DX-Gt: Memory Management and Crossbar Switch Generator for Multiprocessor System-on-a-Chip

Mohamed Shalan*, Eung Shin* and Vincent J. Mooney III⁺
{shalan, shin, mooney}@ece.gatech.edu

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

SASIMI Workshop, April 3-4, 2003

Agenda

- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

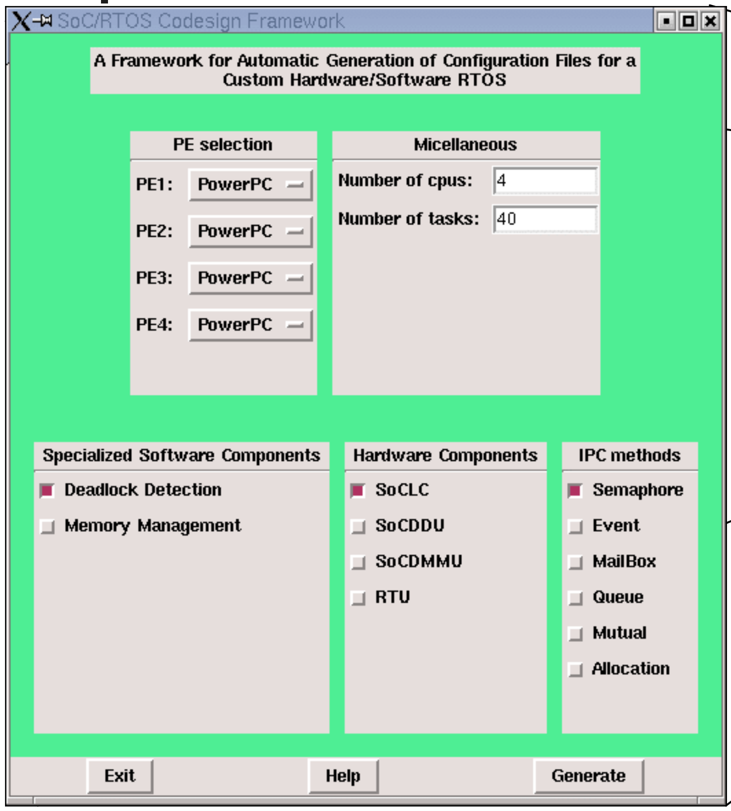
Introduction

- In next five years multiprocessor SoCs will be dominated by designs with four to eight processors and on-chip DRAM of 16Mbytes to 128Mbytes
- As the number of transistors on a single chip increases rapidly, there is a productivity gap between the increasing number of available transistors and the design time.

Introduction

- To reduce the productivity gap, designers use IP cores.
- An IP core should be customized before being used in a system different than the one for which it was designed.
- Either an engineer must spend significant effort altering the core by hand or else an IP generators should be used.

Motivation



SoC/RTOS Codesign Framework

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:	40

Specialized Software Components

- [✓ Deadlock Detection]
- Memory Management

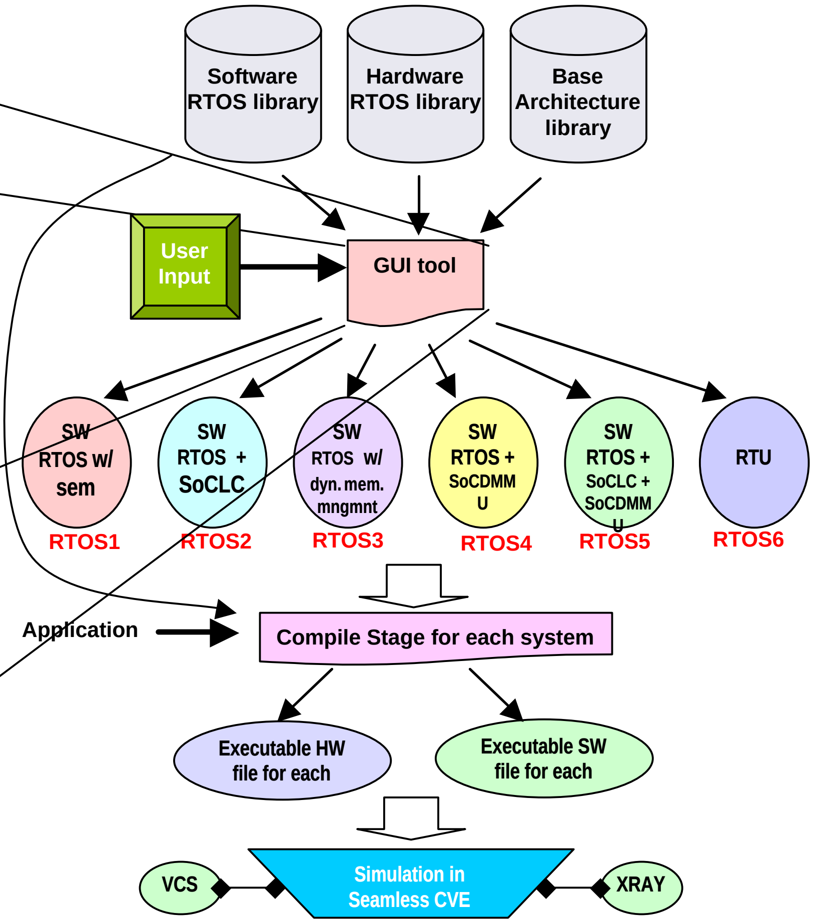
Hardware Components

- [✓ SoCLC]
- SoCDDU
- SoCDMMU
- RTU

IPC methods

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

Exit | Help | Generate



Software RTOS library
Hardware RTOS library
Base Architecture library
User Input
GUI tool
SW RTOS w/ sem
RTOS1
SW RTOS + SoCLC
RTOS2
SW RTOS w/ dyn. mem. mngmnt
RTOS3
SW RTOS + SoCDMMU
RTOS4
SW RTOS + SoCLC + SoCDMMU
RTOS5
RTU
RTOS6
Application
Compile Stage for each system
Executable HW file for each
Executable SW file for each
VCS
Simulation in Seamless CVE
XRAY

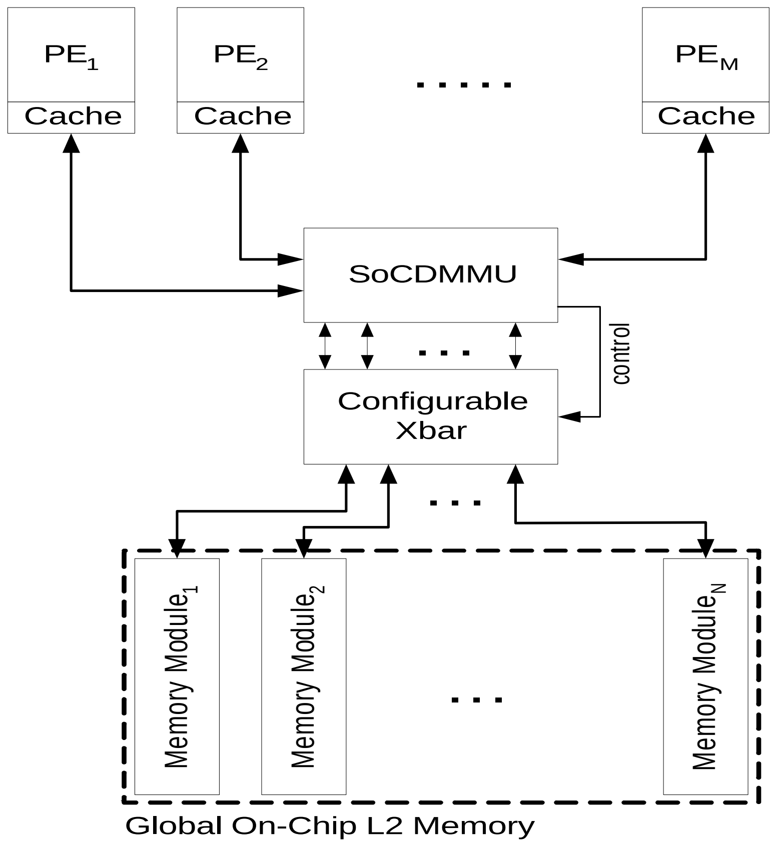
Previous Work

- No known previous work in SoCDMMU *generation* or Xbar *generation*
- Previous work exists in memory management approaches similar to SoCDMMU
- Previous work exists in Xbar design

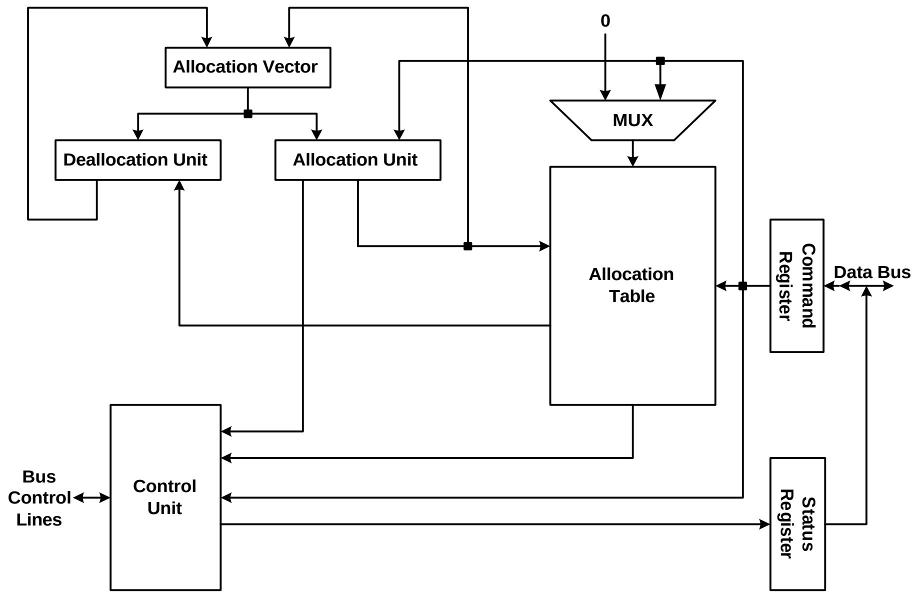
Agenda

- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

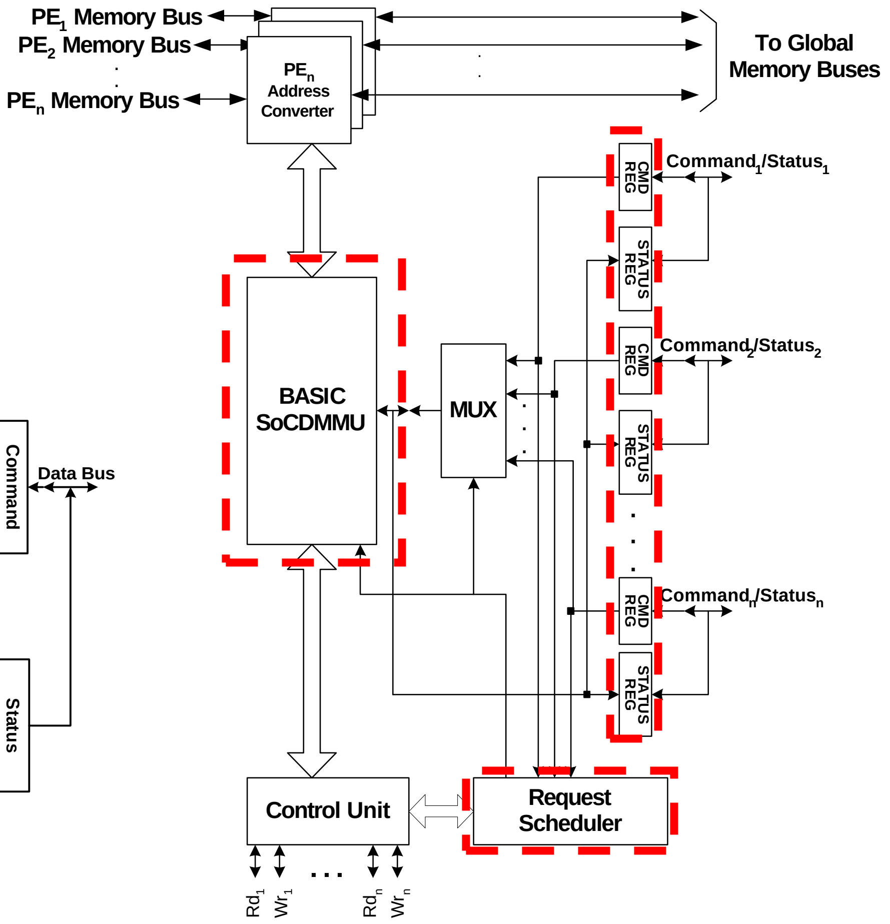
Target Architecture



The SoCDMMU

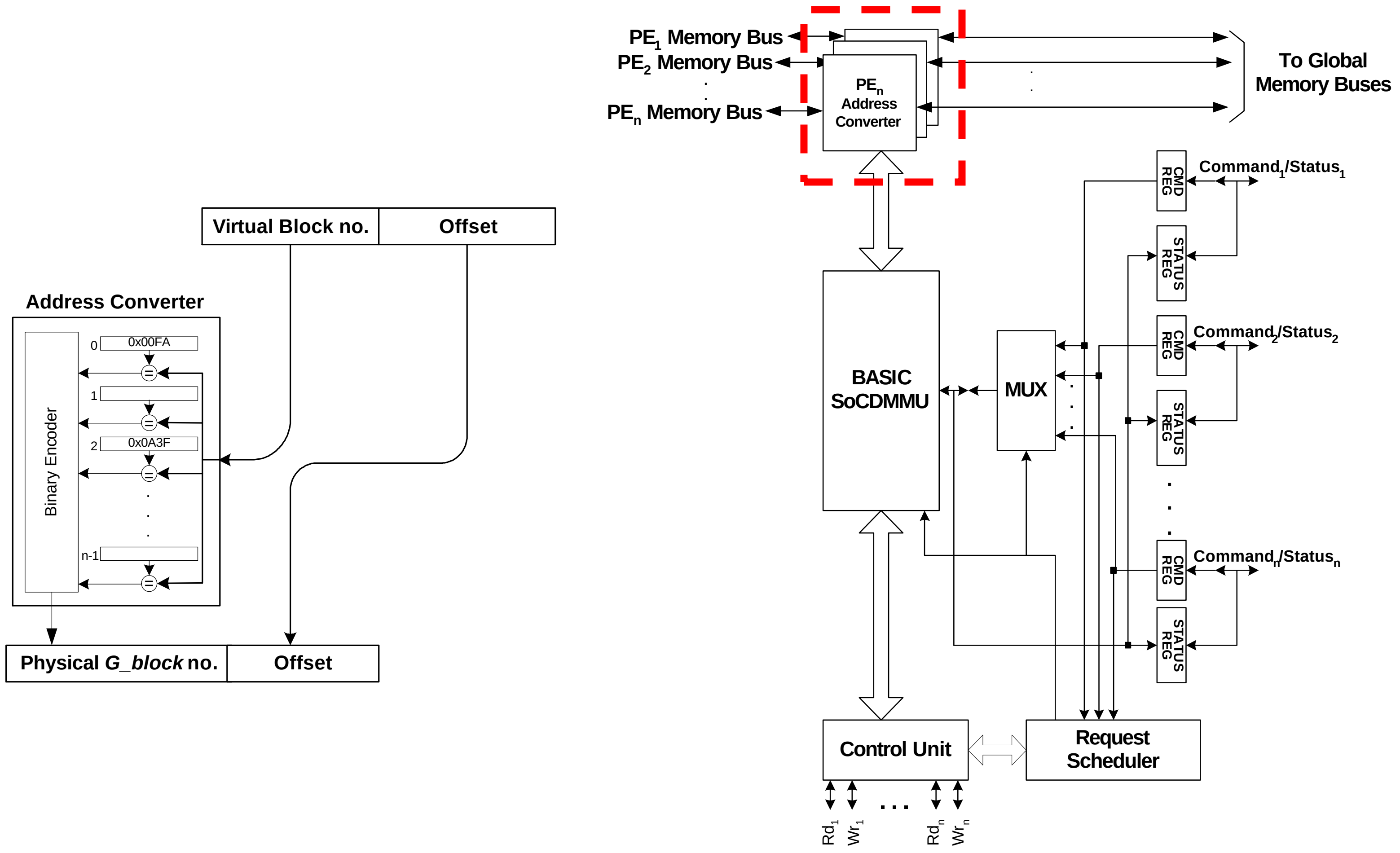


Allocation Vector
Deallocation Unit
Allocation Unit
0
MUX
Allocation Table
Command Register
Data Bus
Status Register
Control Unit
Bus Control Lines



PE1 Memory Bus
PE2 Memory Bus
PEn Memory Bus
PEn Address Converter
To Global Memory Buses
BASIC SoCDMMU
MUX
CMD REG
STATUS REG
Command1/Status1
Command2/Status2
Commandn/Statusn
Control Unit
Request Scheduler
Rd1
Wr1
Rdn
Wrn

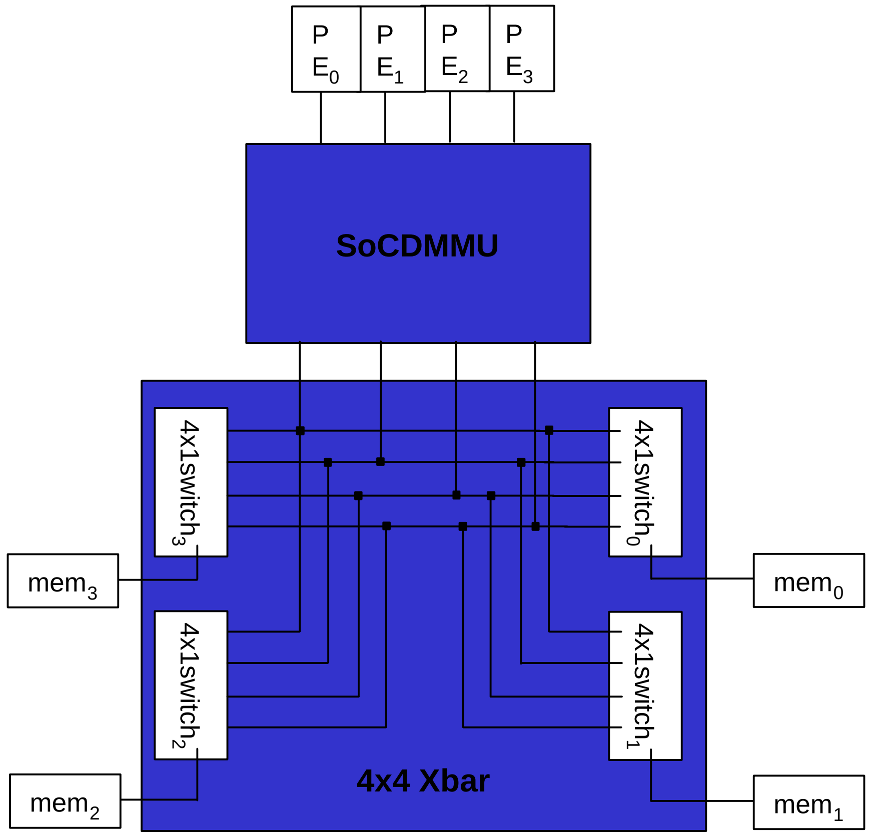
The SoCDMMU



PE$_1$ Memory Bus
PE$_2$ Memory Bus
PE$_n$ Memory Bus
PE$_n$ Address Converter
To Global Memory Buses
Virtual Block no.
Offset
Address Converter
Binary Encoder
0 0x00FA
1
2 0x0A3F
n-1
Physical *G_block* no.
Offset
BASIC SoCDMMU
MUX
CMD REG
STATUS REG
Command$_1$/Status$_1$
Command$_2$/Status$_2$
Command$_n$/Status$_n$
Control Unit
Request Scheduler
Rd$_1$ Wr$_1$ Rd$_n$ Wr$_n$

Xbar Switch

- One configuration of target architecture for 4x4 case
- All switches directly interface to SoCDMMU.
- Each switch compares physical addresses from SoCDMMU and judges if addresses belong to the address space of the attached memory block.

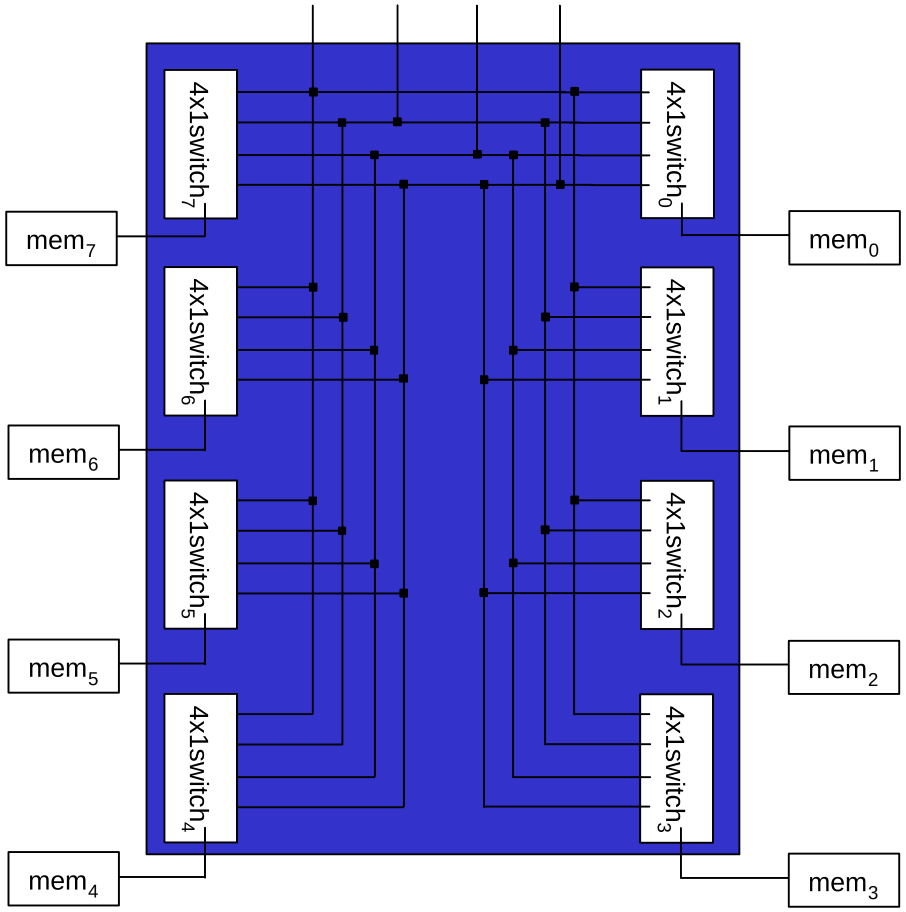


Xbar (Continued)

- An MxN switch consists of N Mx1 switches, where M = # of PEs and N = # of memory blocks.
- If an 'address' input belongs to the address space of the attached memory block, 'mem_req' inside a 4x1 switch is asserted.
- 'mem_req' signals ask an arbiter to grant a single bus attached to the corresponding memory block.
- An arbiter handles M requests from M processors and grants one request in round-robin order

Example of Xbar Configuration

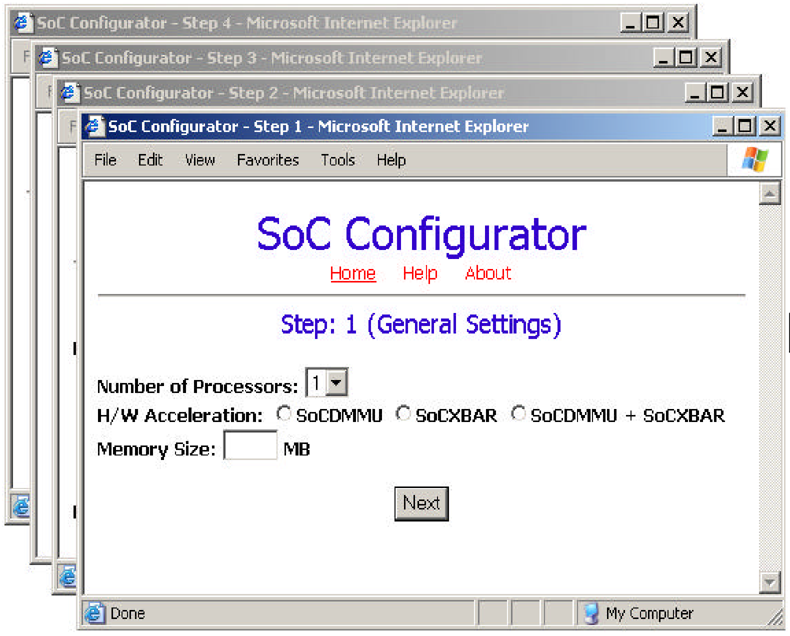
- 4x8 Xbar
  - Supports 4 PEs and 8 memory modules.
  - Connects a particular PE signals to a specific memory module by a 4x1 switch according to a physical address from an SoCDMMU.



Agenda

- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

DX-Gt Overview


SoC Configurator - Step 4 - Microsoft Internet Explorer
SoC Configurator - Step 3 - Microsoft Internet Explorer
SoC Configurator - Step 2 - Microsoft Internet Explorer
SoC Configurator - Step 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Home Help About
Next
Done
My Computer

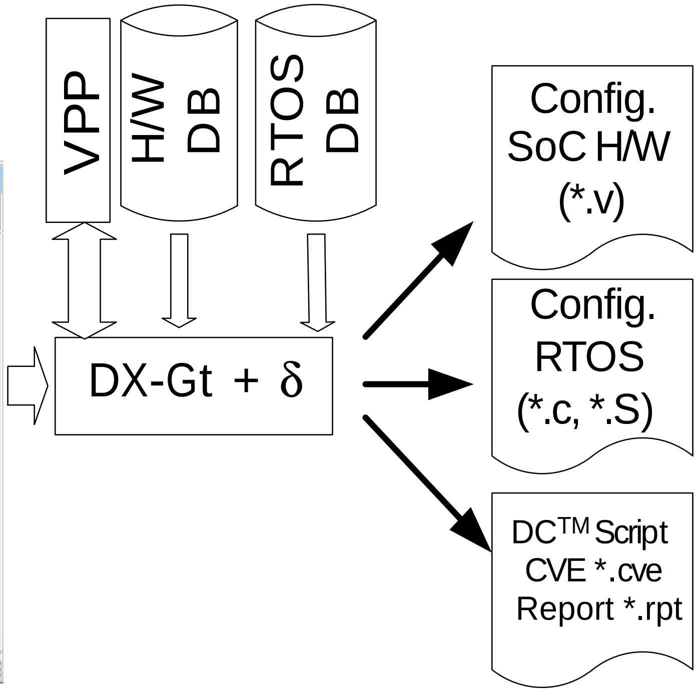
SoC Configurator

Step: 1 (General Settings)

Number of Processors: 1

H/W Acceleration: ○ SoCDMMU ○ SoCXBAR ○ SoCDMMU + SoCXBAR

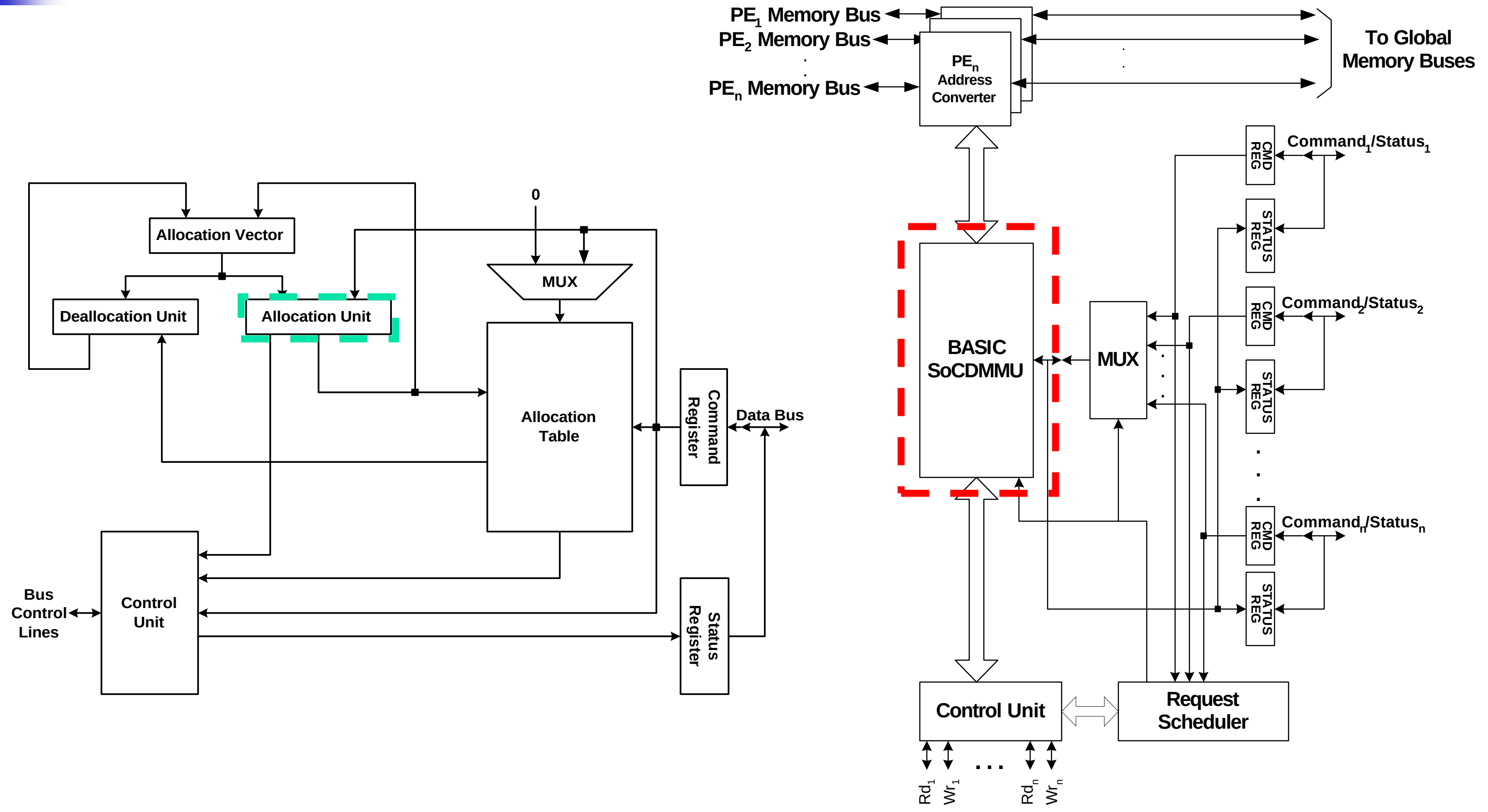
Memory Size: MB



User Specified Parameters

- System wide parameters (first two screens)
  - ✓ The number and type of PEs
  - ✓ The number and size of the global on-chip memory *G_blocks*
  - ✓ The number of memory modules & ports/module
  - ✓ The memory type
  - ✓ The choice of use of SoCDMMU, Xbar, both or none
- SoCDMMU related parameters (third screen)
  - ✓ The scheduling scheme to resolve concurrent SoCDMMU requests
  - ✓ Memory *G_blocks* initially assigned to the PEs
- Xbar related parameters (fourth screen)
  - ✓ The data bus width of each PE

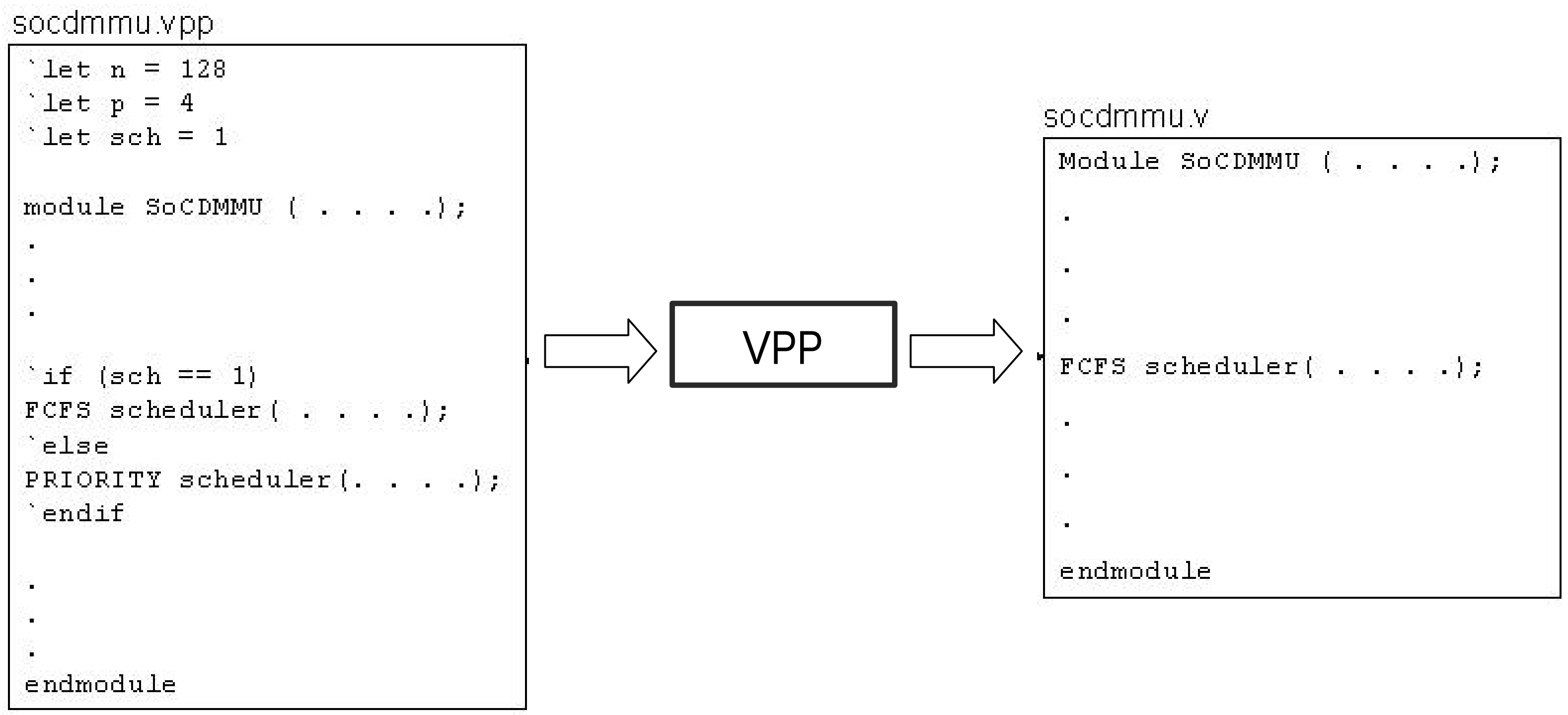
The SoCDMMU Generation



Customizing the SoCDMMU

- Verilog Language
  - `define & `ifdef
  - not enough, e.g., cannot automatically generate *n* modules
- Verilog 2000/2001
  - Generate loops (not supported by available tools)
  - not enough, e.g., cannot calculate *log2(n)*
- Verilog PreProcessor (VPP)
  - **`` `ifdef, `ifndef, `if, `let, `for, `while, `switch & `case ``**
  - **`LOG2, ROUND, CEIL, FLOOR, EVEN, ODD, MAX, MIN & ABS`**

Customizing the SoCDMMU



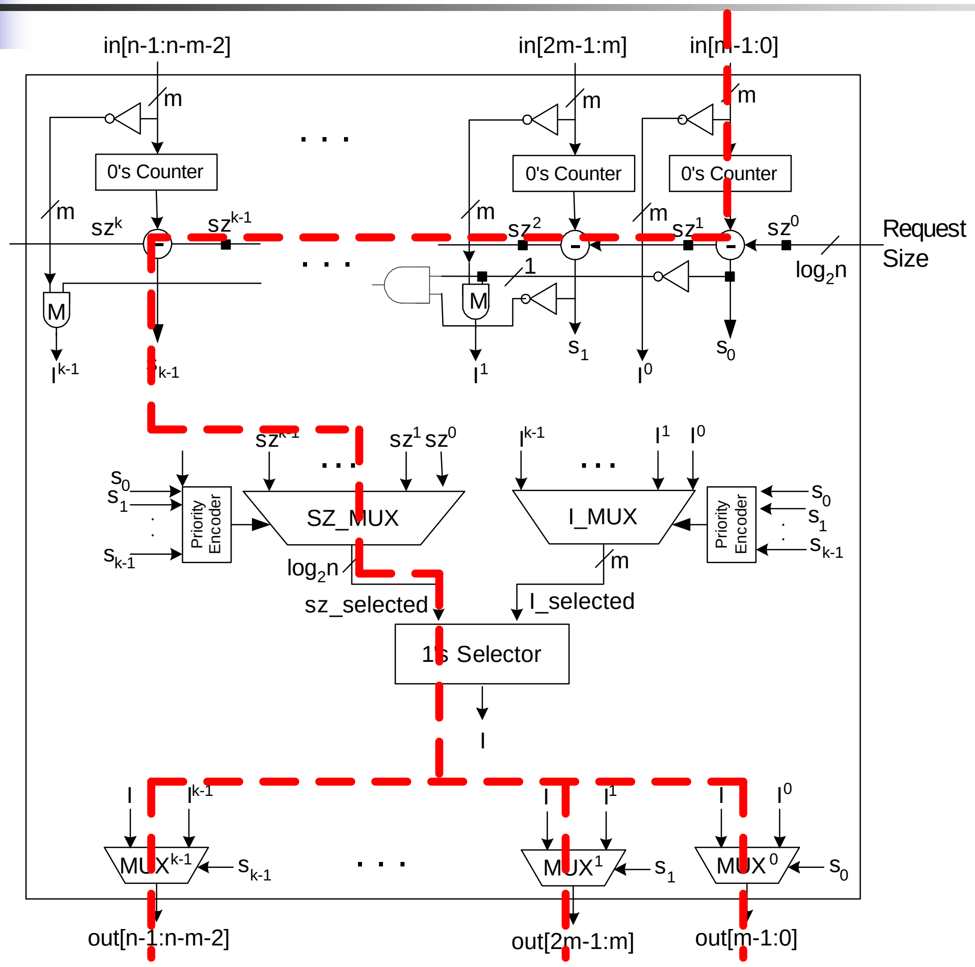
socdmmu.vpp

```
`let n = 128
`let p = 4
`let sch = 1

module SoCDMMU ( . . . .);
.
.
.

`if (sch == 1)
FCFS scheduler( . . . .);
`else
PRIORITY scheduler(. . . .);
`endif

.
.
.
endmodule
```

socdmmu.v

```
Module SoCDMMU ( . . . .);
.
.
.
FCFS scheduler( . . . .);
.
.
.
endmodule
```

Allocation Unit Optimization



0’s Counter

Almost Constant

k Subtractors

$k \times D_S$

SZ_MUX

Almost Constatnt

1’s Selector

$m \times D_1$

MUX

Almost Constant

Allocation Unit Optimization

- Delay over the critical path

  $\mathtt{Delay = C + k*D_s + m*D_1}$

- Also, we have

  $\mathtt{n = k * m}$: n is the no. of G_blocks

- This leads to

  $\mathtt{Delay = C + k*D_s + (n/k)*D_1}$

- The Delay is minimum when

  $\mathtt{k = SQR(n*D_1/D_s)}$: k is power of 2

Xbar Generation

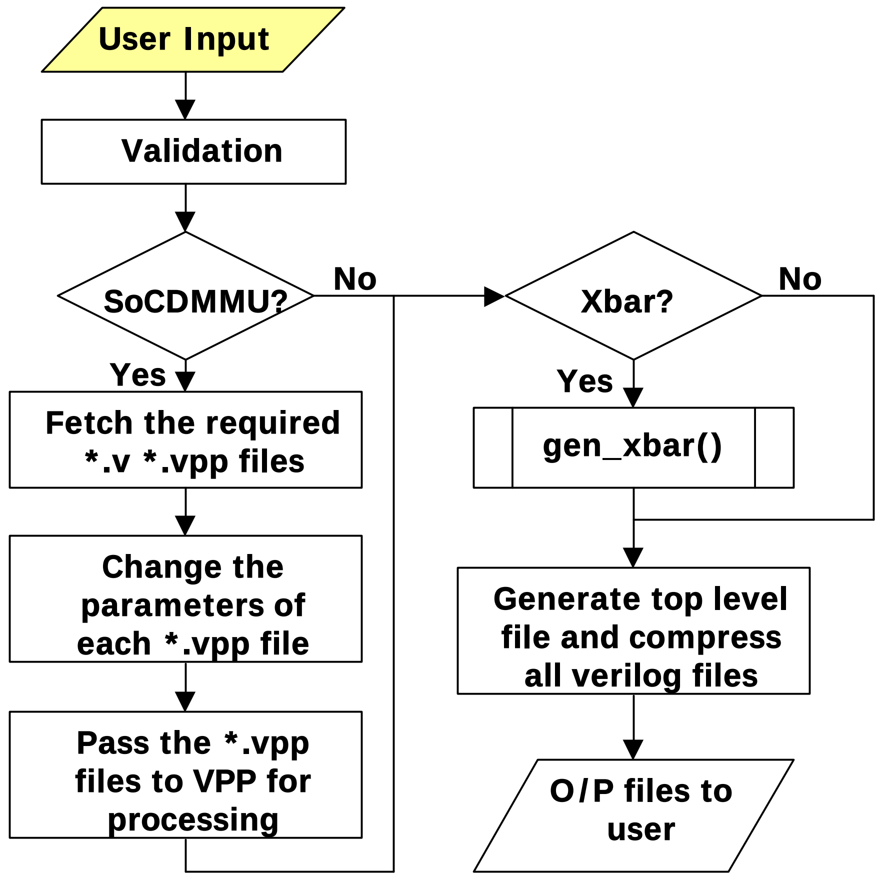
- Customized Xbar generated in Verilog at the RTL level.
  - An arbiter is generated by RAG
  - Parameterizable switch blocks are hand-coded beforehand.
  - All submodules are connected by wire names.

Flowchart of DX-Gt

User Input*

- 4 ARM9tdmi processors (N=4)
- Use SoCDMMU & Xbar
- 256 *G_blocks* (n=256)
- 4 Memory Modules (M=4)
- Initial Memory Allocations

*partial list


User Input
Validation
SoCDMMU?
No
Yes
Fetch the required *.v *.vpp files
Change the parameters of each *.vpp file
Pass the *.vpp files to VPP for processing
Xbar?
No
Yes
gen_xbar()
Generate top level file and compress all verilog files
O/P files to user

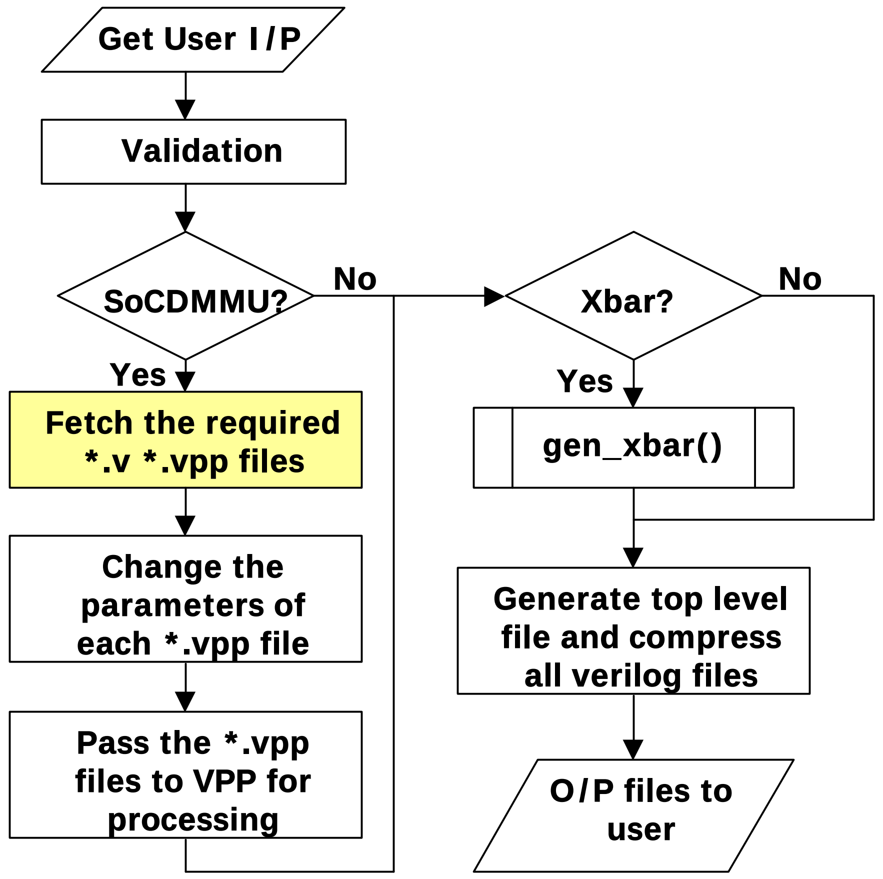
Flowchart of the DX-Gt

Fetch *.vpp files

Fetch SocDMMU bus wrapper for ARM9tdmi

Fetch the SoCDMMU *.vpp & *.v files

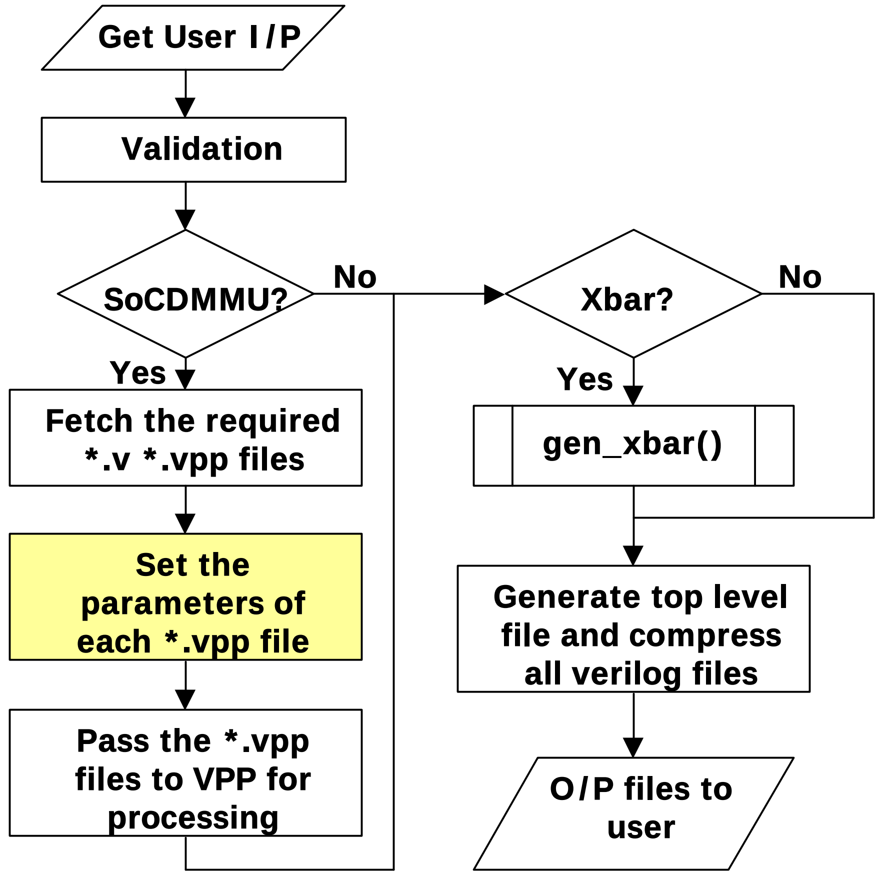
Calculate the k & m parameters that optimize the *Allocation Unit* (k=16, m=16 for TSMC 0.25)


Get User I/P
Validation
SoCDMMU?
No
Yes
Fetch the required *.v *.vpp files
Change the parameters of each *.vpp file
Pass the *.vpp files to VPP for processing
Xbar?
No
Yes
gen_xbar()
Generate top level file and compress all verilog files
O/P files to user

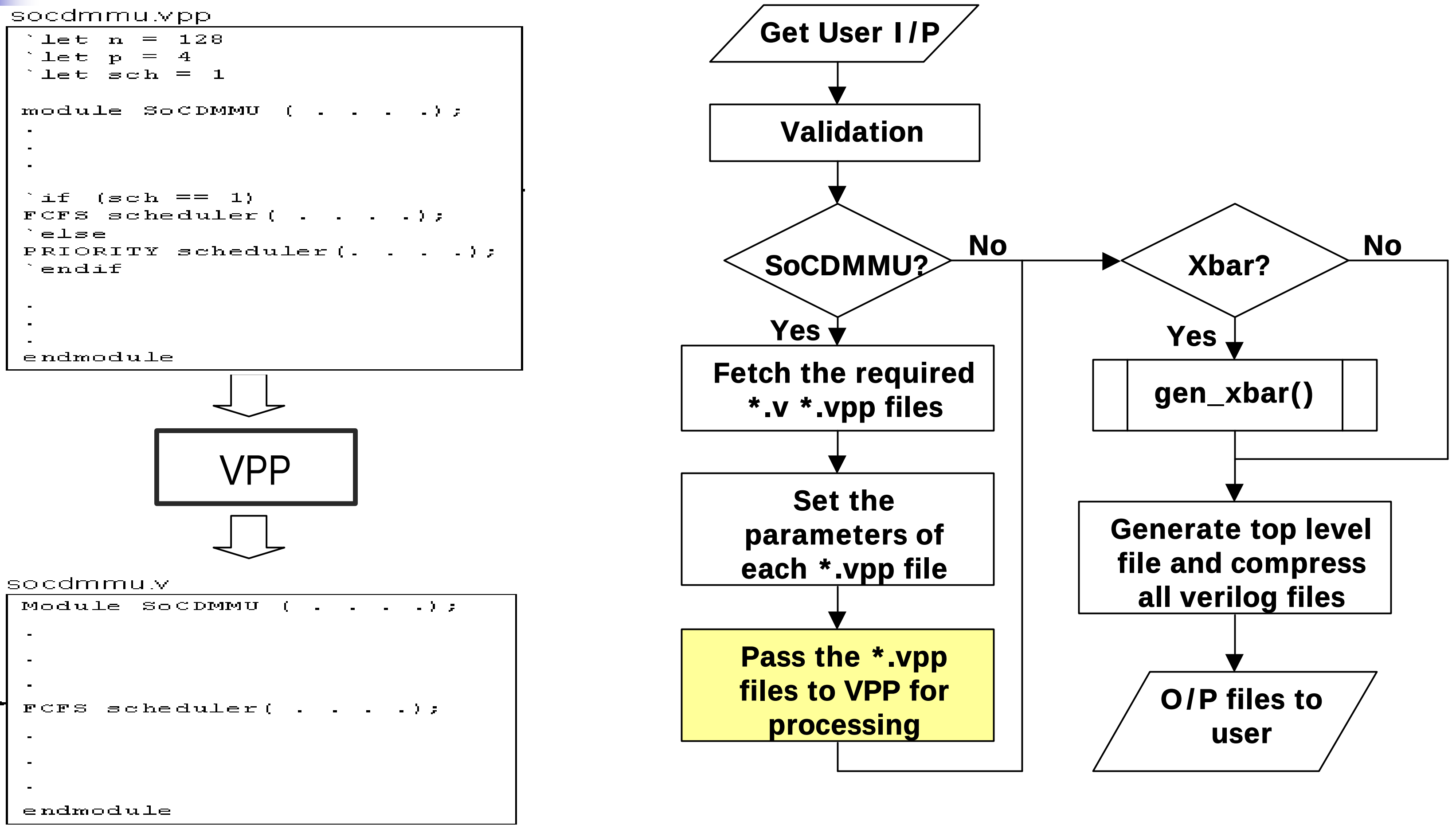
Flowchart of the DX-Gt

AllocationUnit.vpp

```
`let n = 256     //No. of G_blocks
`let p = 4       //No. of processors
`let lbsz = 13   //log2(G_block size)
`let VLOGR = 1   //Verilogger Friendly
`let k = 16      //# of segments
`let m = 16      //Segment width

`let ate = LOG2(p) + LOG2(n) + 2
`let v = 32 - lbsz
`let ln = LOG2 (n)
`let lp = LOG2 (p)
`let ln1 = ln - 1
`let lp1 = lp - 1
`let ate1 = ate - 1
`let n1 = n - 1
`let p1 = p - 1
`let v1 = v - 1

//Allocation Unit Specific
`let k1 = k-1
`let m1 = m-1
`let lk = LOG2(k)
`let lm = LOG2(m)
.
.
.
```


Get User I/P
Validation
SoCDMMU?
No
Yes
Fetch the required *.v *.vpp files
Set the parameters of each *.vpp file
Pass the *.vpp files to VPP for processing
Xbar?
No
Yes
gen_xbar()
Generate top level file and compress all verilog files
O/P files to user

Flowchart of the DX-Gt



socdmmu.vpp

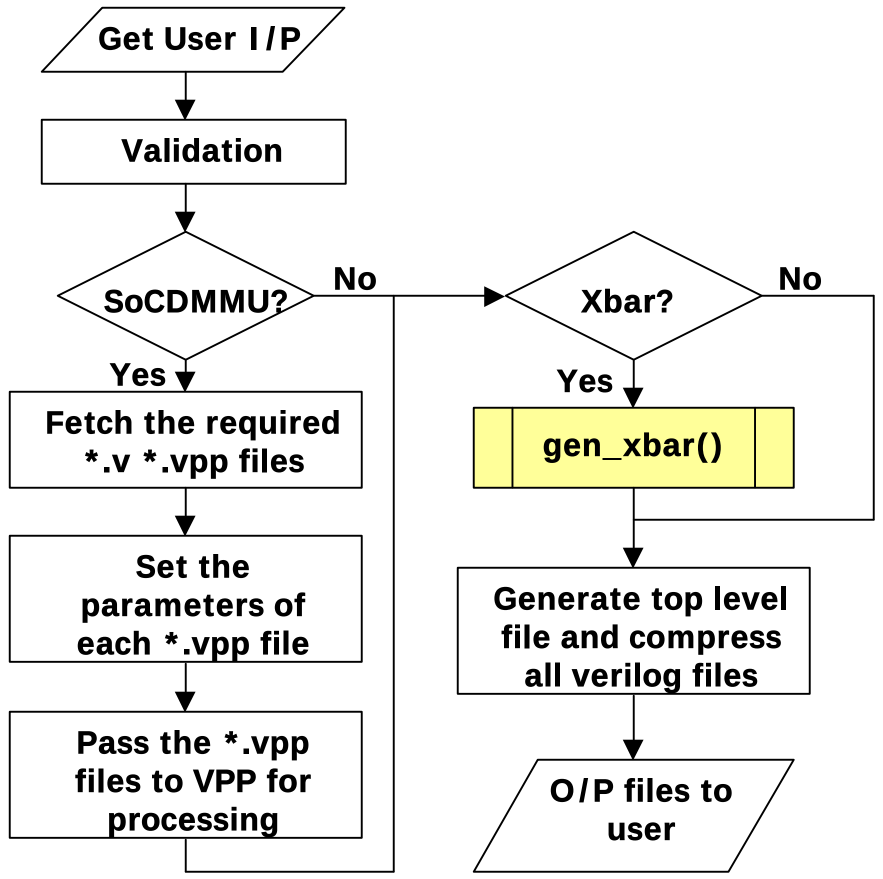
```
`let n = 128
`let p = 4
`let sch = 1

module SoCDMMU ( . . . .);
.
.
.

`if (sch == 1)
FCFS scheduler( . . . .);
`else
PRIORITY scheduler(. . . .);
`endif

.
.
.
endmodule
```

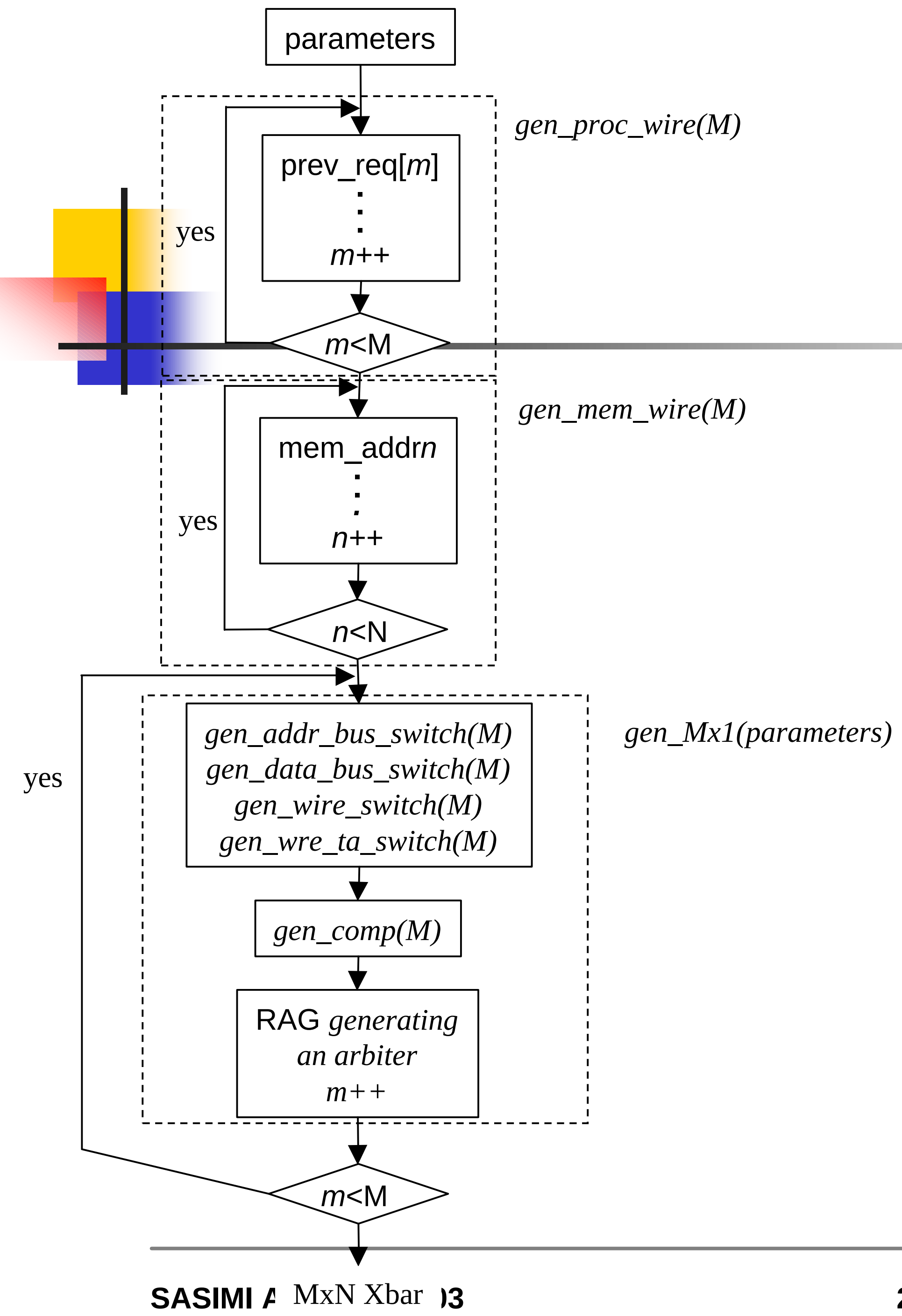
socdmmu.v

```
Module SoCDMMU ( . . . .);
.
.
.
FCFS scheduler( . . . .);
.
.
.
endmodule
```

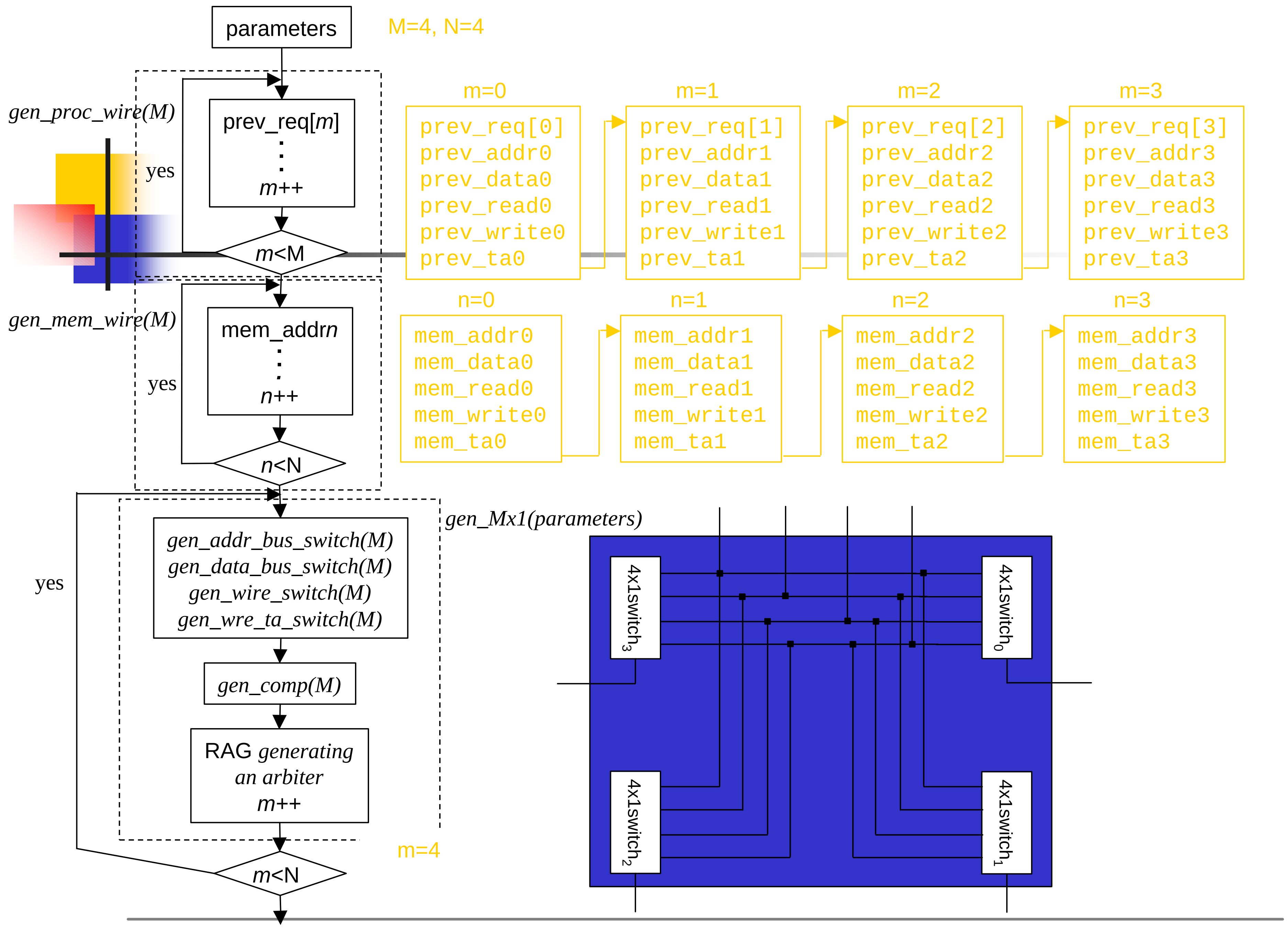
VPP
Get User I/P
Validation
SoCDMMU?
No
Yes
Fetch the required *.v *.vpp files
Set the parameters of each *.vpp file
Pass the *.vpp files to VPP for processing
Xbar?
No
Yes
gen_xbar()
Generate top level file and compress all verilog files
O/P files to user

Flowchart of the DX-Gt


Get User I/P
Validation
SoCDMMU?
No
Yes
Fetch the required *.v *.vpp files
Set the parameters of each *.vpp file
Pass the *.vpp files to VPP for processing
Xbar?
No
Yes
gen_xbar()
Generate top level file and compress all verilog files
O/P files to user



- prev_ indicates that signals come from SoCDMMU.
- mem_ indicates that signals to memory blocks.


parameters
M=4, N=4
gen_proc_wire(M)
prev_req[m]
m++
yes
m<M
m=0
prev_req[0]
prev_addr0
prev_data0
prev_read0
prev_write0
prev_ta0
m=1
prev_req[1]
prev_addr1
prev_data1
prev_read1
prev_write1
prev_ta1
m=2
prev_req[2]
prev_addr2
prev_data2
prev_read2
prev_write2
prev_ta2
m=3
prev_req[3]
prev_addr3
prev_data3
prev_read3
prev_write3
prev_ta3
gen_mem_wire(M)
mem_addrn
n++
yes
n<N
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
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
m++
m=4
m<N
gen_Mx1(parameters)
4x1switch$_3$
4x1switch$_0$
4x1switch$_2$
4x1switch$_1$
4x4 Xbar

Customizing the Xbar

- Verilog Language
  - \`define & \`ifdef
  - not enough, e.g., cannot automatically generate *n* modules
- Verilog 2000/2001
  - Generate loops (not supported by available tools)
  - not enough, e.g., cannot calculate *log2(n)*
- C Code generates custom Verilog directly

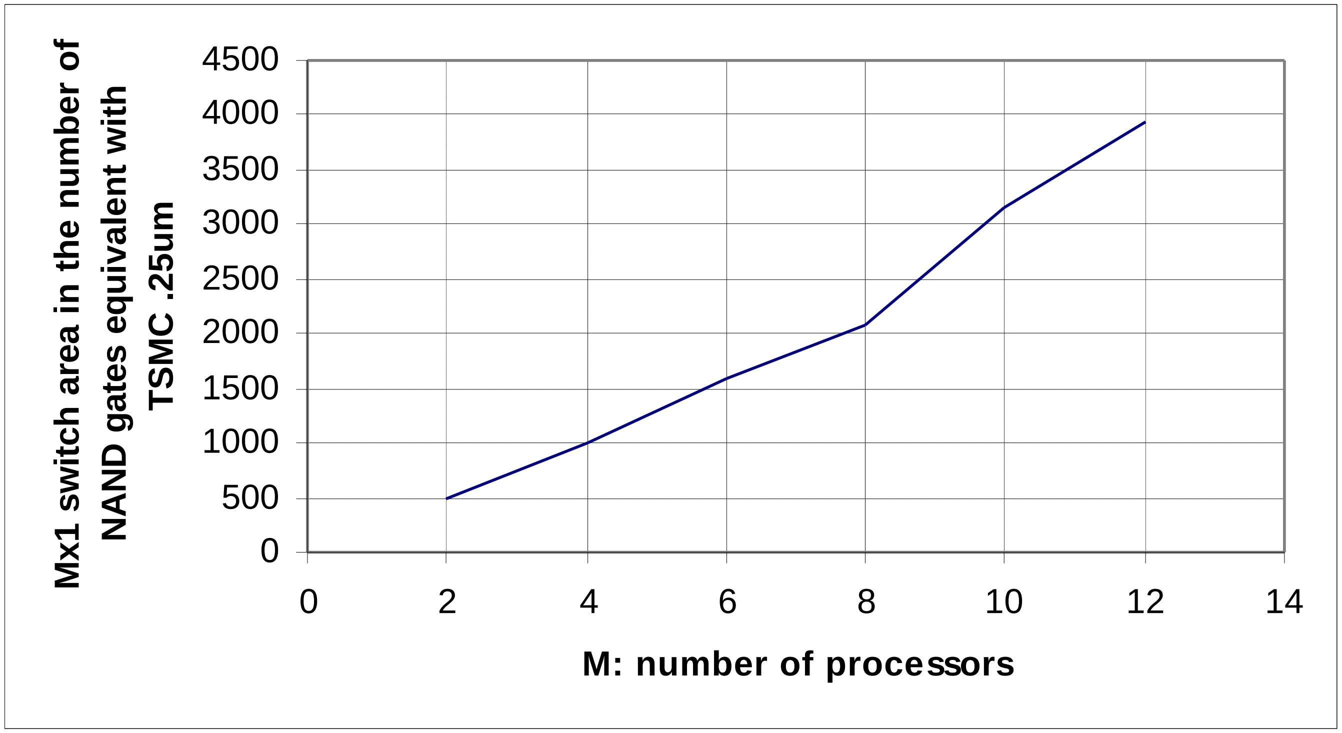
Agenda

- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

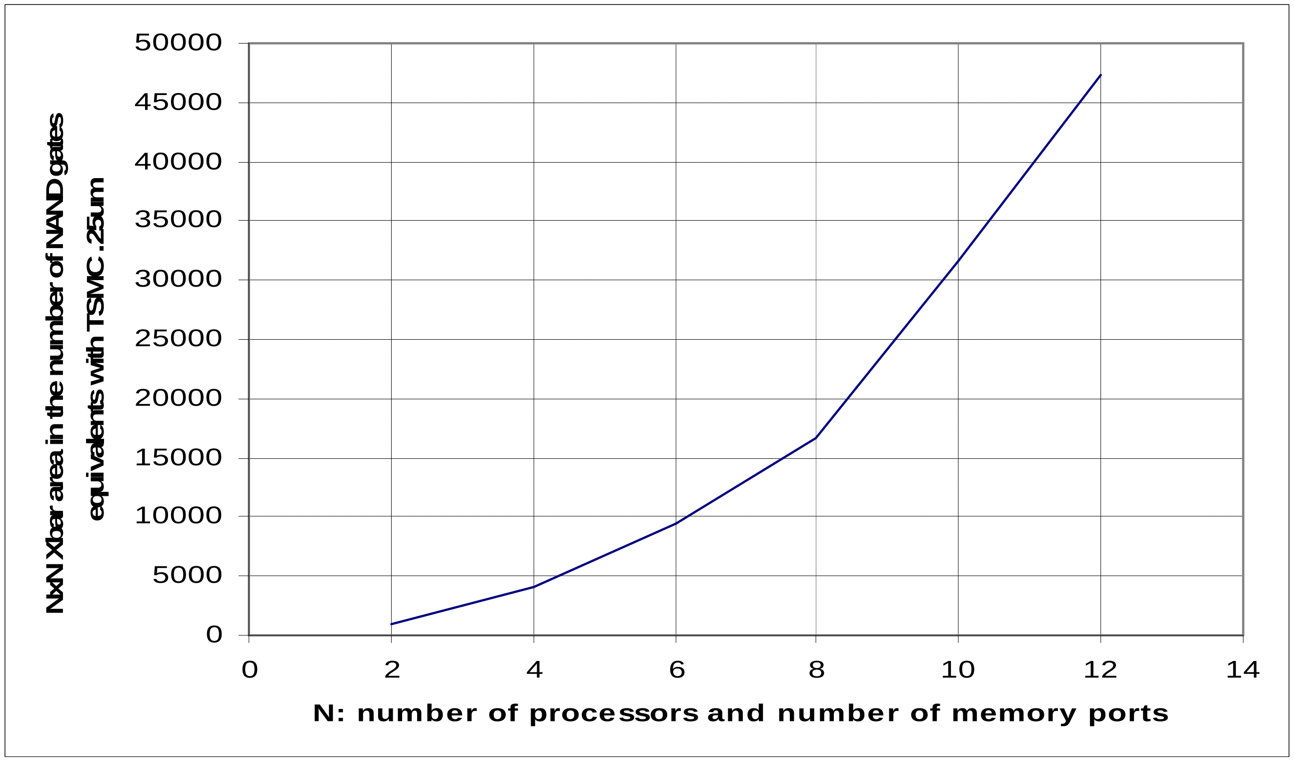
Synthesis Results

- We synthesized different configurations of the SoCDMMU and the Xbar.
- We use the Synopsys Design Compiler ? with a 0.25µm TSMC technology library from LEDA Systems

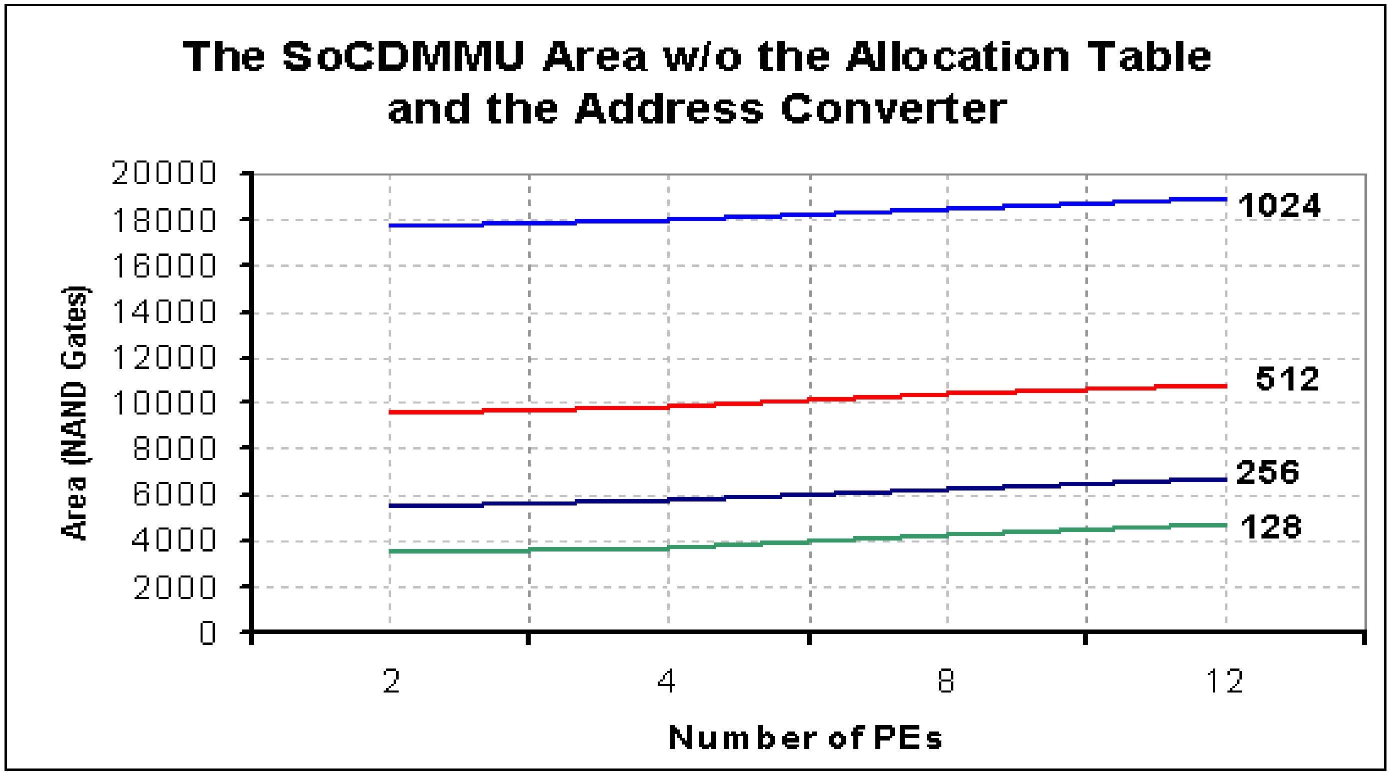
Mx1 Switch Area



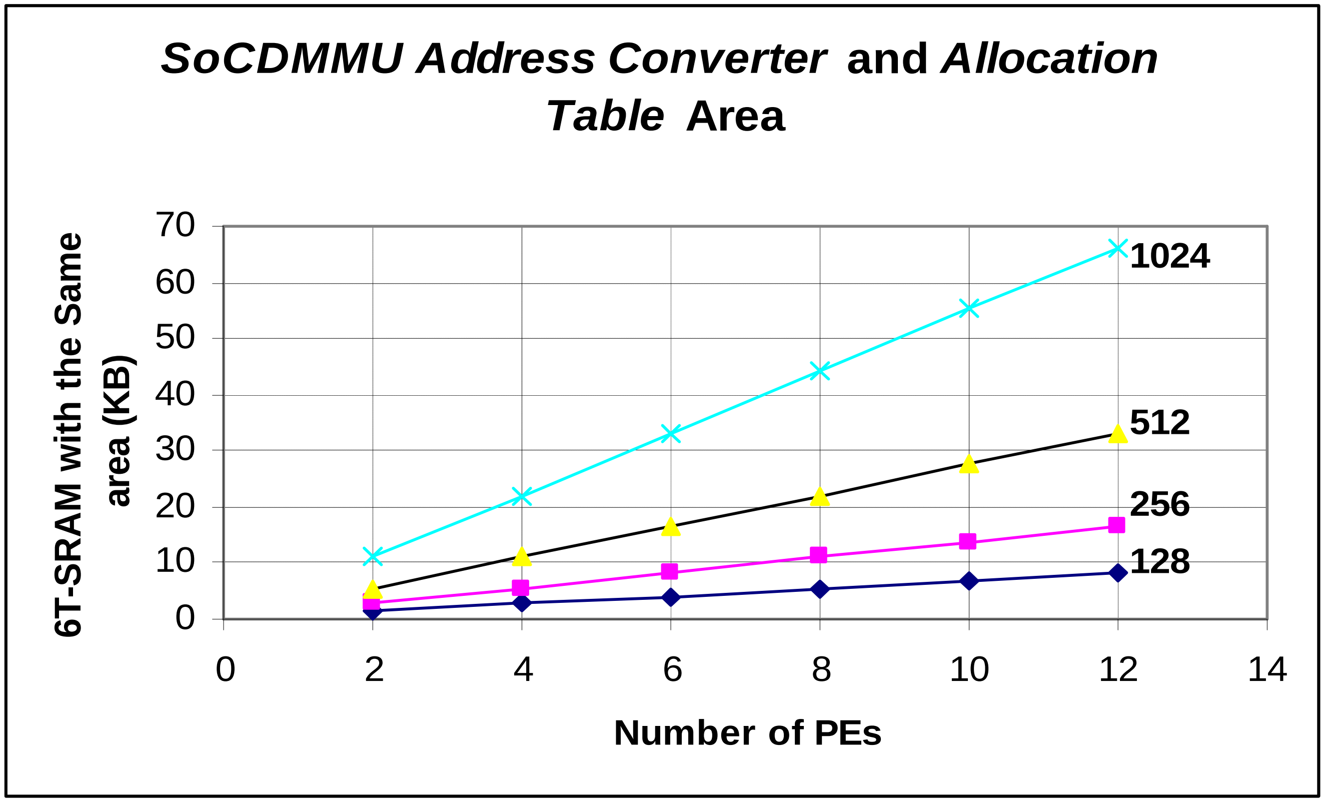
MxN Xbar Area



SoCDMMU Area (w/o memory)



SoCDMMU Area (Memory)



Agenda

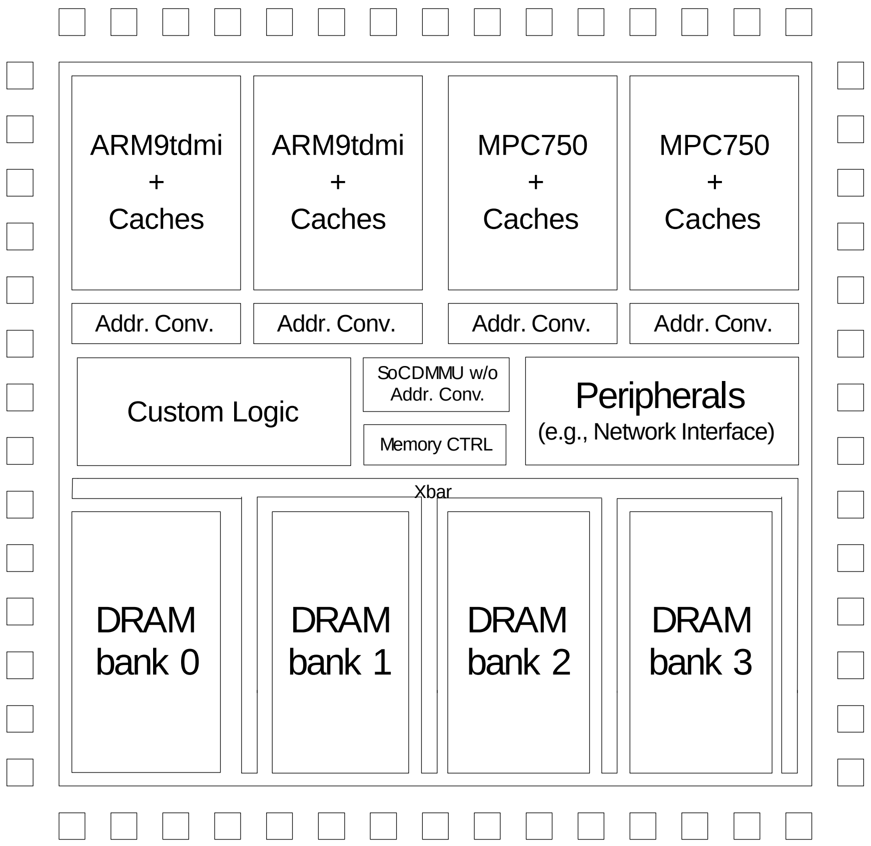
- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

SoC Floorplan

Tools/Information Used to Floorplan:

- ARM website (ARM core area)
- Synopsys Design Compiler
- Cadence Silicon Ensemble

- ARM9TDMI Core: 112k transistors
- L1 \$ (128KB: 64KB I\$ + 64KB D\$): ~6.5M* transistors
- SoCDMMU (w/o the memory elements -- Allocation Table and Address Converters): ~28k transistors.
- Allocation Table: ~168k transistors
- Address Converter: ~320k** transistors
- L2 (Global Memory)=~16M * 8 = ~128M transistors
- For TSMC 0.25u
  - SoCDMMU w/o memory elements: 1.43mm^2
  - Xbar : 0.23mm^2



* Using dual-port 6T SRAM Cells.

** A custom physical design would a much smaller number.

Agenda

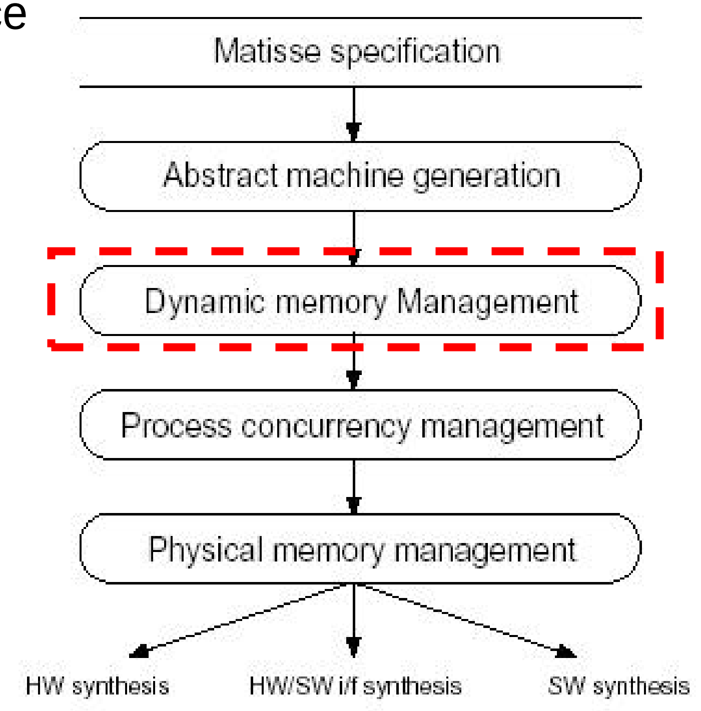
- Introduction & Motivation
- Target Architecture
- The DX-Gt
- Synthesis Results
- SoC Floorplan
- Conclusions

Conclusion

- DX-Gt is a System-on-a-Chip IP generation tool that enables an SoC designer to design a multiprocessor SoC and configure its memory and bus subsystems to meet the design constraints with ease.

Previous Work in Custom Memory Management: Matisse

- Virtual Memory Management Search Space
  - Keeping track of free blocks
  - Choosing a free block
  - Freeing allocated blocks
  - Merging Free Blocks
- Physical Memory Optimization
  - Basic Groups
  - Basic Groups memory assignment
  - Address Optimization



Matisse specification
Abstract machine generation
Dynamic memory Management
Process concurrency management
Physical memory management
HW synthesis
HW/SW i/f synthesis
SW synthesis

Matisse design Flow

Previous Work: Matisse vs. SoCDMMU

- Matisse
  - DMM Synthesis (VM & Physical Memory)
  - Application Specific (suitable for special-purpose systems, e.g., an ATM switch)
- SoCDMMU
  - Run-Time DMM
  - General Purpose (not tied to any application or configuration)