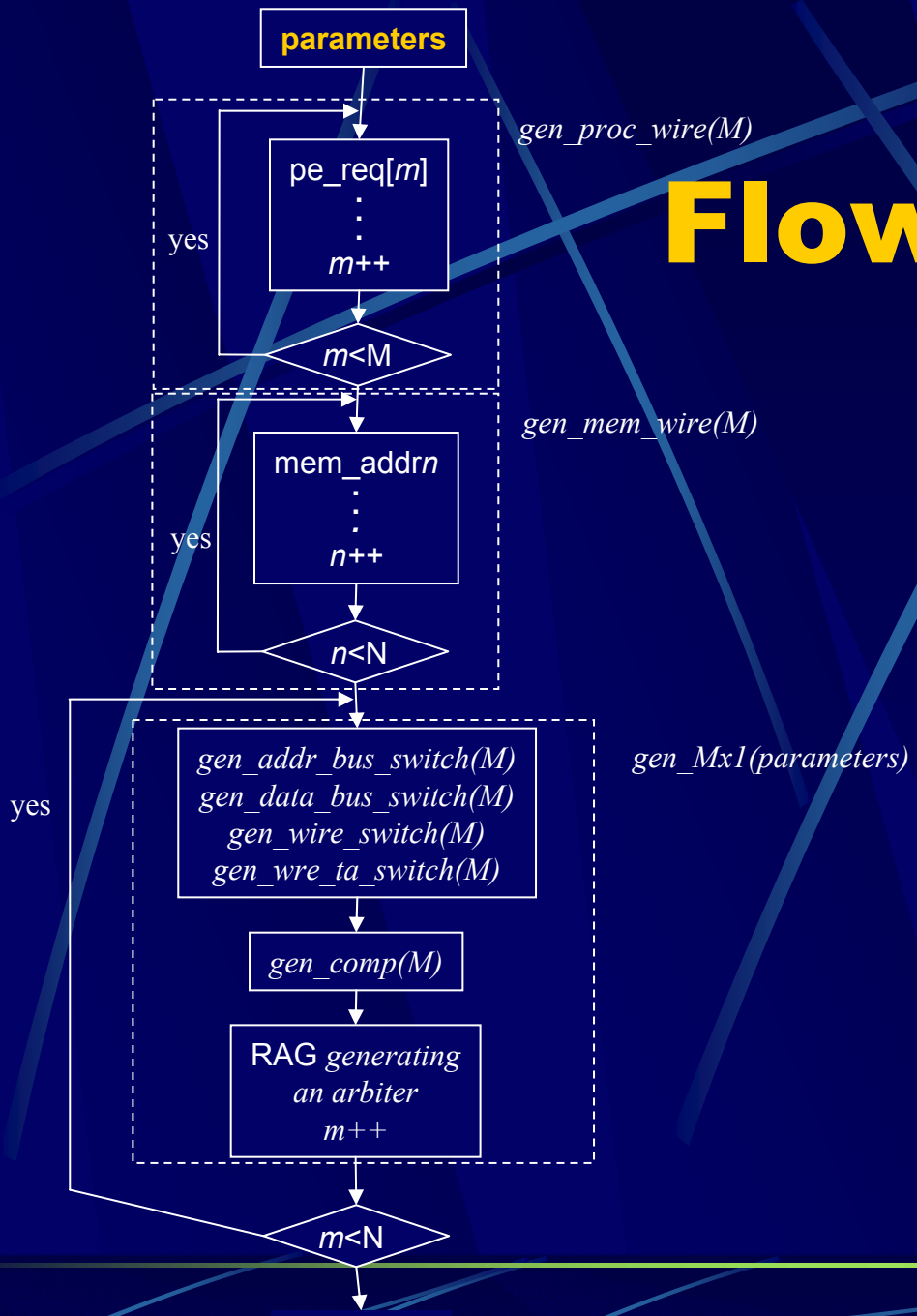
X-Gt (Xbar Generator)

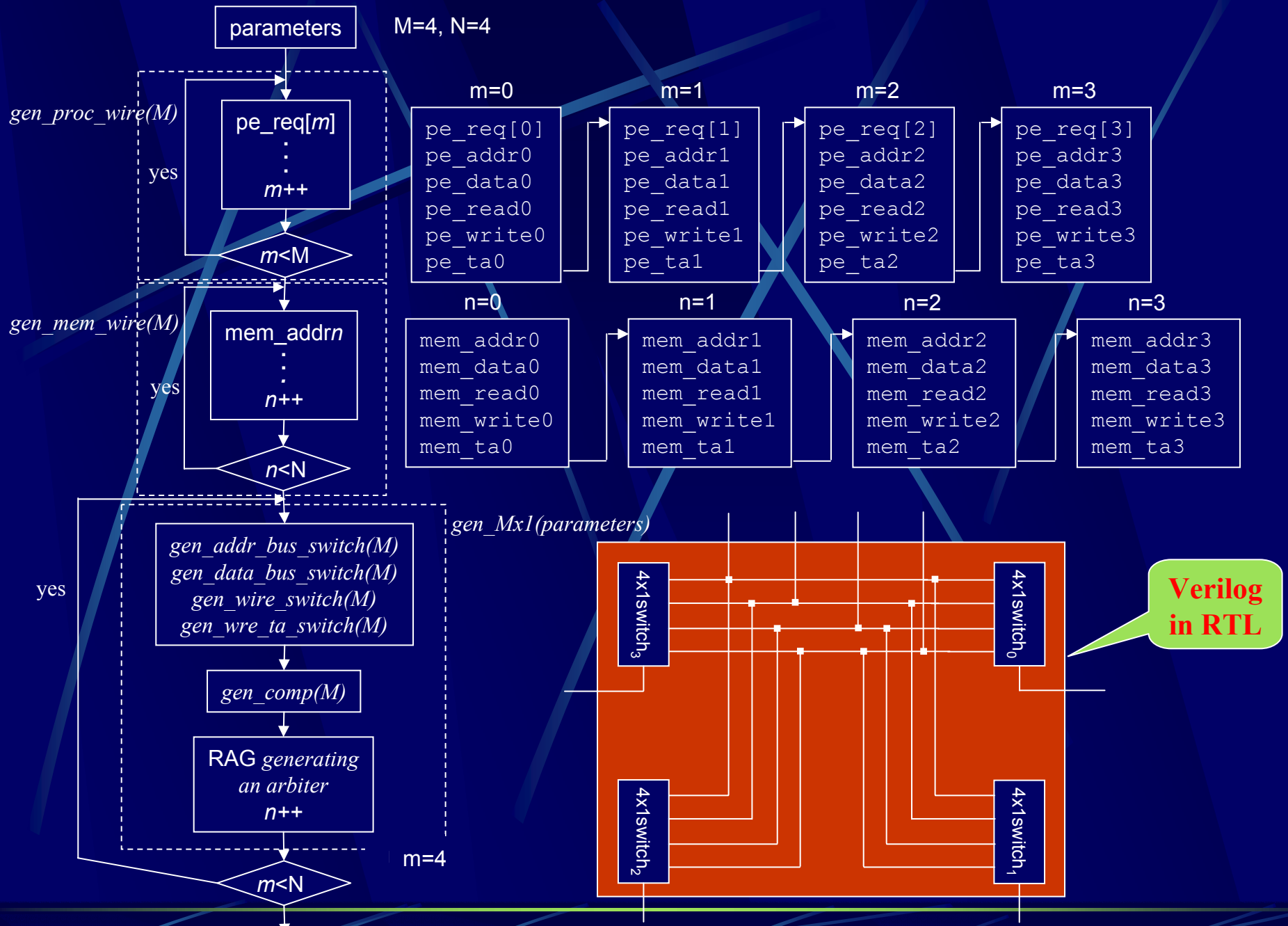
- Generation of customized Xbar in Verilog at the RTL
  - An arbiter generated by RAG
  - Parameterizable switch blocks in an Mx1 switch
  - All submodules connected by wire names

X-Gt (Continued)

- Parameters:
  - The number of PEs that determines M in an MxN Xbar
  - The number of memory blocks that determines N in an MxN Xbar
  - The total global memory size that determines (pe_)address bus width
  - The data bus width of each PE determined by PE type
  - The (mem_)address bus width determined by the size of memory block

Flowchart of X-Gt

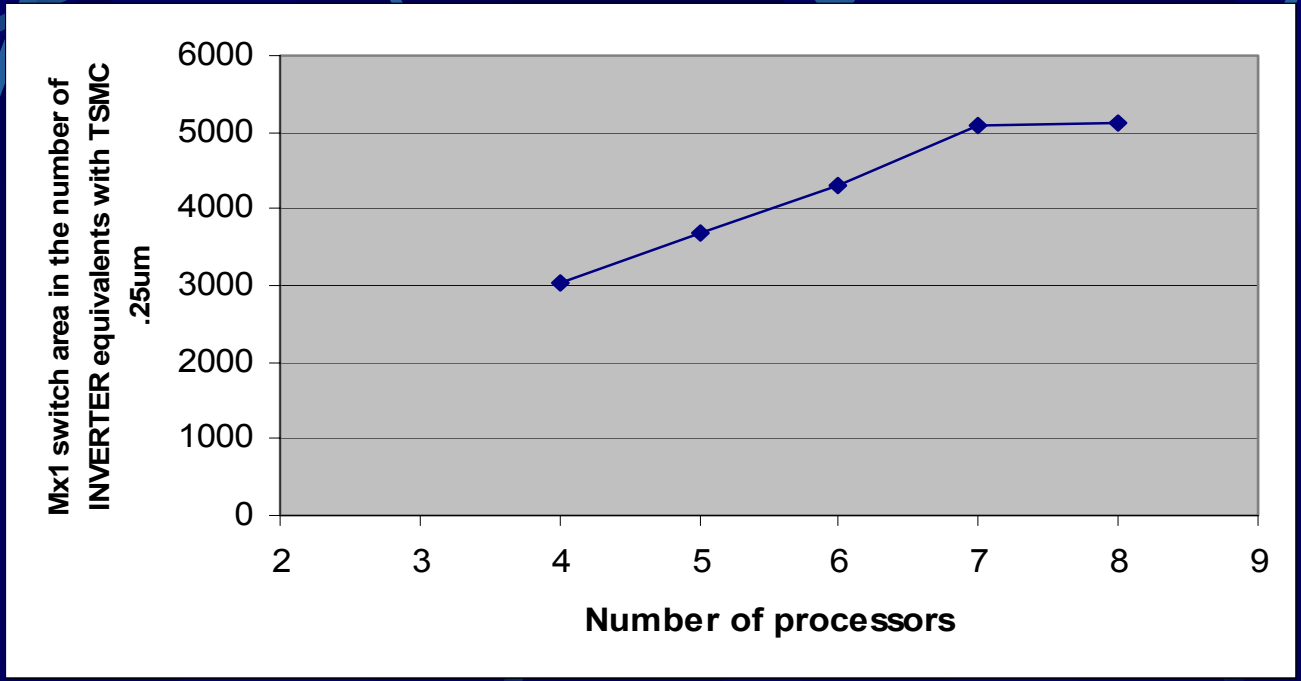



parameters
M=4, N=4
gen_proc_wire(M)
pe_req[m]
m++
yes
m<M
gen_mem_wire(M)
mem_addrn
n++
yes
n<N
yes
gen_addr_bus_switch(M)
gen_data_bus_switch(M)
gen_wire_switch(M)
gen_wre_ta_switch(M)
gen_comp(M)
RAG generating an arbiter
n++
m=4
m<N
4x4 Xbar
m=0
pe_req[0]
pe_addr0
pe_data0
pe_read0
pe_write0
pe_ta0
m=1
pe_req[1]
pe_addr1
pe_data1
pe_read1
pe_write1
pe_ta1
m=2
pe_req[2]
pe_addr2
pe_data2
pe_read2
pe_write2
pe_ta2
m=3
pe_req[3]
pe_addr3
pe_data3
pe_read3
pe_write3
pe_ta3
n=0
mem_addr0
mem_data0
mem_read0
mem_write0
mem_ta0
n=1
mem_addr1
mem_data1
mem_read1
mem_write1
mem_ta1
n=2
mem_addr2
mem_data2
mem_read2
mem_write2
mem_ta2
n=3
mem_addr3
mem_data3
mem_read3
mem_write3
mem_ta3
gen_Mx1(parameters)
4x1switch3
4x1switch0
4x1switch2
4x1switch1
Verilog in RTL

Outline

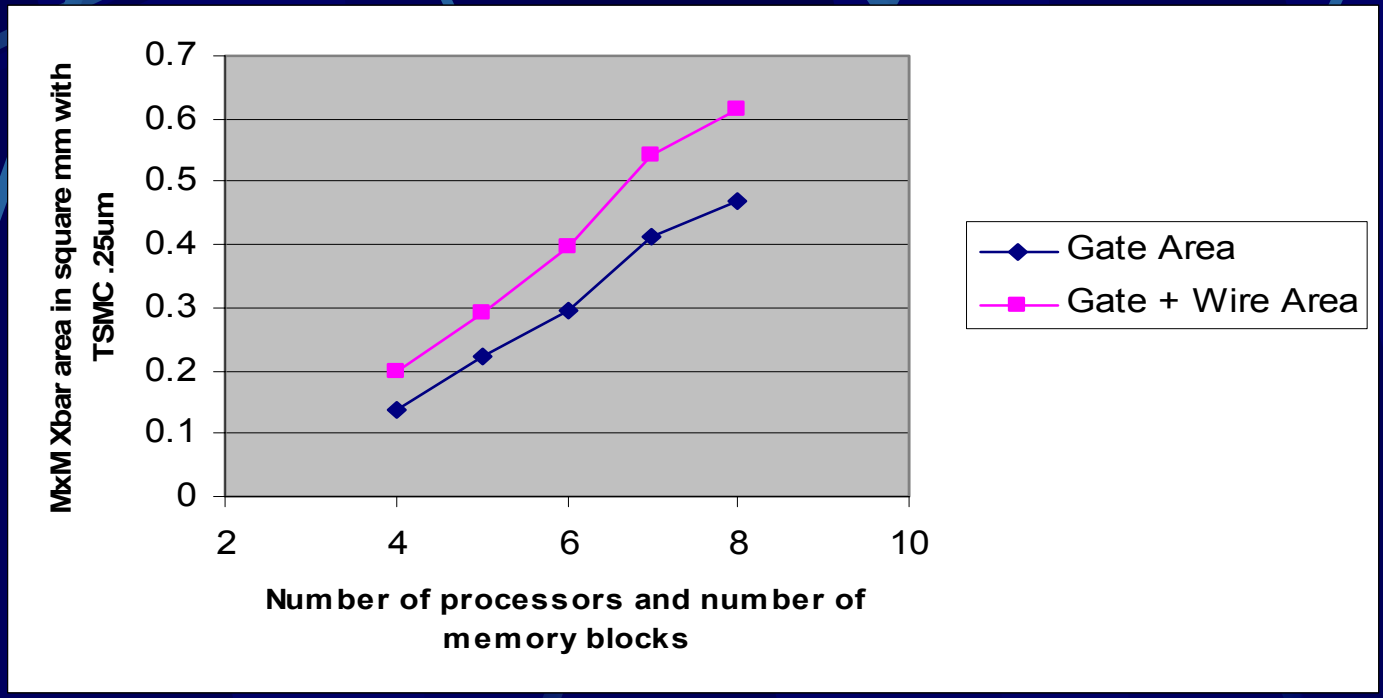
- Terminology
- Origin and history of problems:
  - Arbiter design: PPE and PPA
  - Crossbar switch design: “Smart” Memory
- Arbiter design
- Arbiter experiments
- RAG: Round-robin Arbiter Generator
- X-Gt: Xbar Generator
- Xbar experiments
- Conclusion

Xbar Synthesis: Mx1 Gate Area



- Using TSMC 0.25μ std. cell library from Artisan Components
- Estimated by Design Compiler from Synopsys

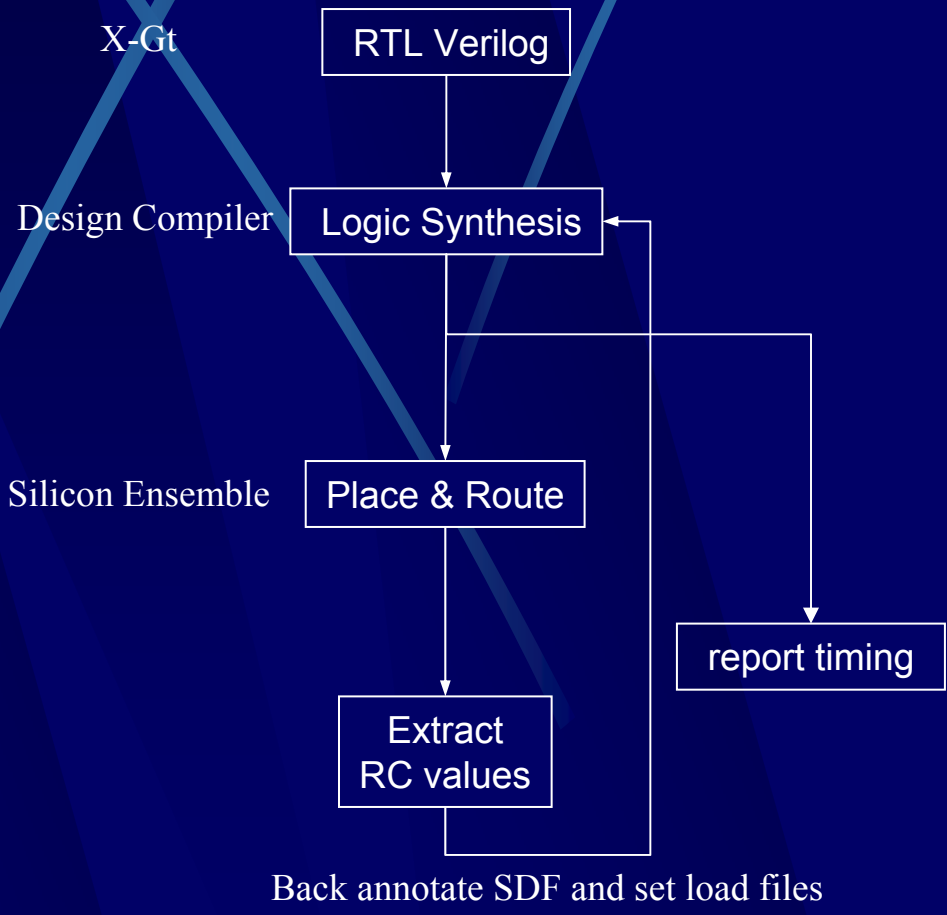
Xbar Synthesis: MxM Area



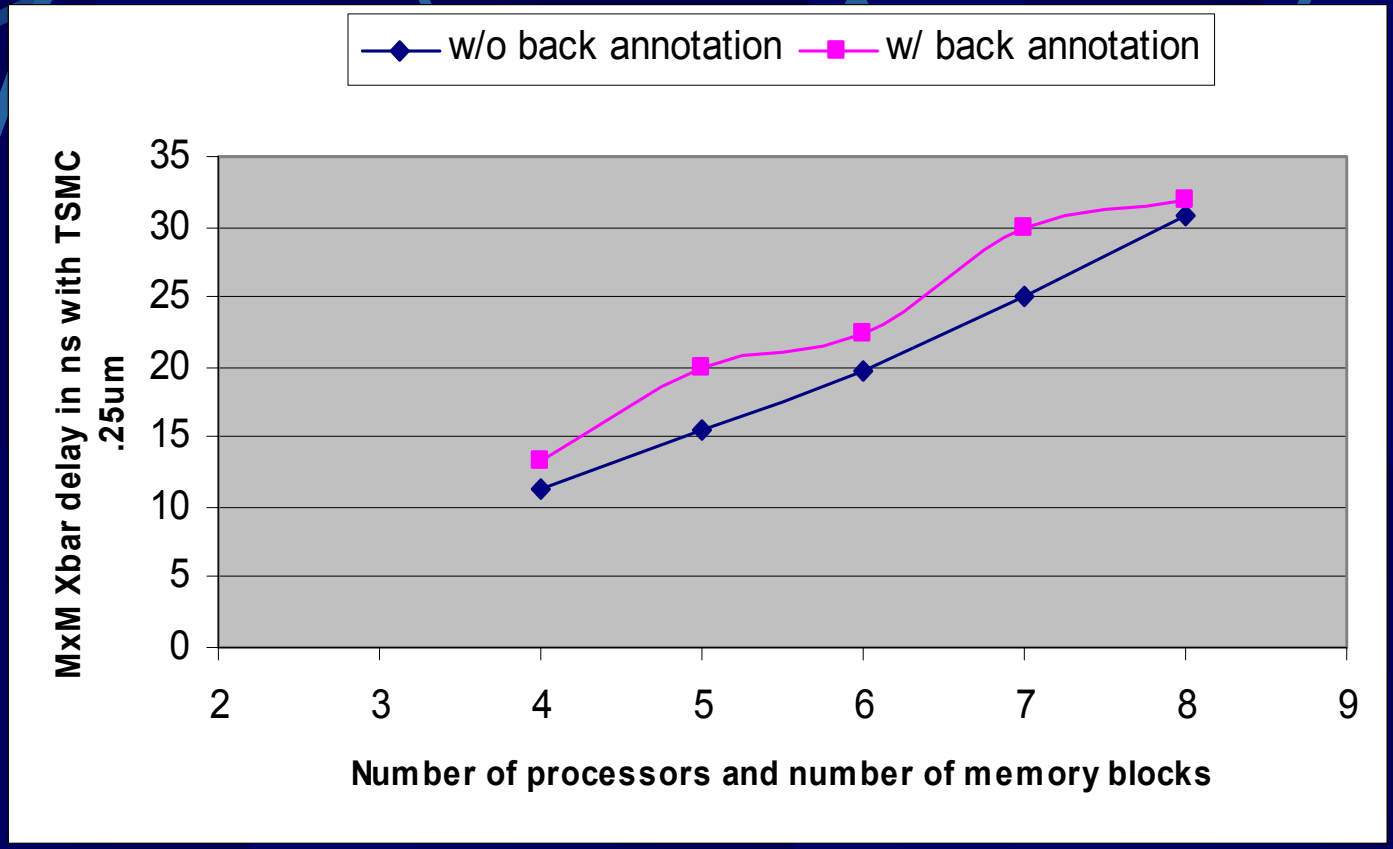
- Using TSMC 0.25μ std. cell library from Artisan Components
- Gate Area estimated by Design Compiler from Synopsys
- Gate + Wire Area estimated by Silicon Ensemble from Cadence

Back Annotated Timing

- An RTL Verilog (output of X-Gt) synthesized by DC with TSMC .25µ from Artisan Components
- Back annotation of SDF and set_load files to logic synthesis stage after place and route using Synopsys Design Compiler and Cadence Silicon Ensemble



MxM Xbar Delay



Conclusion

- Showed how we design hierarchical SA and hierarchical BA.
- Demonstrated $\geq$ 1.8X speedups for our hierarchical SA over PPE and PPA.
- Illustrated how RAG generates a hierarchical SA.
- Demonstrated Xbar switch and showed how X-Gt generates an Xbar according to user specifications.
- Illustrated gap between delays from logic synthesis stage and delays from physical synthesis stage for MxM Xbars.

Publications

- **As (co-) First Author:**
  - K. Ryu, E. S. Shin, and V. Mooney, “A Comparison of Five Different Multiprocessor SoC Bus Architectures,” *Proceedings of the 2001 Euromicro Symposium on Digital Systems Design (DSD’01)*, September 2001.
  - E. S. Shin, V. Mooney, and G. Riley, “Round-robin Arbiter Design and Generation,” *Proceedings of 15th International Symposium on System Synthesis (ISSS’02)*, October 2002.
  - E. S. Shin, V. Mooney, and G. Riley, “Round-robin Arbiter Design and Generation,” Georgia Institute of Technology Technical Report, GIT-CC-02-38, Available HTTP: http://www.cc.gatech.edu/tech_reports/index.02.html
  - M. Shalan, E. S. Shin and V. Mooney, “DX-Gt: Memory Management and Crossbar Switch Generator for Multriprocessor System-on-a-Chip,” *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI’03)*, April 2003.
  - E. S. Shin, V. Mooney, and G. Riley, “Round-robin Arbiter Design and Generation,” submitted to IEEE Transactions on CAD.

Publications (Continued)

- Other publications:
  - P. Cheng, Eung S. Shin, G. R. Riley and V. J. Mooney III, “SASim: Switch Arbiter Simulator,” Georgia Institute of Technology Technical Report, GIT-CC-03-38, Available HTTP: http://www.cc.gatech.edu/tech_reports/index.03.html.
  - A. Talpasanu , J. A. Davis, E. S. Shin and V. J. Mooney, “Crossbar Switch Interconnect Delay Calculation,” Georgia Institute of Technology Technical Report, GIT-CC-03-37, Available HTTP: http://www.cc.gatech.edu/tech_reports/index.03.html.
- Provisional Patent:
  - E. S. Shin and V. Mooney, “Fast Distributed Switch Arbiter Design and Generation.”