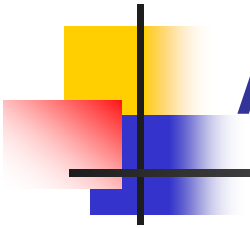
Dynamic Memory Management for Real-Time Multiprocessor System-on-a-Chip

Mohamed A. Shalan

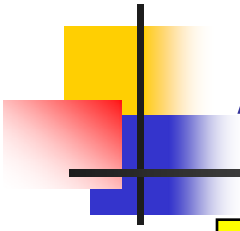
Dissertation Advisor
Vincent J. Mooney III

School of Electrical and Computer Engineering



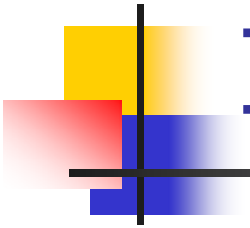
Agenda

- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments



Agenda

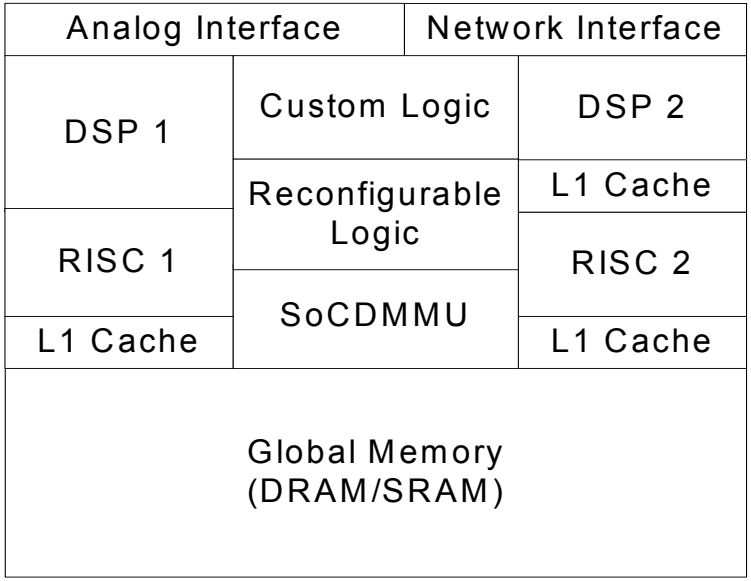
- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments



Introduction

- In few years, we will have chips with one-billion transistors
- Chips will no longer be a stand-alone system components but "Silicon boards"
- A typical Chip will consist of multiple PEs of various types, large global on-chip memory, analog components, and custom logic (e.g., network interface)

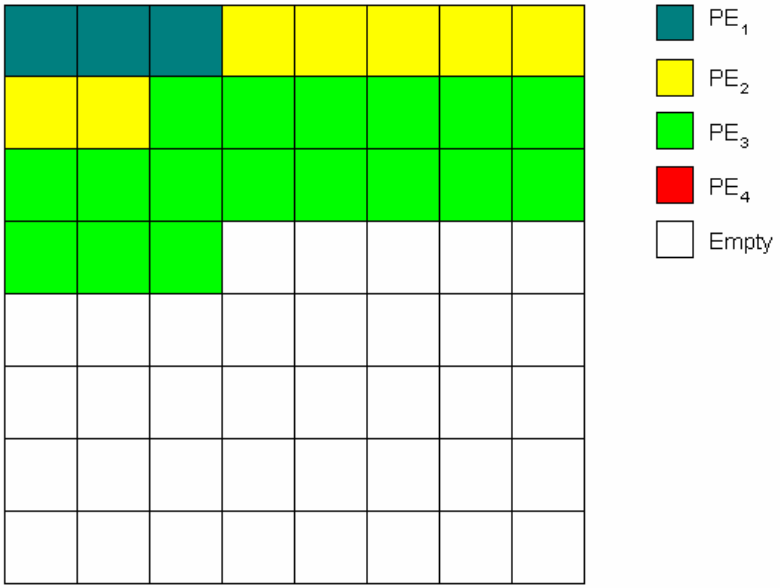
System-on-a-Chip (SoC)

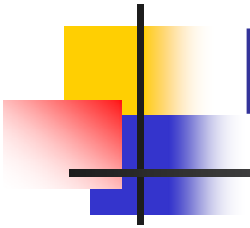


- This architecture is suitable for embedded multimedia applications, which require great processing power and large volume data management

SoC

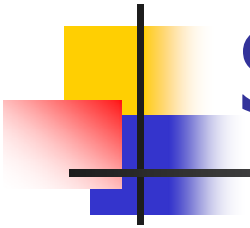
- The existence of *global* on-chip memory, arises the need for an efficient way to dynamically allocate it among the PEs





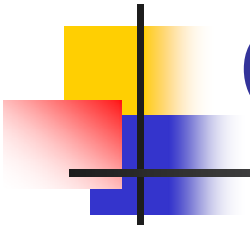
Problem

- How to deal with the allocation of the large global on-chip memory between the PEs in a dynamic yet deterministic way?



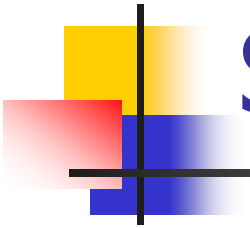
Solution 1

- Custom Memory Configuration (Static)
  - Hardware/Software co-synthesis with memory hierarchies [Wayne Wolf]
  - Matisse [IMEC]
  - Memory synthesis for telecom applications [WUYTACK et Al.], [YKMAN et al.]



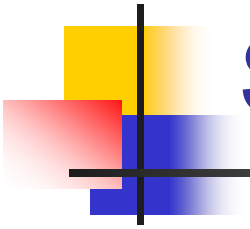
Custom Memory Configuration

- Pros:
  - Easy
  - Deterministic
- Cons:
  - Inefficient memory utilization
  - System modification after implementation is very difficult if not impossible



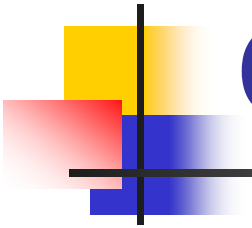
Solution 2

- Shared memory multiprocessor (Dynamic)
  - Using conventional software memory Allocation/Deallocation techniques (e.g., Sequential Fits, Buddy Systems, etc.)
  - Sharing one heap (using locks)
  - Multiple heaps (one per processor)



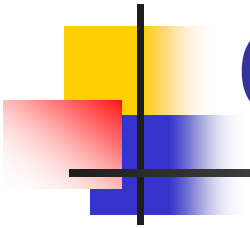
Shared memory multiprocessor

- Pros
  - Flexible
  - Efficient memory utilization
- Cons
  - Worst case execution time is very high and usually not deterministic



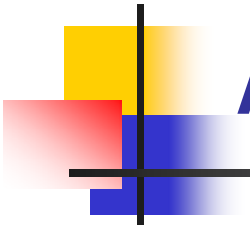
Our Solution

- We introduce a new memory management hierarchy, Two-Level Memory Management, for a multiprocessor SoC
- Two-Level Memory Management combines the best of dynamic memory management techniques (flexibility and efficiency) with the best of static memory allocation techniques (determinism).



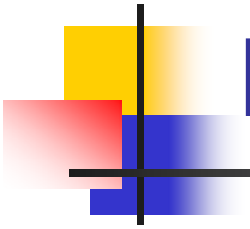
Our Solution (2)

- In Two-Level Memory Management, large on-chip memory is managed between the on-chip processors (Level Two)
- Memory assigned to any processor is managed by the operating system running on that particular processor (Level One)
- To manage Level Two, we present the System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)



Agenda

- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments



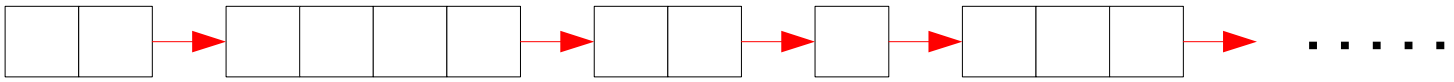
Dynamic Memory Management

- Automatic
  - Automatically recycles memory that a program will not use again
  - Either as a part of the language or as an extension
- Manual
  - The programmer has direct control over when memory is allocated and when memory may be de-allocated (e.g., by using *malloc()* & *free()*)

Memory Allocation

Software Techniques

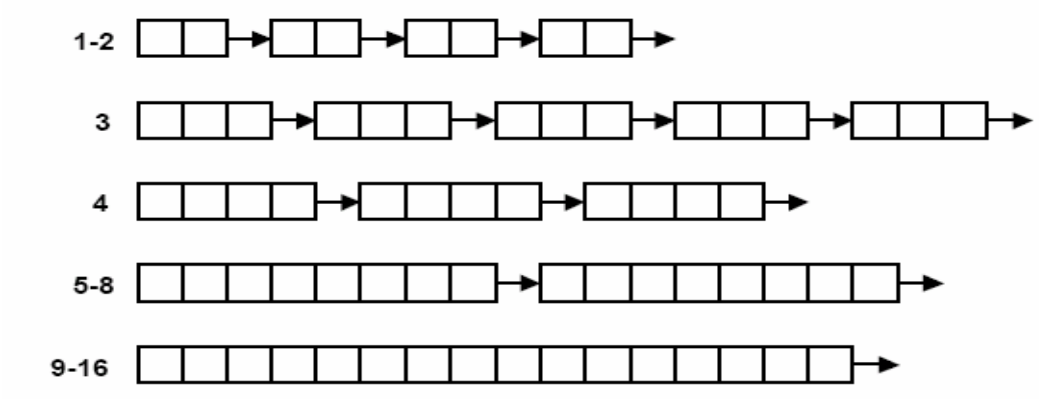
- **Sequential Fits**



  - First Fit,
  - Next Fit,
  - Best Fit or
  - Worst Fit

Memory Allocation
Software Techniques

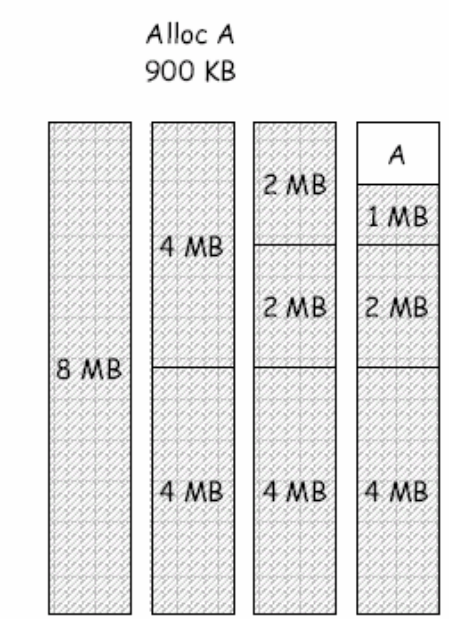
- **Segregated Free Lists**



  - Simple Segregated Storage
  - Segregated Fit

Memory Allocation
Software Techniques

- **Buddy System**



- **Bitmapped Fits**

Memory Allocation
Hardware Techniques

- **Knowlton***

  Binary buddy allocator that can allocate memory blocks whose sizes are a power of 2

- **Puttkamer** *

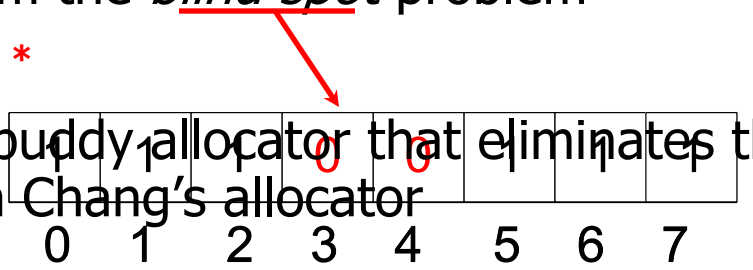
  Hardware buddy allocator (using Shift Register)

- **Chang and Gehringer** *

  Modified hardware-based binary buddy system that suffers from the *blind spot* problem

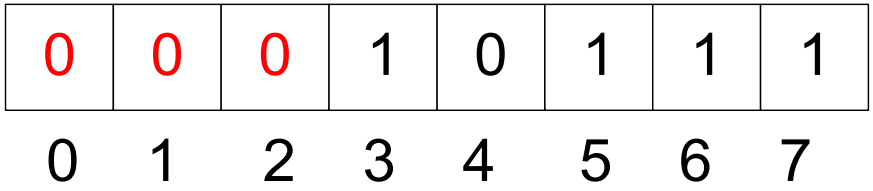
- **Cam et al.** *

  Hardware buddy allocator that eliminates the *blind spot* problem in Chang's allocator



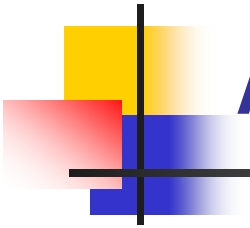
* References are available in the thesis

Memory Allocation
Hardware Techniques



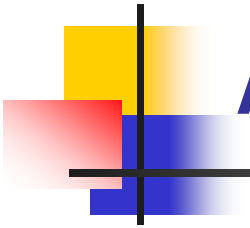
- Request size is 3
- It searches for 4

  [3 rounded to the nearest power of 2]



Agenda

- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments

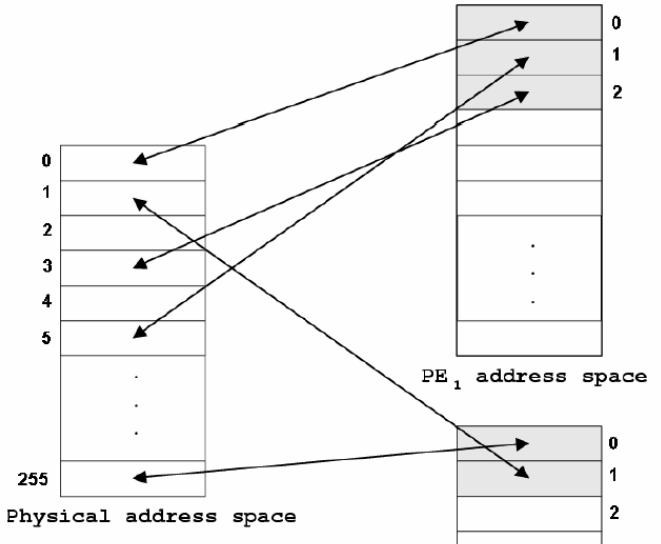


Assumptions

- The global memory is divided into a fixed number of equally sized blocks (e.g., 16KB)
- The global memory allocation done by the SoCDMMU will be referred to as *G_allocation*
- The global memory de-allocation done by the SoCDMMU will be referred to as *G_deallocation*
- The PE can *G_allocate* one or more than one block.
- Different PEs can issue the *G_allocation/ G_de-allocation* commands simultaneously

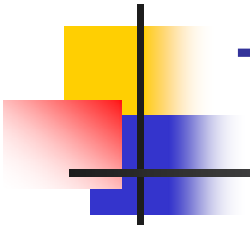
Assumptions

- Each memory block has one physical address and one or more virtual addresses. The block virtual address may differ from one PE to another
- The block virtual address will be referred to as PE-address



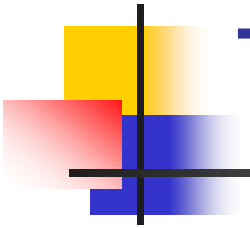
Physical Memory Block No.	Virtual Block No.
0	0
5	1
3	2

Physical Memory Block No.	Virtual Block No.
255	0
1	1



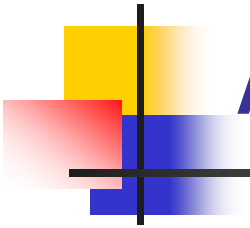
Two-Level Memory Management

- The SoCDMMU manages the memory between the PEs
- The OS (or custom software) on each PE manages the memory between the processes that run on that PE
- The process requests the memory allocation from the OS or custom software. If there in not enough memory, the OS requests memory allocation from the SoCDMMU



Types of Memory Allocation

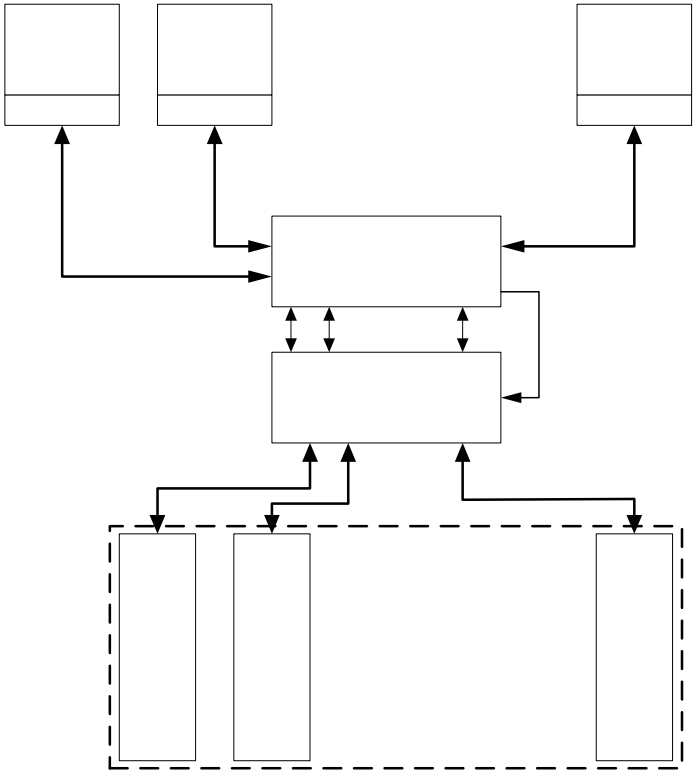
- **Exclusive**
  - Only the owner can access it. No other PE can access it
- **Read/Write**
  - The owner can read/write to it. Other PEs can read from it if they *G_allocated* it as read only
- **Read Only**
  - The PE *G_allocates* the memory for read only. Other PE *G_allocated* it as Read/Write

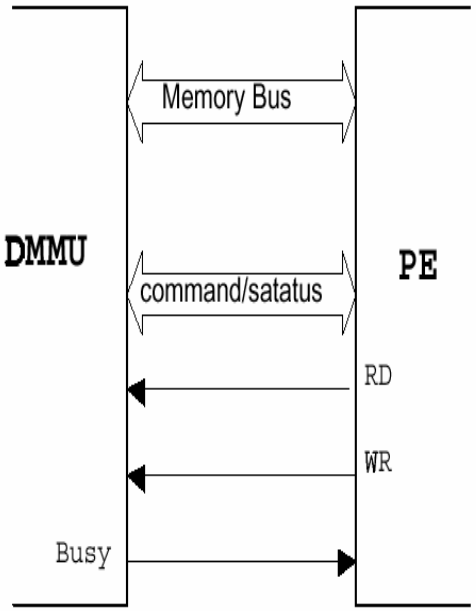


Agenda

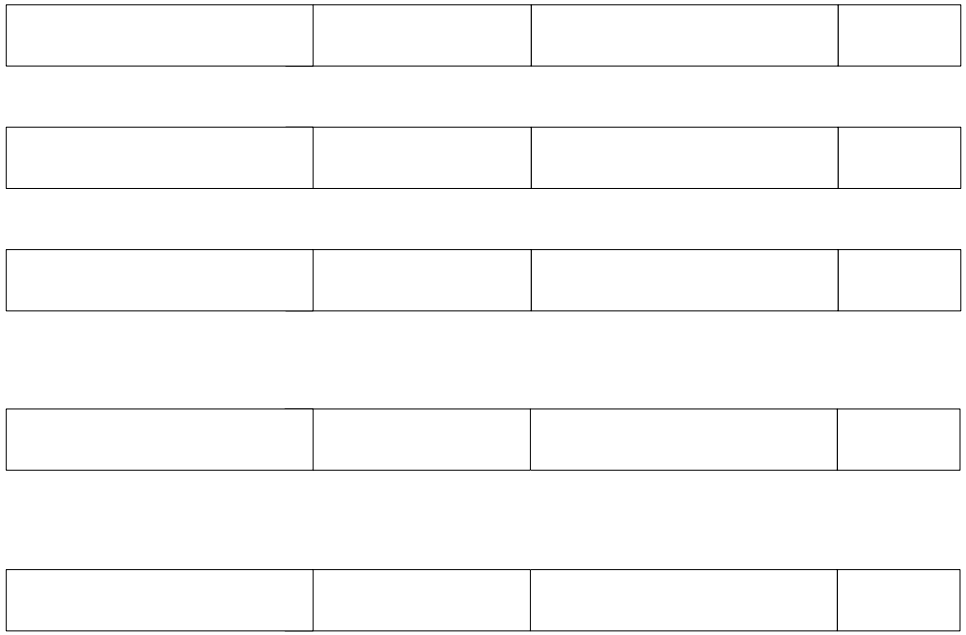
- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments

PE-SoCDMMU Interface

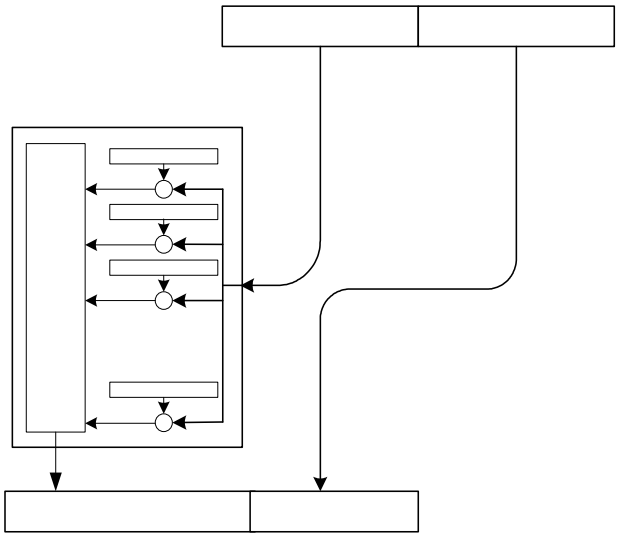




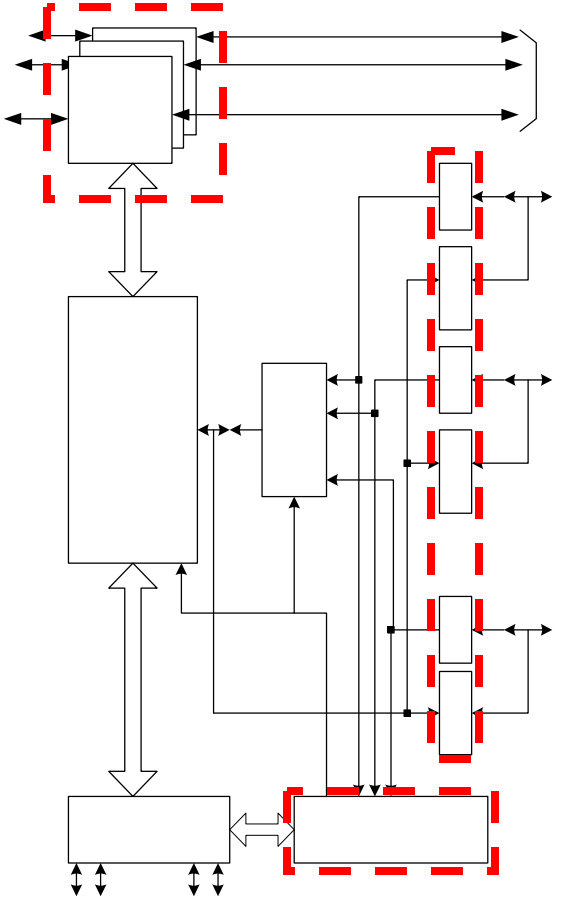
SoCDMMU Commands



The SoCDMMU Hardware

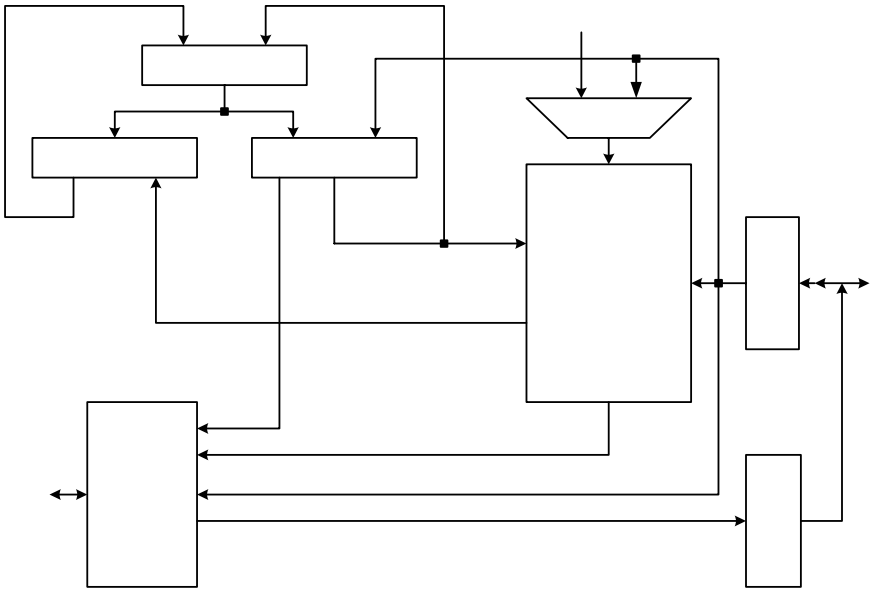


Address Converter

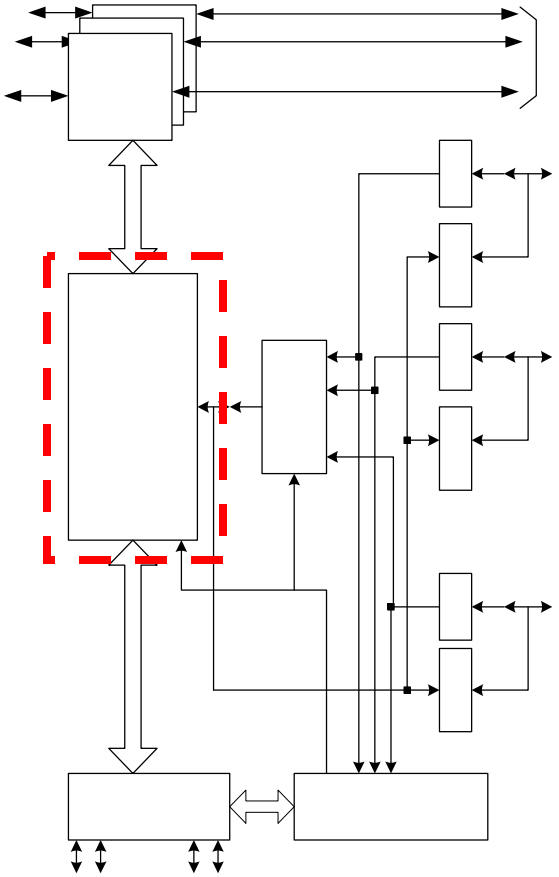


The SoCDMMU Hardware

The Basic SoCDMMU

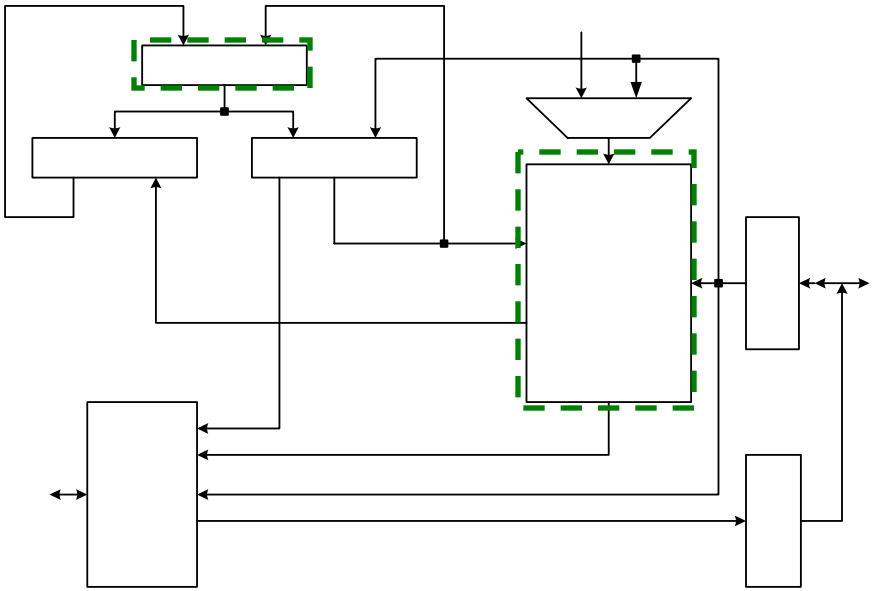


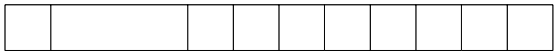
Basic SoCDMMU

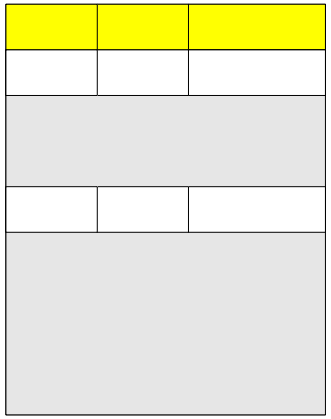


The SoCDMMU Hardware

The Basic SoCDMMU



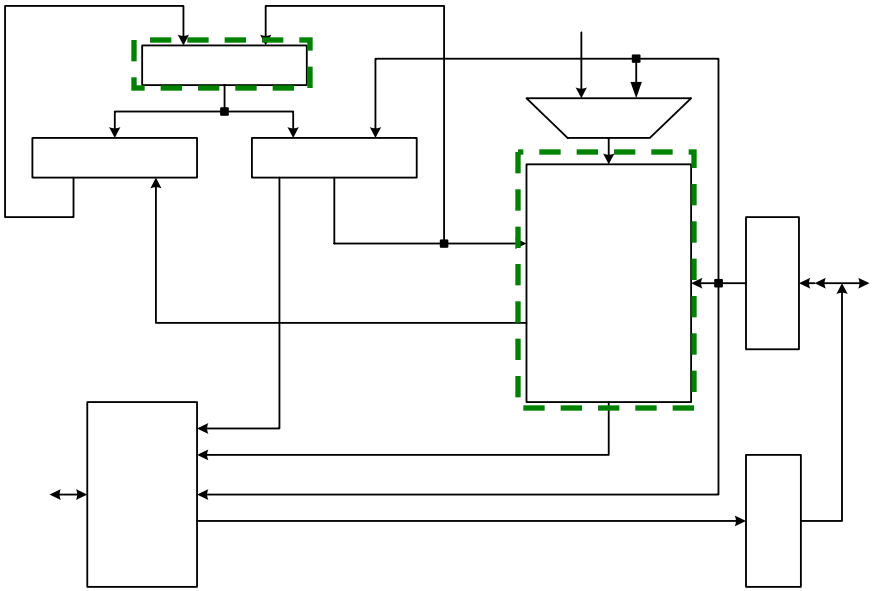




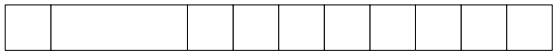
Basic SoCDMMU

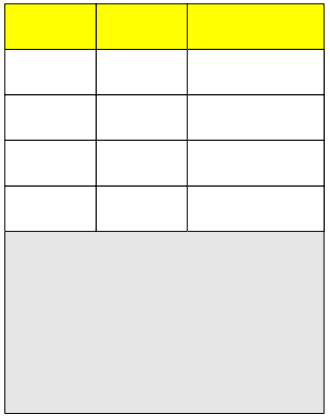
The SoCDMMU Hardware

The Basic SoCDMMU



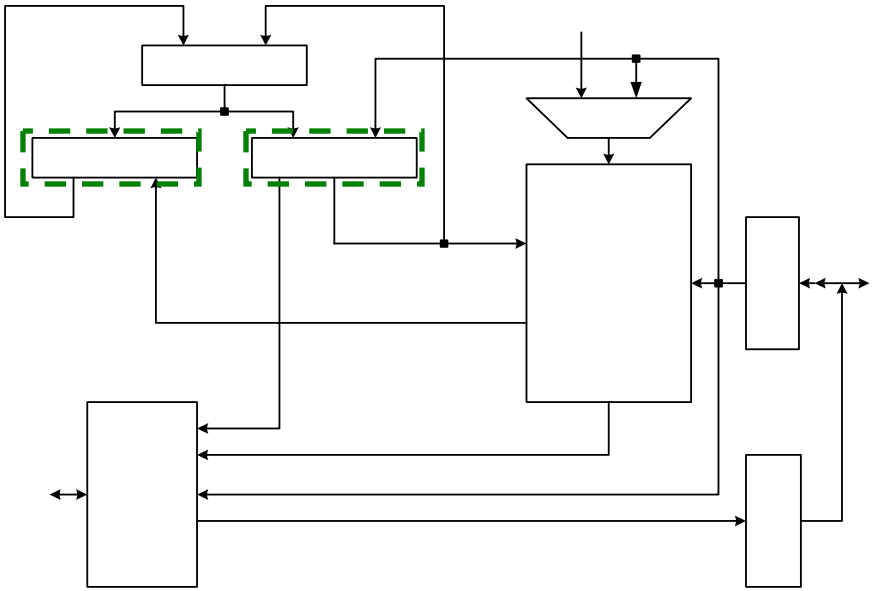
Basic SoCDMMU



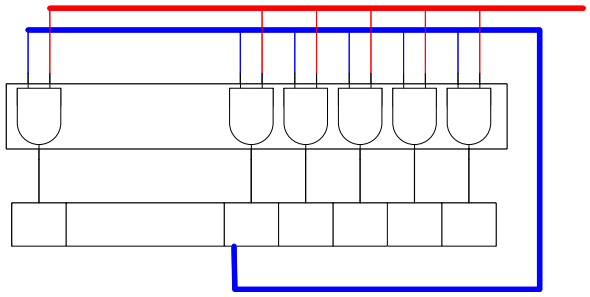


The SoCDMMU Hardware

The Basic SoCDMMU



Basic SoCDMMU



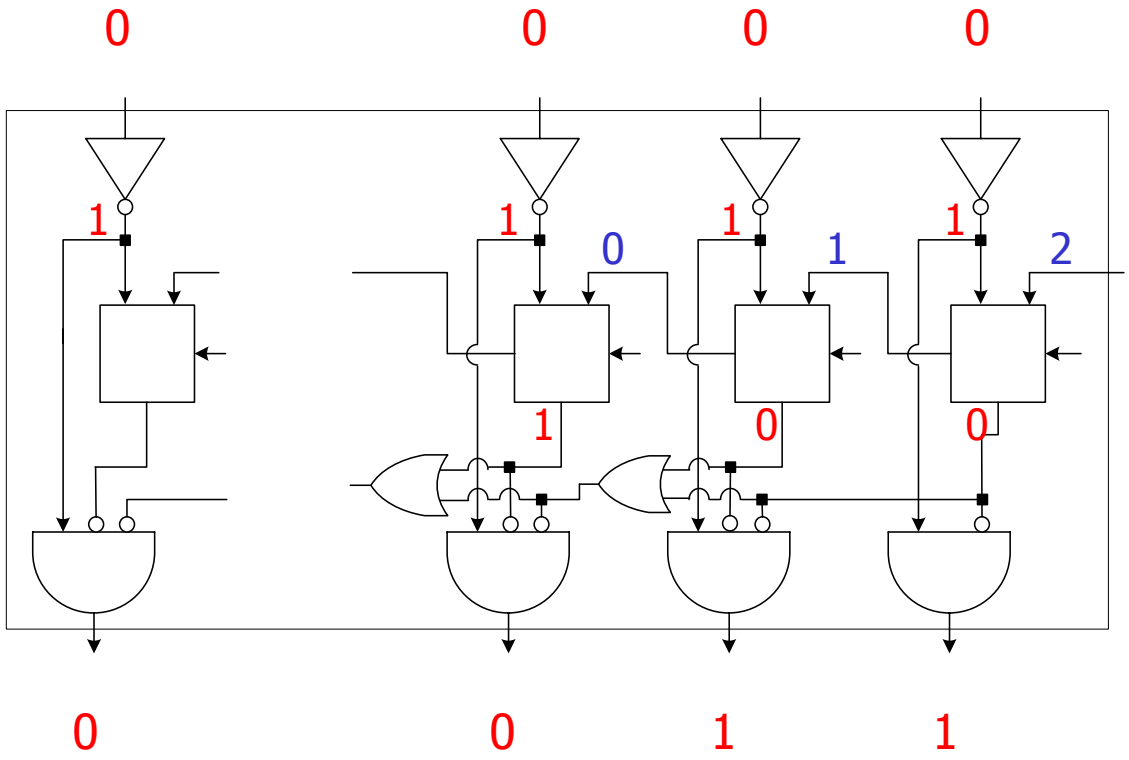
The SoCDMMU Hardware

The *Allocation Unit*

```
1 allocate(size,in[0:n-1]) {
2       for (i:=0 to n-1) {
3           if (in[i]==0 and size>0) {
4               out[i]:=1;
5               size:=size-1;
6           } else out[i]:=0;
7       }
8       if (size>0) return NOT_ENOUGH_MEMORY;
9       else return out;
10 }
```

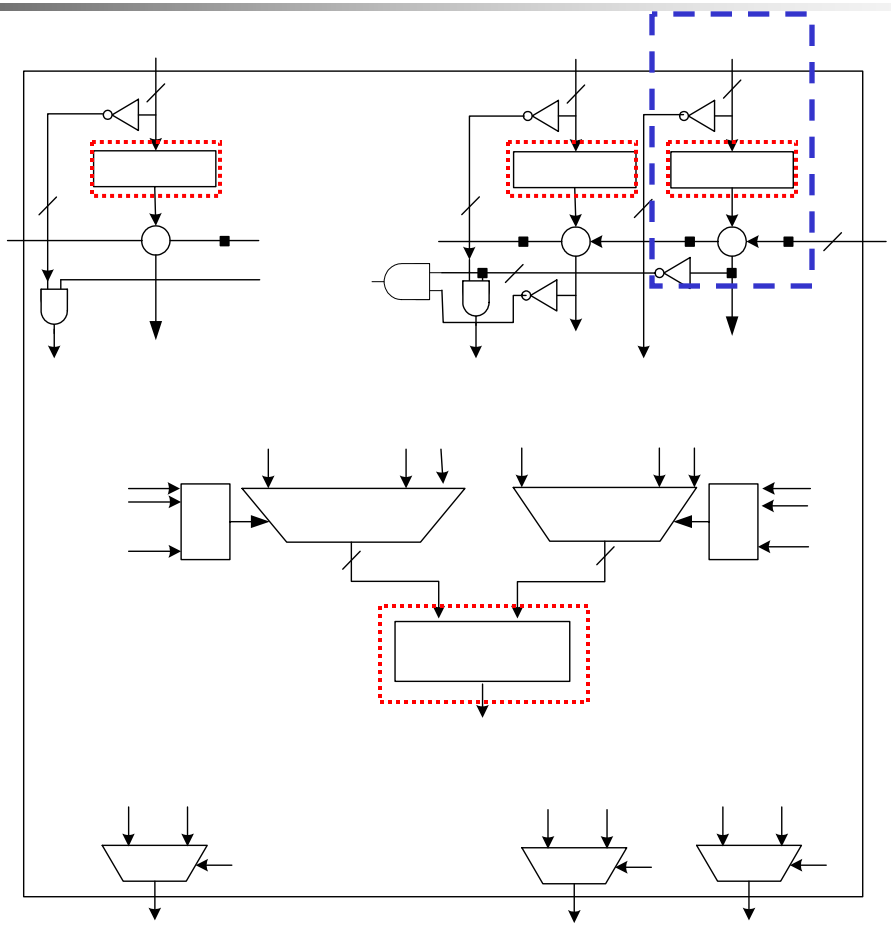
The SoCDMMU Hardware

The *Allocation Unit*



The SoCDMMU Hardware

The *Allocation Unit*



The SoCDMMU Hardware

The *Allocation Unit*

	Area (NAND gates)	**Worst Delay (ns)**	**Max. Clock Speed (MHz)**
Optimized Allocator	5364	6.6 ns	150 MHz
Un-optimized Alocator	17930	56.3 ns	17.5 MHz
Comparison	3.3X	8.5X	

- 256 *G_block*s.
- Synthesized using Synopsys Design Compiler™ and a TSMC 0.25u library from LEDA Systems.

The SoCDMMU Hardware

Execution Times/Synthesis

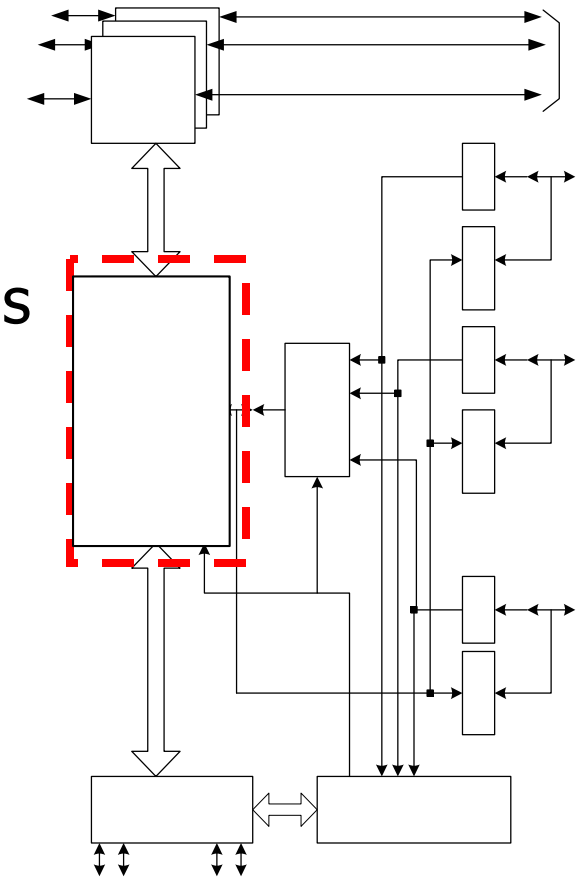
- Synthesized using the TSMC 0.25u .
- Clock Speed: 300MHz
- Size:
  - ~7500 gates (not including the Allocation Table and Address Converter)
  - Allocation Table: The size of 0.66KB 6T-SRAM
  - Address Converter: The size of 1.22 KB 6T-SRAM

Command	Number of Cycles
G_alloc_ex	4
G_alloc_rw	4
G_alloc_ro	3
G_dealloc	4
4-Processors WCET	**16**

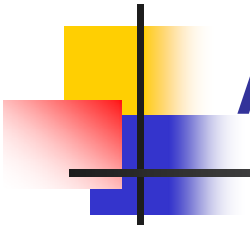
Microcontroller Implementation

Microcontroller Roles:

- Stores the allocation Status
- Executes the allocation commands
- Executes the de-allocation commands



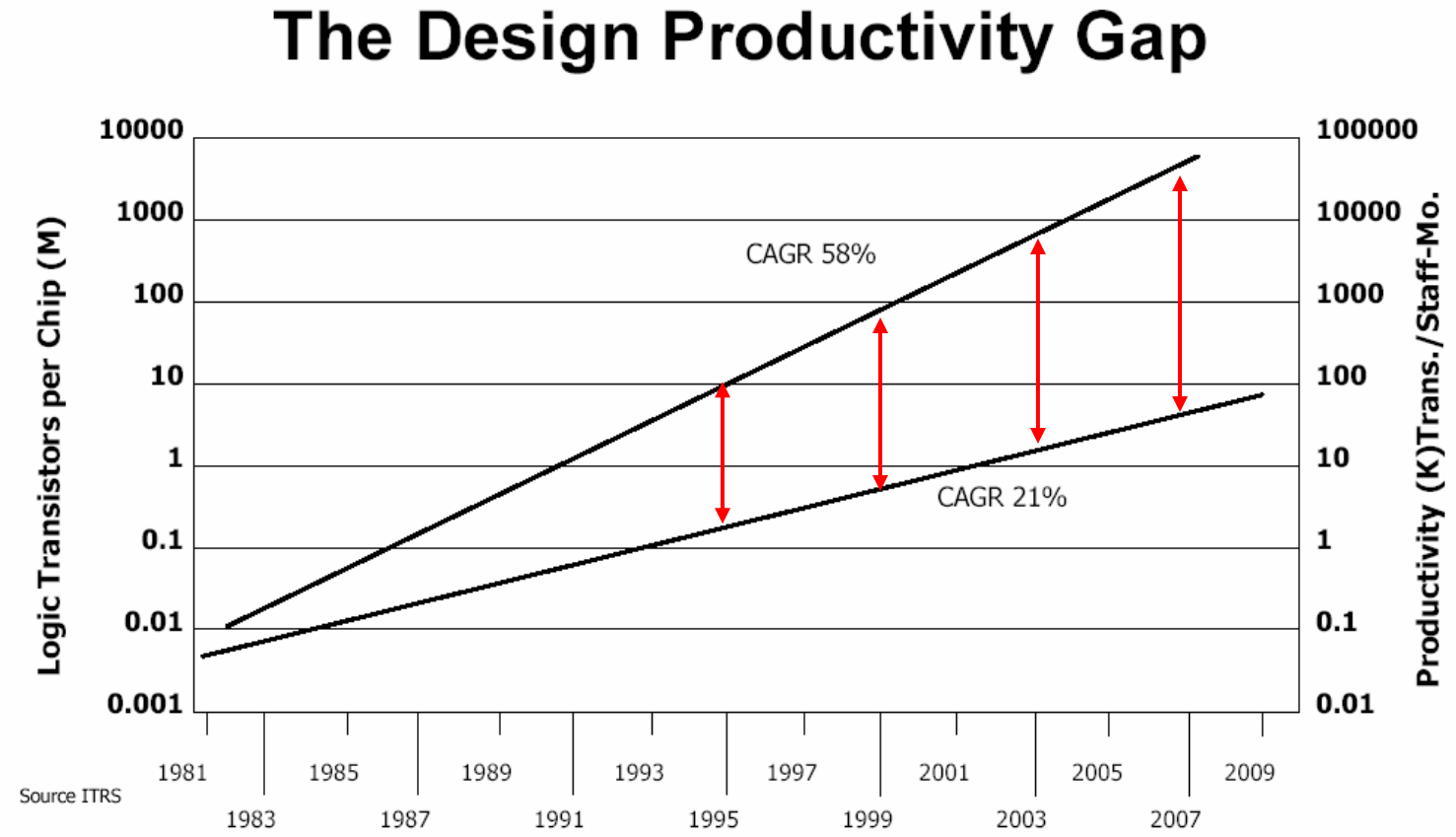
- Custom HW: 16 Cycles WCET
- uC: 231 Cycles BCET

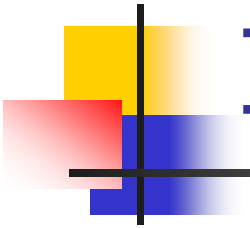


Agenda

- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments

Introduction

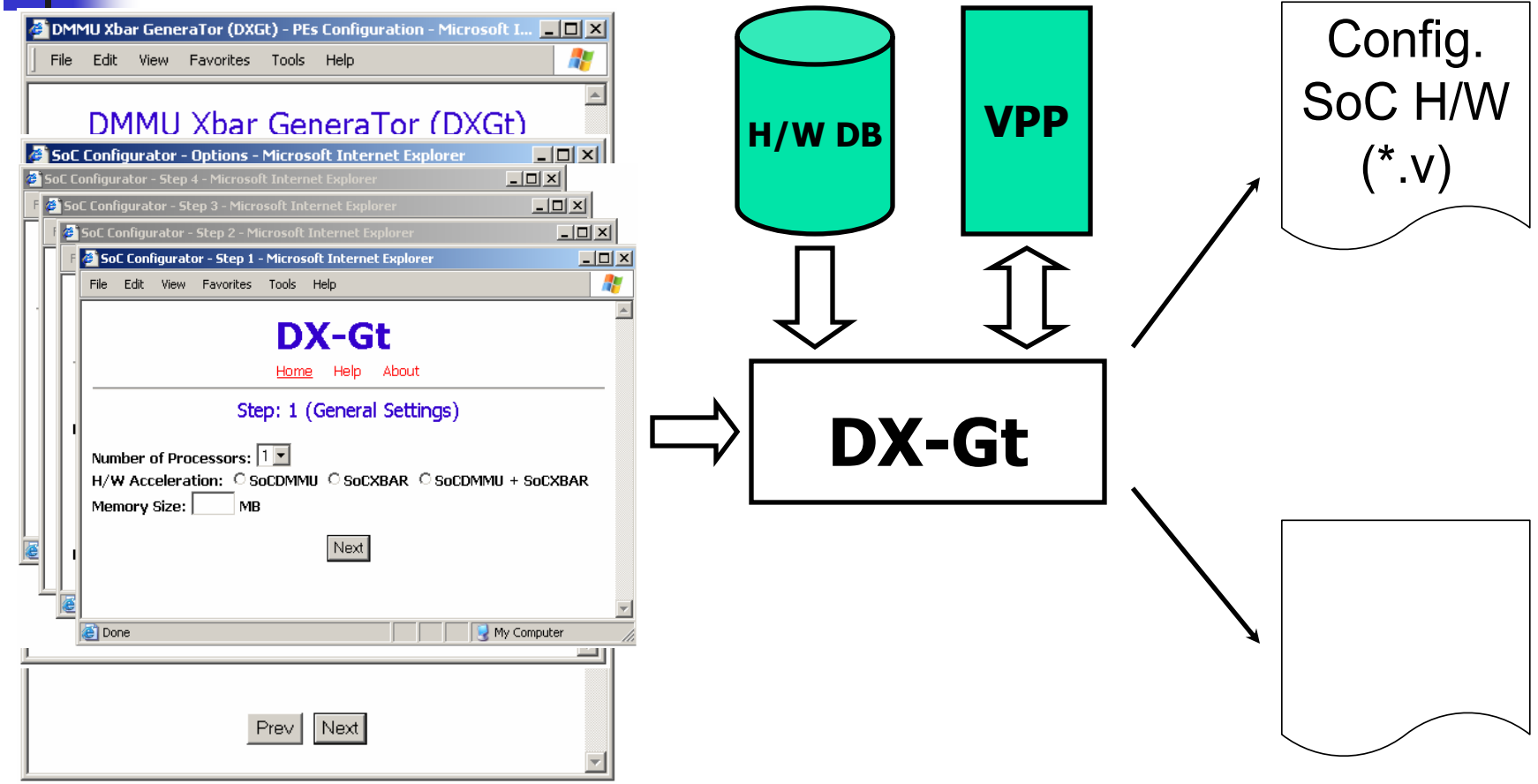


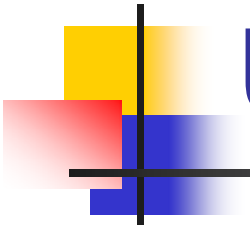


Introduction

- To overcome the productivity gap, Intellectual Property (IP) cores should be used in SoC designs
- Also, tools should be used to automatically customize/configure the IPs
  - Processor Generators: Tensilica, ARC Core, etc.
  - Memory Compilers: Artisan, LEDA, etc.
- The SoCDMMU as an IP core should be customized before being used in a system different than the one for which it was designed

DX-Gt Overview

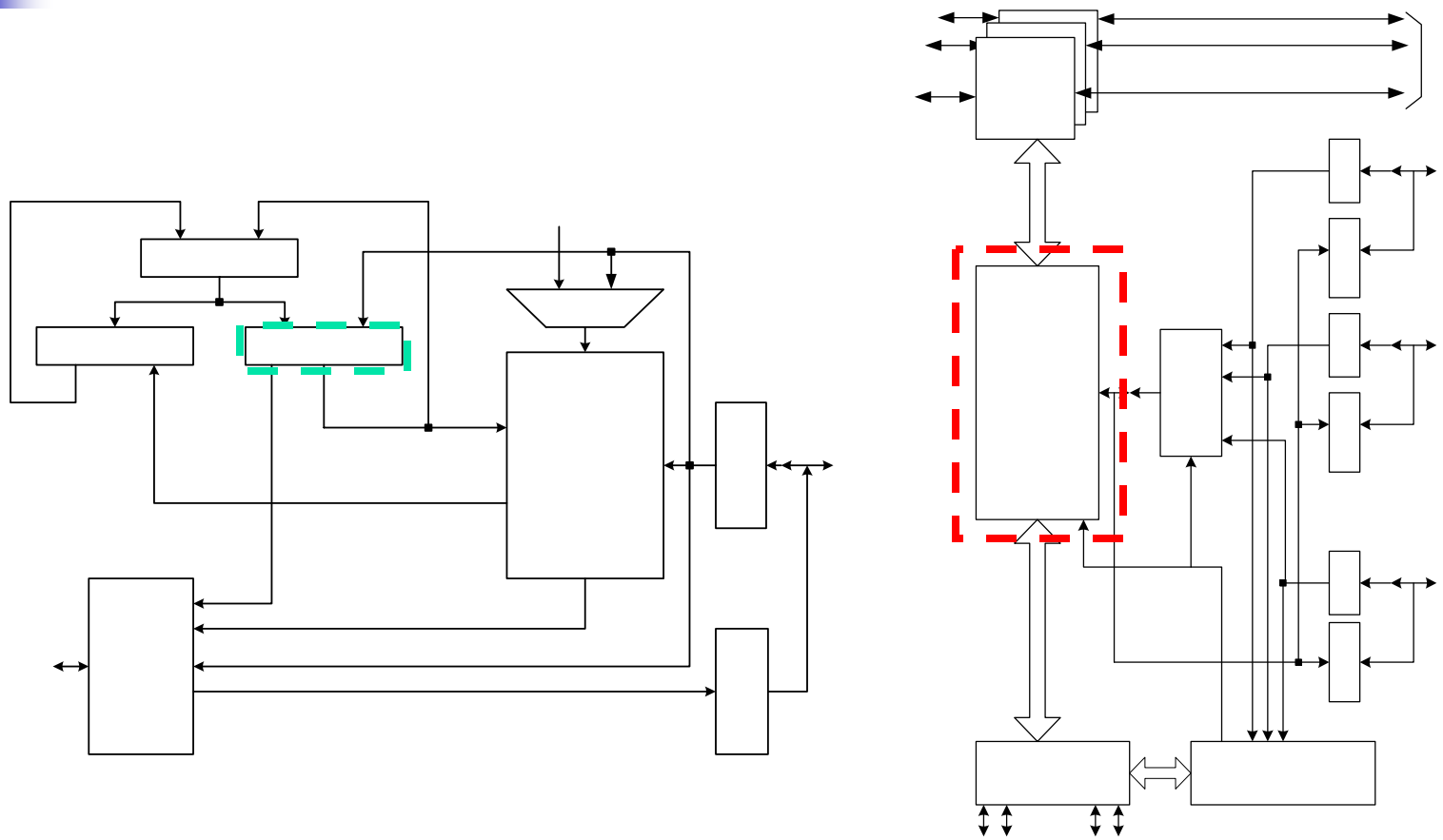




User Specified Parameters

- The number and type of PEs
- The number and size of the global on-chip memory *G_block*s
- The memory type
- The scheduling scheme to resolve concurrent SoCDMMU requests
- Memory *G_block*s initially assigned to the PEs

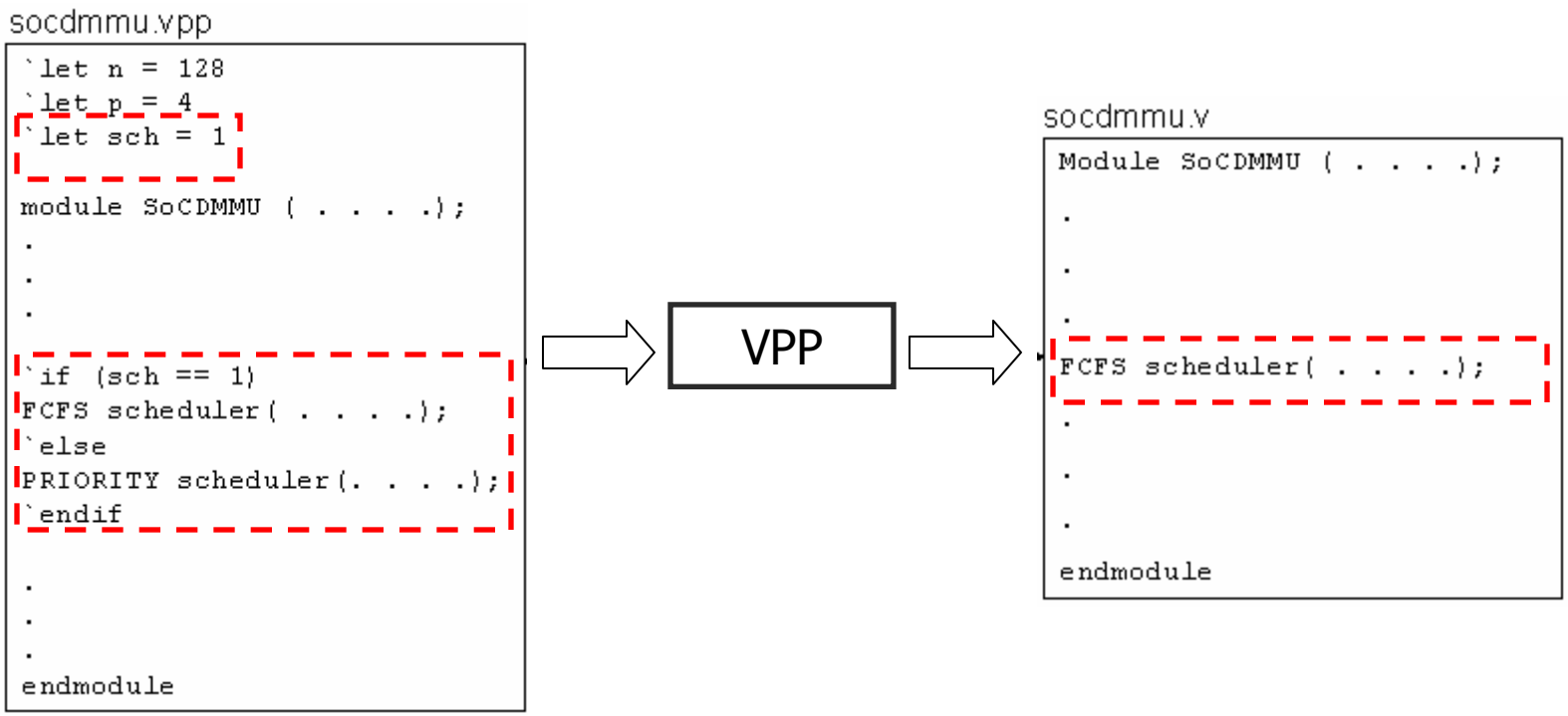
The SoCDMMU Generation



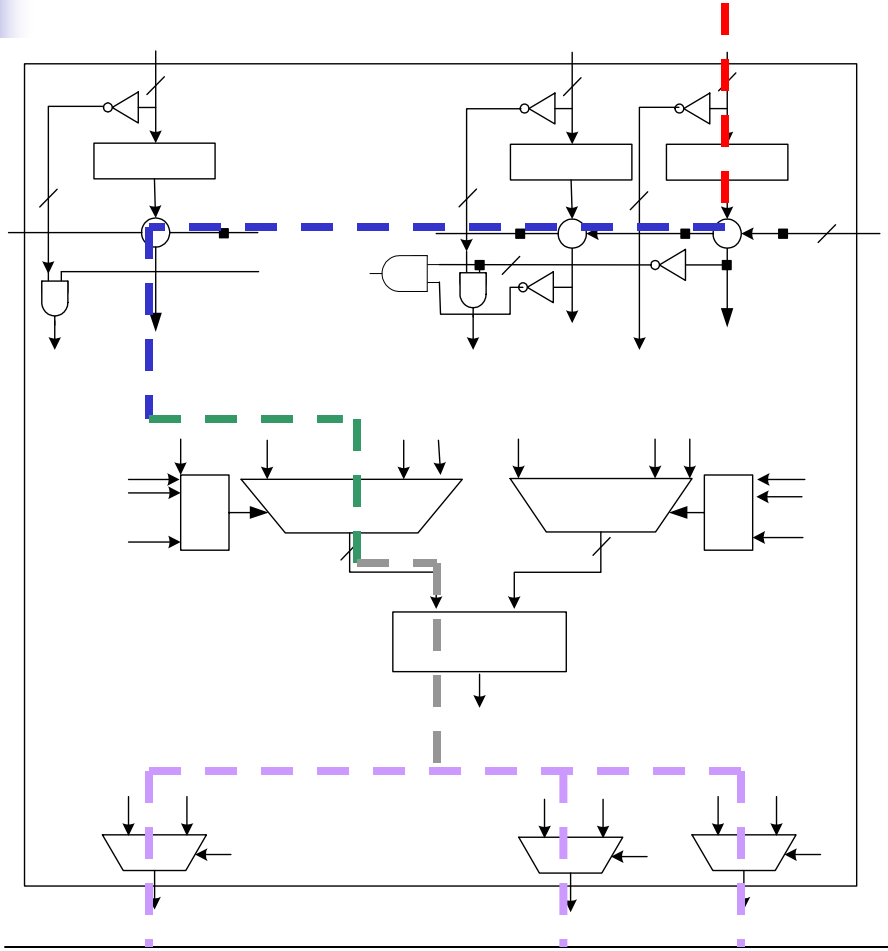
Customizing the SoCDMMU

- **Verilog Language**
  - `define & `ifdef
- **Verilog 2000/2001**
  - Generate loops (not supported by available tools)
- **Verilog PreProcessor (VPP)**
  - **`ifdef, `ifndef, `if, `let, `for, `while, `switch & `case**
  - **LOG2, ROUND, CEIL, FLOOR, EVEN, ODD, MAX, MIN & ABS**

Customizing the SoCDMMU



Allocation Unit Optimization



0's Counter

Almost Constant

k Subtractors

k x D_S

SZ_MUX

Almost Constant

1's Selector

m x d_1

MUX

Almost Constant

in[n-1:n-m-2]

Allocation Unit Optimization

- Delay over the critical path

  $\mathbf{Delay = C + k*D_s + m*d_1}$

- Also, we have

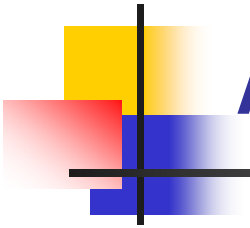
  $\mathbf{n = k * m}$: n is the no. of G_blocks

- This leads to

  $\mathbf{Delay = C + k*D_s + (n/k)*d_1}$

- The Delay is minimum when

  $\mathbf{k = SQR(n*d_1/D_s)}$: k is power of 2



Agenda

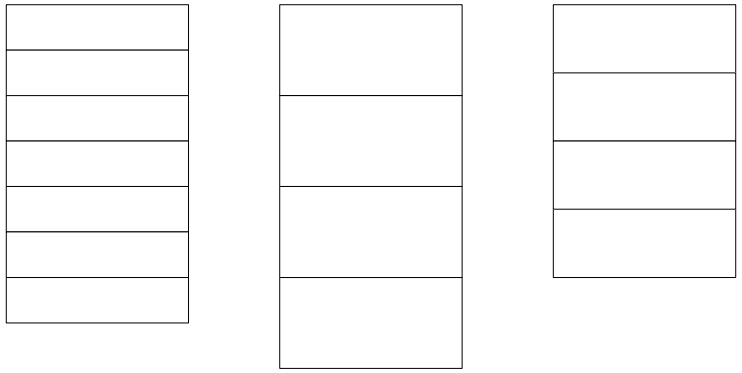
- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments

RTOS Support
Introduction

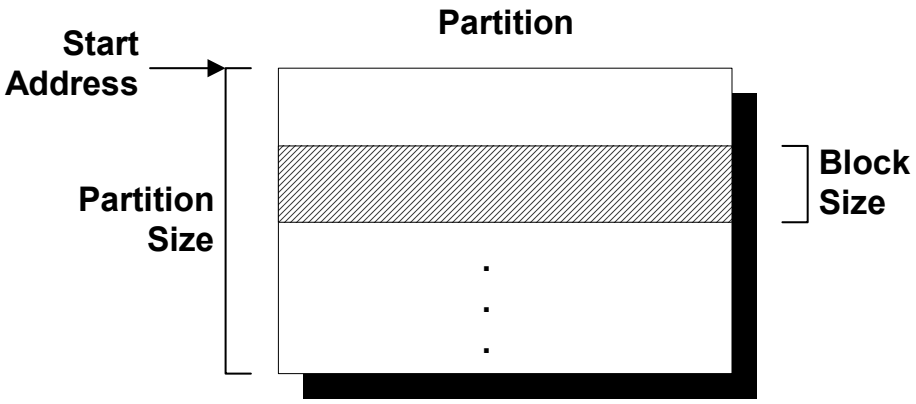
- Conventional memory allocation algorithms (e.g., Buddy-heap) are not suitable for Real-Time systems because they are not deterministic and/or the WCET is high
- This is mainly because of memory fragmentation and compaction. Also, most allocation algorithms usually use linked lists that do not have constant search time.
- An RTOS uses a different approach to make the allocation deterministic

RTOS Support
Introduction

- An RTOS (e.g., uCOS-II, eCOS, VRTXsa, etc.,) usually divides the memory into pools each of which is divided into fixed-sized allocation units and any task can allocate only one unit at a time



Atalanta Memory Management
Overview



- Atalanta is an open source RTOS developed at GaTech
- Atalanta allows tasks to obtain fixed-sized memory blocks from partitions made of a contiguous memory area
- Allocation and de-allocation of these memory blocks are done in a constant time
- No partition can be created at the run-time

Atalanta Memory Management
API Functions

- *asc_partition_gain*
  - Get free memory block from a partition (non-blocking)
- *asc_partition_seek*
  - Get free memory block from a partition (blocking)
- *asc_partition_free*
  - Free a memory block
- *asc_partition_reference*
  - Get partition information

Atalanta Support for the SocDMMU
Objectives

- Add Dynamic Memory Management to Atalanta
- Use the same Memory Management API Functions
- Keep the Memory Management Deterministic

Atalanta Support for the SocDMMU

Facts

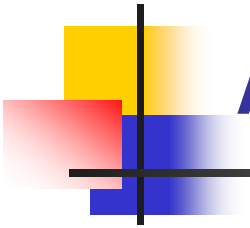
- The SoCDMMU needs to know where the allocated physical memory will be placed in the PE address space
- The PE address space is much larger than the physical address space (64 MB* vs. 4GB)
- The PE-Address Space Fragmentation can be overcome by:
  - Using the SoCDMUU *G_move* Command (pointers problems)
  - Replicate the physical address space

* A typical global on-chip memory size for billion transistor multiprocessor SoC

Atalanta Support for the SocDMMU

New API New Functions

Function Name	Description
asc_partition_create	Create a partition by requesting memory allocation from the SoCDMMU if necessary.
asc_partition_delete	Delete a partition and de-allocate memory block if required.
asc_memory_find	Find a place in the PE address space to which to map the allocated memory.

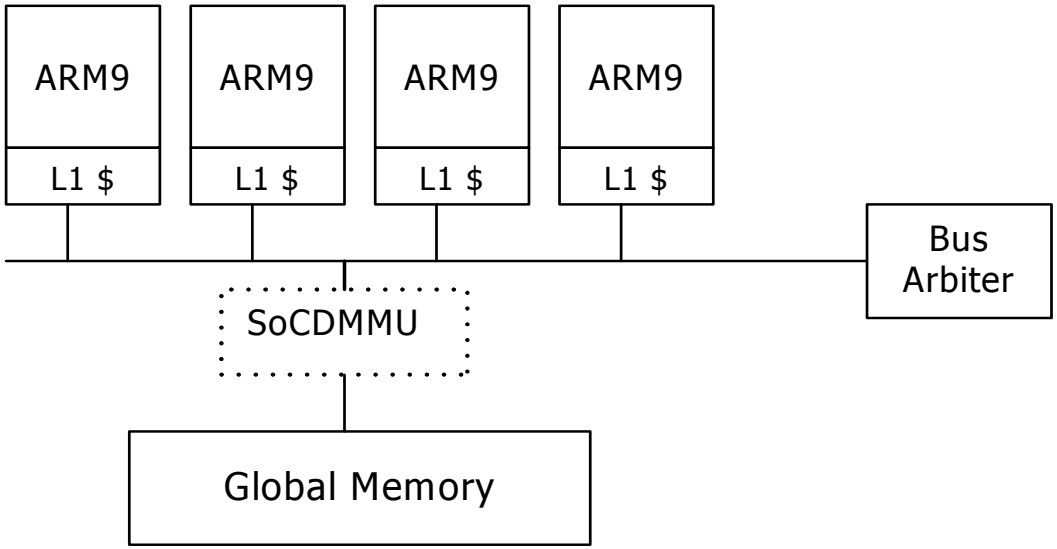


Agenda

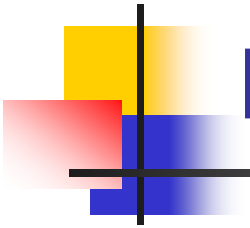
- Introduction & Motivation
- Dynamic Memory Management Background
- The SoCDMMU Programming Model
- The SoCDMMU
- Automatic Generation of Custom SoCDMMU
- RTOS Support
- Experiments

Comparison to a Fully Shared-Memory Multiprocessor System

Simulation Setup



- Simulation was carried out using Mentor Graphics Co-Verification Environment (CVE) , the cycle-accurate XRAY sotware simulator/debugger and Synopsys VCS Verilog simulator
- ARM SDT was used for software development



Experiment 1

- Global memory of 16MB; Data L1 $ is 64 KB, Instruction L1 $ is 64 KB
- The ARM runs at 150 MHz.
- Accessing the Global Memory costs 5 cycles for the first access
- A handheld device that utilizes this SoC can be used for OFDM communication as well as other applications (MPEG2 video player)
- Initially the device runs an MPEG2 video player. When the device detects an incoming signal it switches to the OFDM receiver. The switching time (which includes the time for memory management) should be short or the device might lose the incoming message

Experiment 1

- Sequence of Memory Allocations Required

MPEG-2 Player	OFDM Receiver
2 Kbytes	34 Kbytes
500 Kbytes	32 Kbytes
5 Kbytes	1 Kbytes
1500 Kbytes	1.5 Kbytes
1.5 Kbytes	32 Kbytes
0.5 Kbytes	8 Kbytes
	32 Kbytes

Experiment 1
Speedup of a single *malloc()*

	Execution Time (Average Case)	**Execution Time (Worst Case)**
SDT2.5 embedded malloc()	106 cycles	559 cycles
uClib *malloc()*	222 cycles	1646 cycles
SoCDMMU allocation	28 cycles	199 cycles
Speed up over SDT *malloc()*	3.78X	2.8X
Speed up over uClibc *malloc()*	7.92X	8.21X

Experiment 1
Speedup of a single *free()*

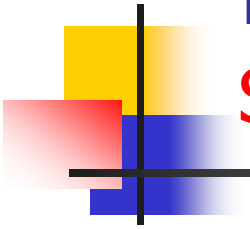
	Execution Time (Average Case)	**Execution Time (Worst Case)**
SDT2.5 embedded free()	83 cycles	186 cycles
uClib *free()*	208 cycles	796 cycles
SocDMMU deallocation	14 cycles	28 cycles
Speed up over SDT *free()*	5.9X	6.64X
Speed up over uClibc *free()*	14.8X	28.42X

Experiment 1
Speedup in transition time

	Using the SOCDMMU	Using SDT *malloc()* and *free()*	Speedup
Average Case	280 cycles	1240 cycles	4.4X
Worst Case	1244 cycles	4851 cycles	3.9X

	Using the SOCDMMU	Using uClibc *malloc()* and *free()*	Speedup
Average Case	280 cycles	2593 cycles	9.26X
Worst Case	1244 cycles	15502 cycles	12.46X

Experiment 2
Speedup in Execution Time



- Same setup used for Experiment 1
- GCC and Glibc were used for development
- 3 kernels from the SPLASH-2 application suite are used
  - Complex 1D FFT (FFT)
  - Integer RADIX sort (RADIX)
  - Blocked LU decomposition (LU)
- They were modified to replace all the static memory allocations by dynamic ones

Experiment 2

Speedup in Execution Time

- *Glibc malloc() & free()*

Benchmark	E.T. (Cycles)	Memory Management E. T. (Cycles)	% of E. T. used to Memory Management
LU	318307	31512	9.90%
FFT	375988	101998	27.13%
RADIX	694333	141491	20.38%

- Using the SoCDMMU

Benchmark	E.T. (Cycles)	Memory Management E. T. (Cycles)	% of E. T. used to Memory Management	% Reduction in Time used to Manage Memory	% Reduction in Benchmark E. T.
LU	288271	1476	0.51%	95.31%	9.44%
FFT	276941	2951	1.07%	97.10%	26.34%
RADIX	558347	5505	0.99%	96.10%	19.59%

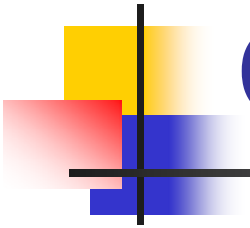
Area Estimation of The SoC

Element	Number of Transistors
4 ARM9TDMI Cores	4 x 112K = 448K Transistors
4 L1 Caches (64KB+64KB)	4 x 6.5M = 26M Transistors*
Global On-Chip Memory (16MB)	134.217M Transistors
SoCDMMU (w/o memory elements)	30K Transistors
SoCDMMU *Allocation Table*	30K Transistors
SoCDMMU *Address Converters* (4)	4 x 60K = 240K Transistors
SoCDMMU (total)	**300K Transistors**
SoC (total)	**160.965M Transistors**
SoCDMMU w/o memory elements to SoC	**0.0186%**
SoCDMMU to SoC (%)	**0.186%**

* Using dual-port 6T SRAM Cells

Area Estimation of The SoC

- For this 161 Million transistor chip, the SoCDMMU consumes 300K transistors (0.186% of 161M) and yields a 4-10X speedup in memory allocation/de-allocation



Conclusion

- We introduced The Two-Level memory management hierarchy for multiprocessor SoC
- We showed how Level Two in the hierarchy can be implemented using the SoCDMMU
- We gave a sample hardware implementation of the SoCDMMU
- We introduced DX-Gt to automatically configure/customize the SoCDMU hardware
- We showed how to add the SoCDMMU support to a real-time OS
- Our Experiments show that using the SoCDMMU speeds up the application transition time as well as the application execution time

Topic Related Publications

- M. Shalan and V. Mooney, "A Dynamic Memory Management Unit for Embedded Real-Time System-on-a-Chip," *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2000)*, pp. 180-186, November 2000.
- M. Shalan and V. Mooney, "Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management," *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, pp. 79-84, May 2002.
- M. Shalan, E. Shin and V. Mooney, "DX-Gt: Memory Management and Crossbar Switch Generator for Multiprocessor System-on-a-Chip," to appear in the *Proceedings of the 11th Workshop on Synthesis and System Integration of Mixed Information technologies (SASIMI 2003)*, April 2003.
- M. Shalan and V. Mooney, "Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management," Accepted for publication in *ACM Transactions in Embedded Computing Systems (TECS)*.
- M. Shalan and V. Mooney, "Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management," Georgia Institute of Technology, Atlanta, Georgia, Technical Report GIT-CC-03-02, 2003.
- Hardware Software Real-Time Operating System, The δ RTOS, preparing 2 chapters

Questions