Research Trends in Hardware/ Software Codesign of Embedded Operating Systems for FPGAs

Vincent J. Mooney III
http://codesign.ece.gatech.edu
http://www.crest.gatech.edu

Associate Professor, School of Electrical and Computer Engineering
Adjunct Associate Professor, College of Computing
Co-Director, Center for Research on Embedded Systems and Technology
Georgia Institute of Technology
Atlanta, Georgia, USA

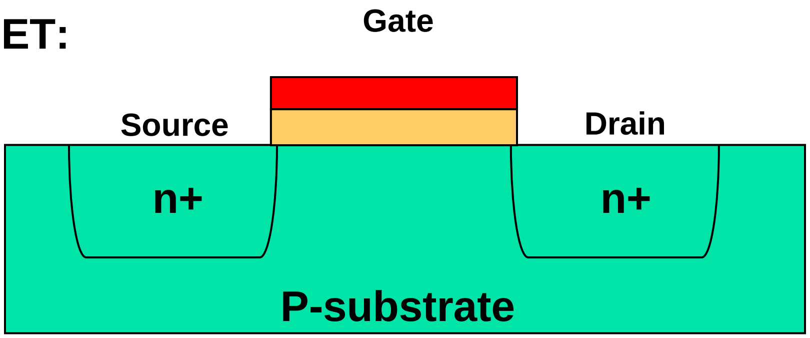
FPGAworld 13 September 2007

Outline

- Trends
- Custom RTOS Hardware IP
  - System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)
  - The δ Hardware/Software RTOS Generation Framework
- Compilation for Reverse Execution & Debugging
- Conclusion

Moore’s Prediction (“Law”)

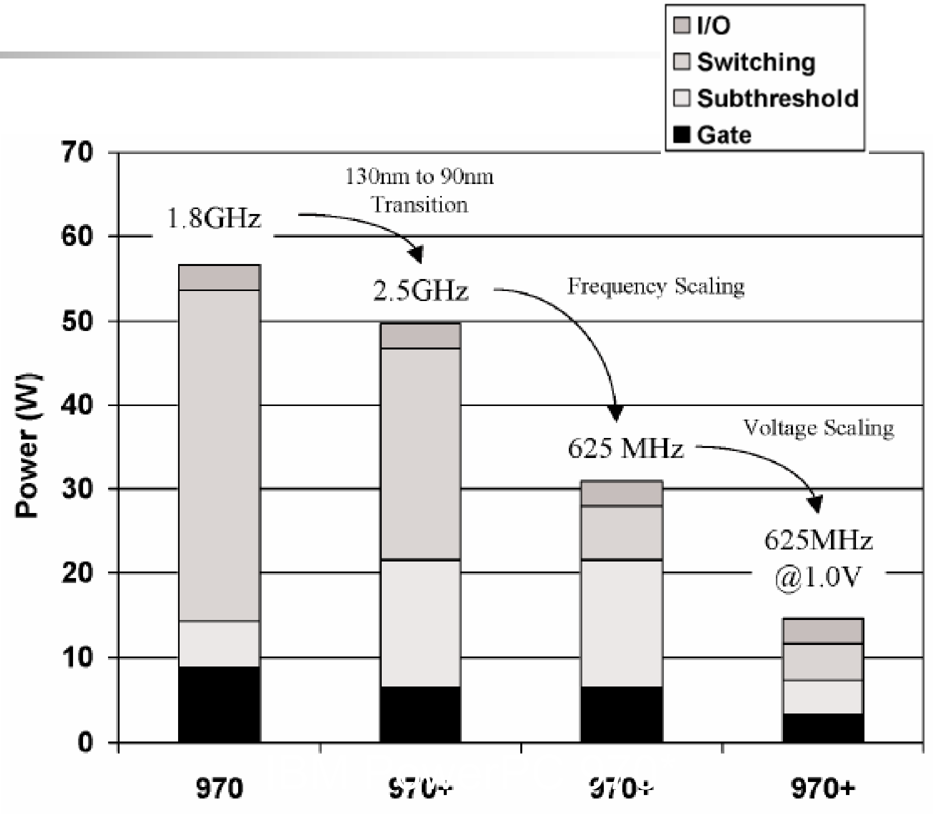
NFET:



- Si atom ~ 2Å
- .09u = 90nm = 900Å   July 2004
  - ⇒ ~450 atoms, 90^2 = 8100
- .065u = 65nm = 650Å   Jan 2006
  - ⇒ ~325 atoms, 65^2 = 4225
- .045u = 45nm = 450Å   July 2007
  - ⇒ ~225 atoms, 45^2 = 2025
- .032u = 32nm = 320Å   Jan 2009
  - ⇒ ~160 atoms, 32^2 = 1024
- .022u = 22nm = 220Å   July 2010
  - ⇒ ~110 atoms, 22^2 = 488
- .016u = 16nm = 160Å   Jan 2012
  - ⇒ ~80 atoms, 16^2 = 256
- .011u = 11nm = 110Å   July 2013
  - ⇒ ~55 atoms, 11^2 = 121
- .008u = 8nm = 80Å   Jan 2015
  - ⇒ ~40 atoms, 8^2 = 64
- .006u = 6nm = 60Å   July 2016
  - ⇒ ~30 atoms, 6^2 = 36
- .004u = 4nm = 40Å   Jan 2018
  - ⇒ ~20 atoms, 4^2 = 16
- .003u = 3nm = 30Å   July 2019
  - ⇒ ~15 atoms, 3^2 = 9
- .002u = 2nm = 20Å   Jan 2020
  - ⇒ ~10 atoms, 2^2 = 4

Power Consumption

- Power consumption of VLSI is a fundamental problem of mobile devices as well high-performance computers
  - Limited operation (battery life)
  - Heat
  - Operation cost
- Power = dynamic + static
  - Dynamic power more than 90% of total power (0.18u tech. and above)
  - Static nearly equal to dynamic power for latest processes (.65u)
- Dynamic power reduction:
  - Technology scaling
  - Frequency scaling
  - Voltage scaling

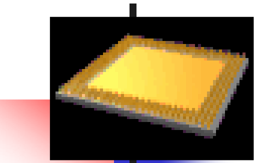


*N. Rohrer et al., “PowerPC 970 in 130nm and 90nm Technologies," *IEEE International Solid-State Circuits Conference,* vol 1, pp. 68-69, February 2004.

Cost of latest Silicon Processes

- Mask sets for 90nm: over $1 million (U.S. $)
- NRE costs $10 to $100 million U.S.
  - Verification costs can exceed design costs
  - Embedded software costs may exceed hardware costs

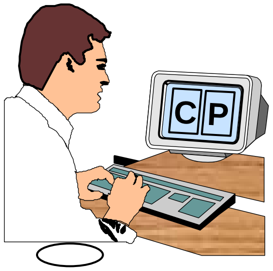
Increasing Cost of Customization*

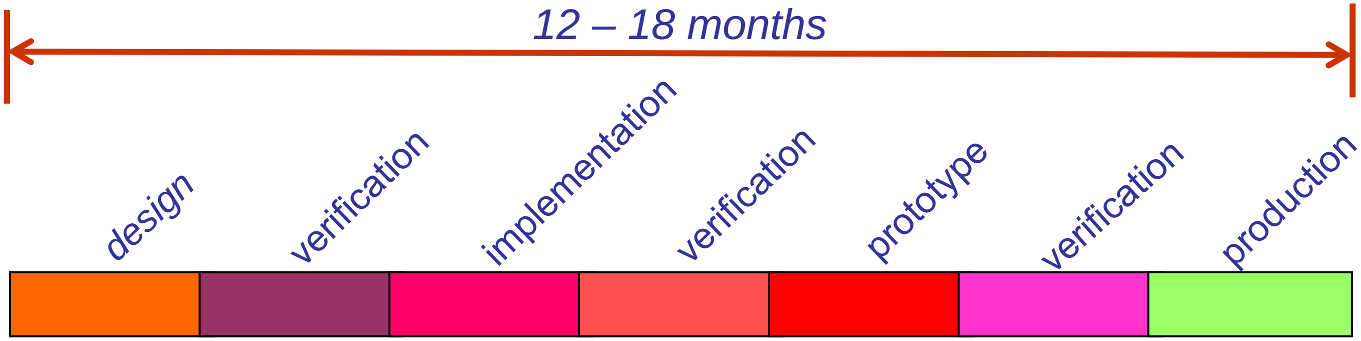


Estimated Cost - $85 M -$90 M

Example: Design with 80 M transistors in 100 nm technology

- Cost and Risk rising to unacceptable levels
- Top cost drivers
  - Verification (40%)
  - Architecture Design (26%)
  - Embedded Software Design
    - 1400 man months (SW)
    - 1150 man months (HW)
  - HW/SW integration





12 – 18 months

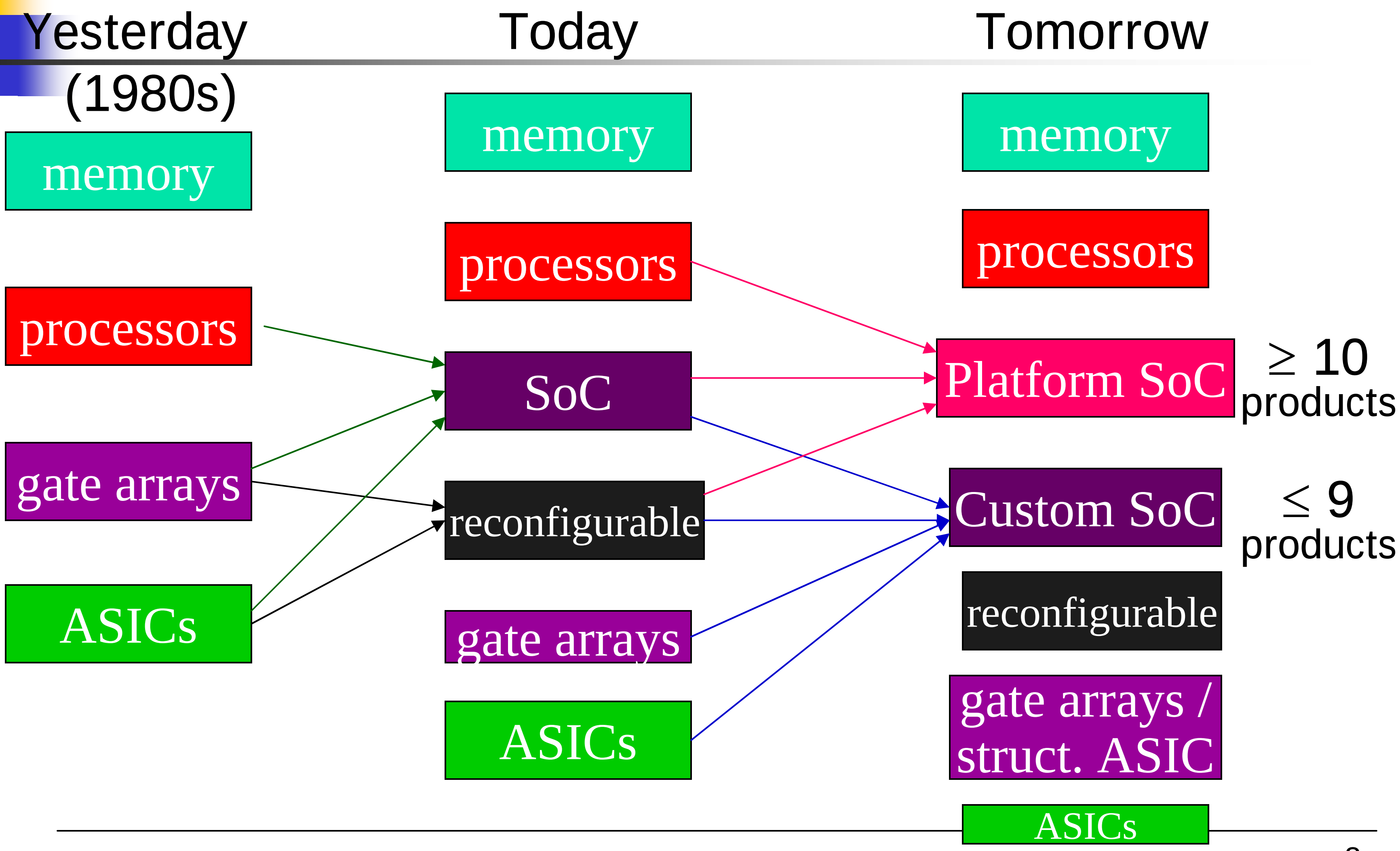
design, verification, implementation, verification, prototype, verification, production

*Handel H. Jones, "How to Slow the Design Cost Spiral," *Electronics Design Chain*, September 2002, www.designchain.com

Yearly Chip Design Starts

- Roughly 8,000 in 1996
- Peaked at approx. 12,000 in 2000
- Recent years approx. 3,000 per year
- However, comparing, say, 2005 to 2000, number of EDA tool licenses purchased roughly the same
  - Larger team per chip design start
- System level chip design 2005
  - 33.5% U.S.
  - 26.1% Japan
  - 9.9% Taiwan, 5.8% Germany, 5.4% China, 5.3% Korea

Digital Silicon CMOS VLSI Trends



Embedded Systems Market Drivers

- 1950s-1980s: Business Applications
- 1990s-Today: Consumer Electronics

- Some observations
  - profit margins tight in consumer
  - volumes and risk high in consumer
  - supply chain as or more important than raw technology

Chip Software Business Models

- Electronics Design Automation
  - Synopsys, Cadence, Mentor, Synplicity
  - Yearly/quarterly subscriptions
- Embedded Operating System Companies
  - Windriver, Montavista, Timesys, Integrity, etc.
  - Upgrades, debuggers, Integrated Development Environment (IDE)
  - Per-seat costs 1/10 of EDA
    - $10K per seat (as compared to $100K for EDA)
- Cooley's observation
  - FPGA software sells for far less than ASIC software
  - Many FGPA companies give away software (lock-in...)

Failed Reconfigurable Attempts

- ST
- Phillips
- Texas Instruments
- Pilkington
- Atmel
- Samsung
- Motorola
- IBM
- Vantis
- Lattice
- Quicksilver
- Concurrent Logic
- Toshiba
- Chameleon
- Plessey
- Adaptive Silicon
- Dynachip
- Crosspoint
- Lucent
- National Semiconductor

A Few Successes

- Triscend
- Synplicity

- Note1:
  - Xilinx, Altera: each probably have, as a lower bound, approx. 12 million lines of code for FPGA programming tools
- Note2:
  - Key reconfigurable patents from 1970s have expired

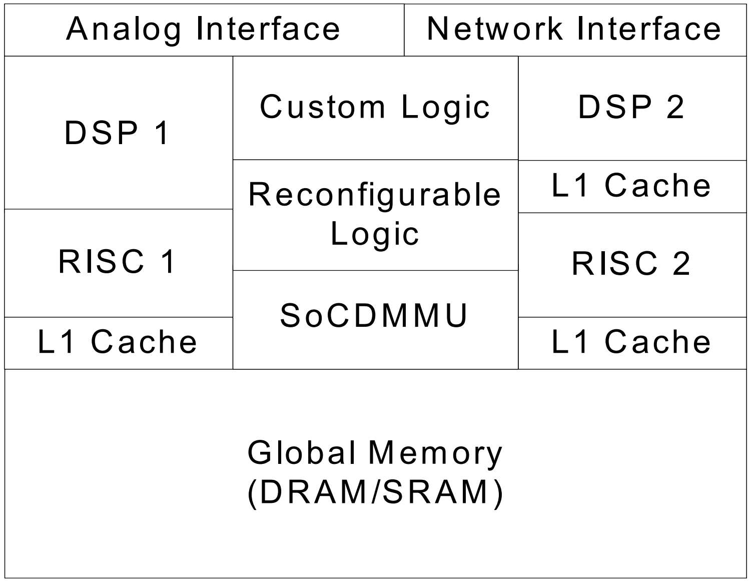
Comments (probably incorrect) on Trends!

- Reconfigurable like to increase superlinearly (increasing % of market)
  - Tools critical to adoption
- Chips for consumer electronics to drive technology
  - E.g., Cell processor
- Embedded software and operating systems more important and less supported than ever
- Breakthrough platform SoC design flows to determine chip success
  - E.g., will not be Cell because too difficult to program...

Outline

- Trends
- Custom Operating System Hardware IP
  - System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)
  - The δ Hardware/Software RTOS Generation Framework
- Compilation for Reverse Execution & Debugging
- Conclusion

System-on-a-Chip (SoC)



Analog Interface

Network Interface

DSP 1

Custom Logic

DSP 2

L1 Cache

Reconfigurable Logic

RISC 1

RISC 2

SoCDMMU

L1 Cache

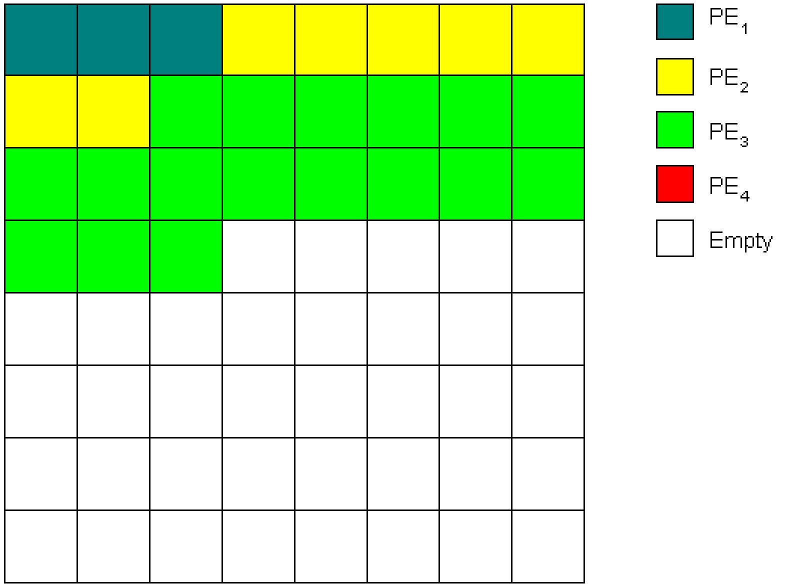
L1 Cache

Global Memory (DRAM/SRAM)

- 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)

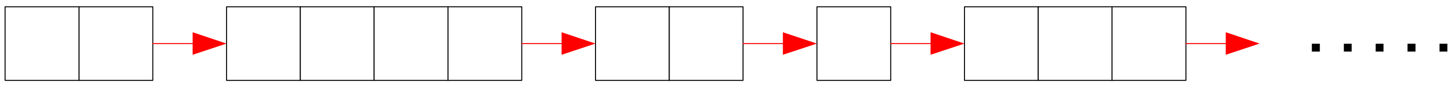
Dynamic Memory Management

- 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**

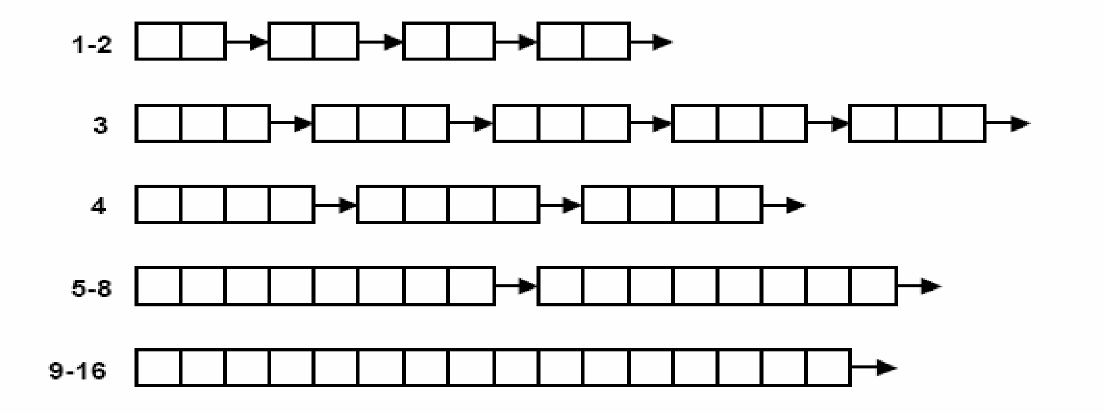


  - First Fit,
  - Next Fit,
  - Best Fit or
  - Worst Fit

Memory Allocation

Software Techniques

- **Segregated Free Lists**

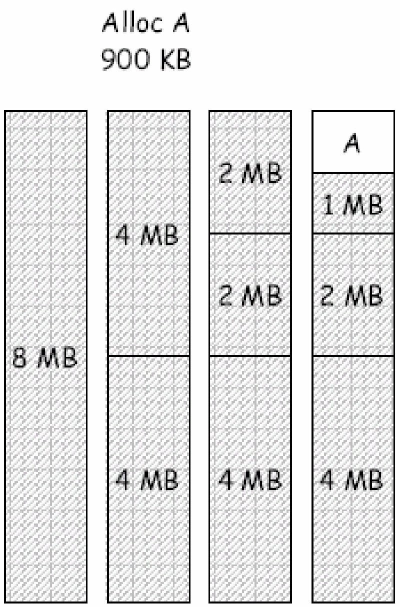


  - Simple Segregated Storage
  - Segregated Fit

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

1	1	1	0	0	1	1	1
0	1	2	3	4	5	6	7

- **Cam et al.** *

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

* References are available in the thesis by M. Shalan

Memory Allocation

Hardware Techniques

0	0	0	1	0	1	1	1
0	1	2	3	4	5	6	7

- Request size is 3
- It searches for 4

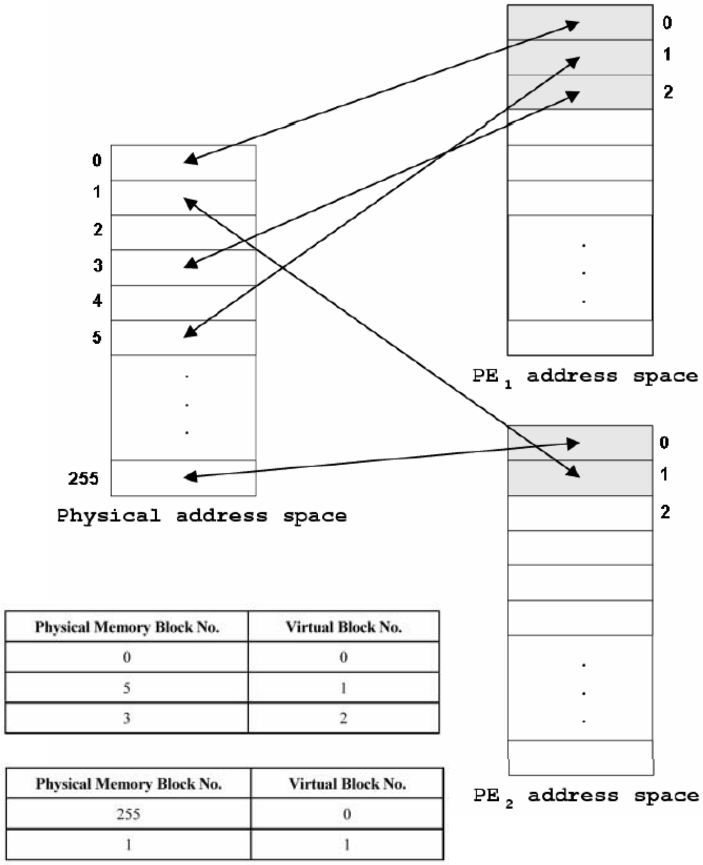
  [3 rounded to the nearest power of 2]

Memory Model 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

Memory Model Assumptions (continued)

- Each memory block has one physical address and one or more virtual addresses. The block virtual address may differ from 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

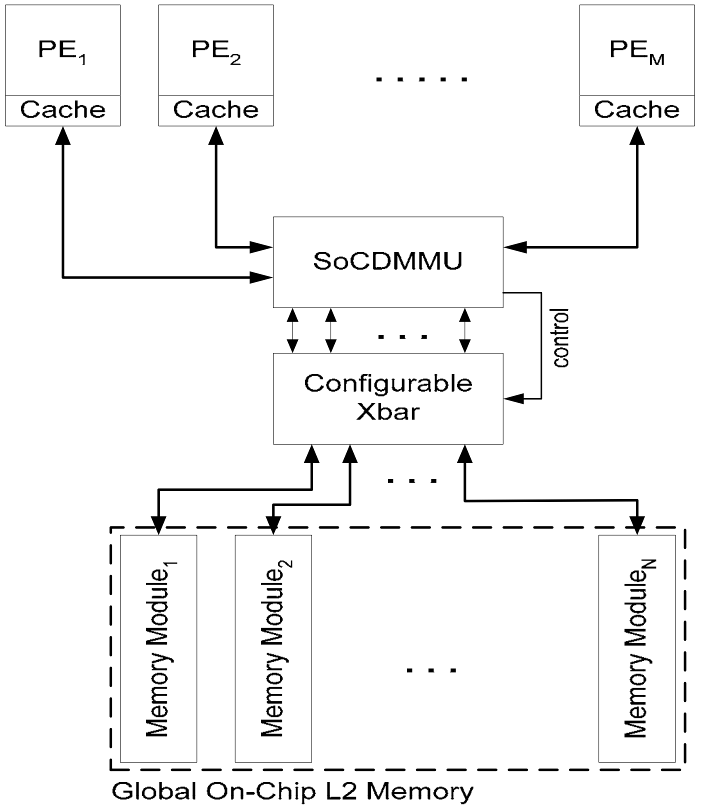
Two-Level Memory Management

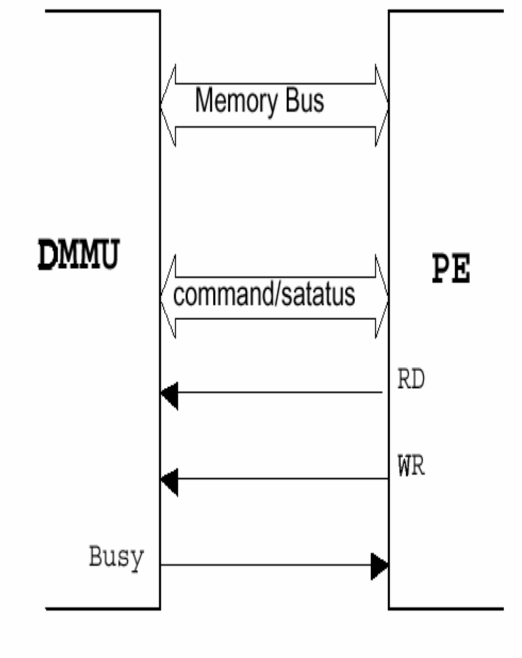
- The SoCDMMU manages the memory between the PEs
- The OS (or custom software) on each PE manages the memory between the processes that run on that PE
- The process requests the memory allocation from the OS or custom software. If there in not enough memory, the OS requests memory allocation from the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

Types of Memory Allocation

- **Exclusive**
  - Only the the owner can access it. No other PE can access it
- **Read / Write**
  - The owner can read/write to it. Other PE's can read from it if it *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

PE-SoCDMMU Interface

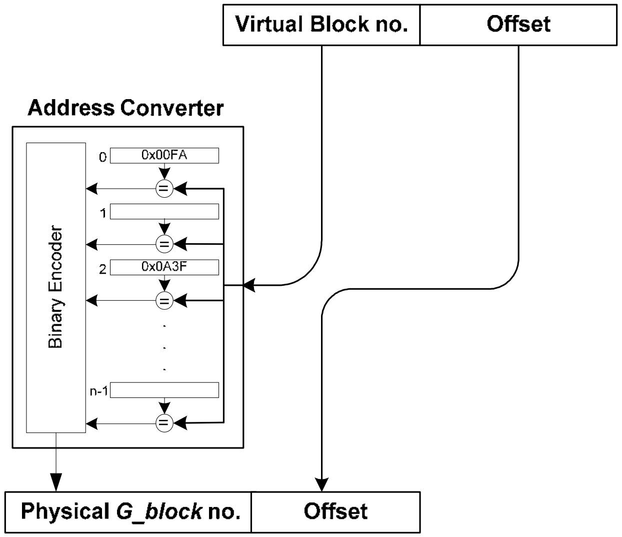




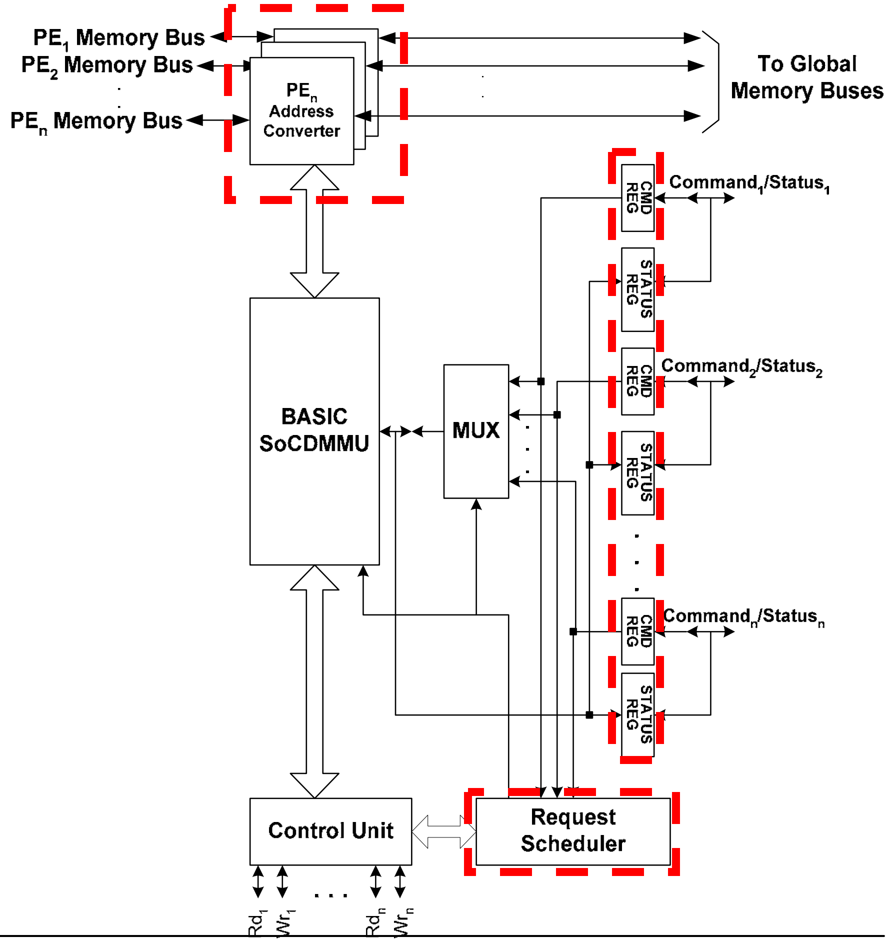
SoCDMMU Commands

G_alloc_ex	**Page ID (SW ID)**	**Size**	**Virtual Block No.**	**000**
G_alloc_rw	**Page ID (SW ID)**	**Size**	**Virtual Block No.**	**001**
G_alloc_ro	**Page ID (SW ID)**	**n/a**	**Virtual Block No.**	**010**
G_dealloc	**Page ID (SW ID)**	**n/a**	**n/a**	**011**
G_move	**Page ID (SW ID)**	**n/a**	**Virtual Block No.**	**100**

The SoCDMMU Hardware

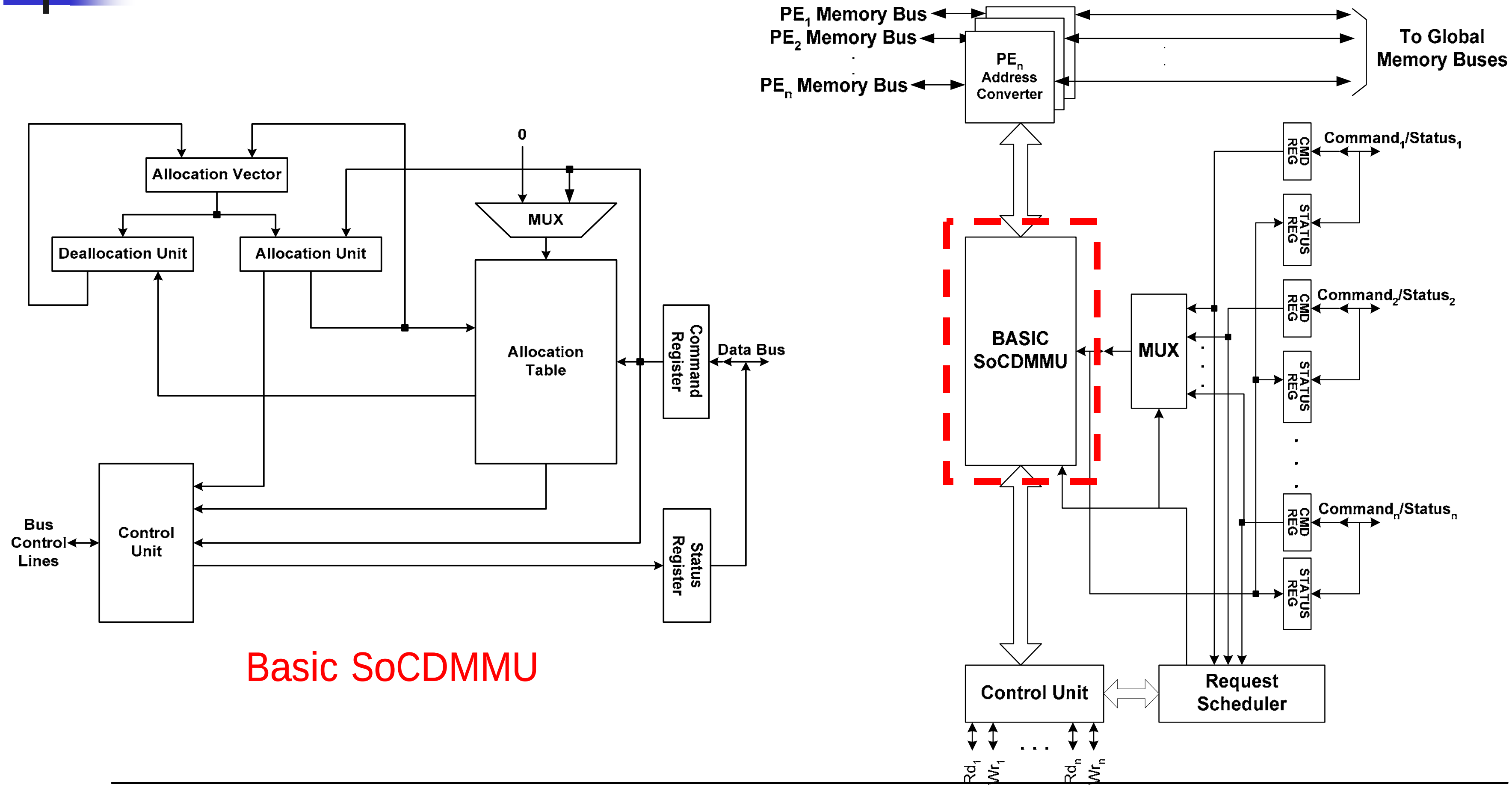


Address Converter



The SoCDMMU Hardware

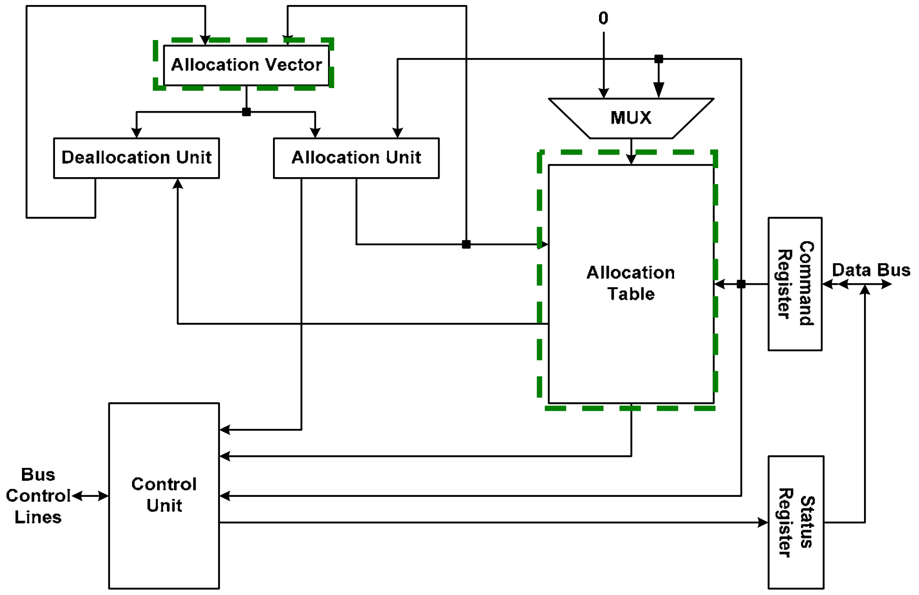
The Basic SoCDMMU



Basic SoCDMMU

The SoCDMMU Hardware

The Basic SoCDMMU



Basic SoCDMMU

255	.....			5	4	3	2	1	0
0	.....	0	0	0	0	1	1	1	1

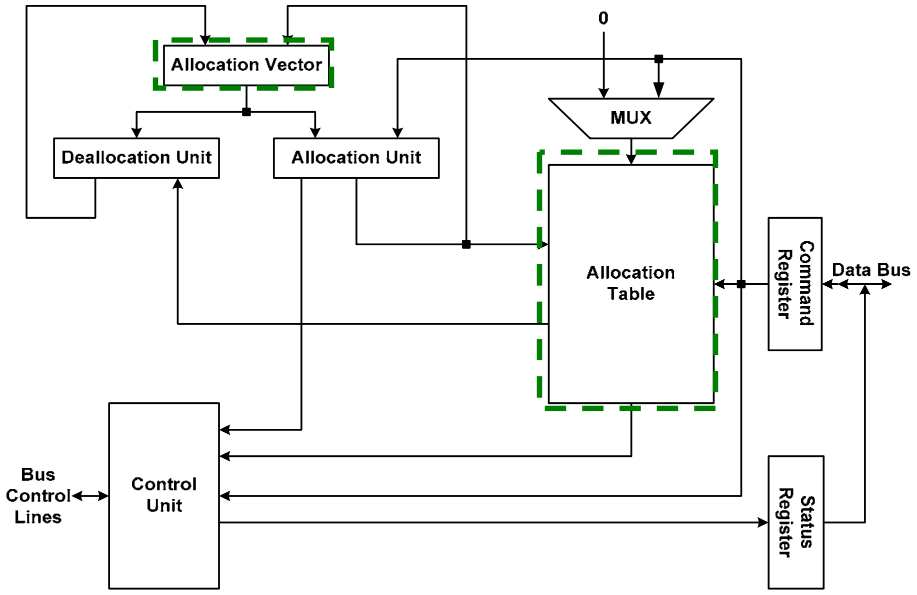
Allocation Vector

	MODE	PE	SW ID
0	Ex	PE 1	25
1			
2			
3	Ex	PE 2	11
.			
.			
.			
.			
255			

Allocation Table

The SoCDMMU Hardware

The Basic SoCDMMU



Basic SoCDMMU

255	. . . . .			5	4	3	2	1	0
0	. . . . .	0	0	0	0	1	**1**	**1**	1

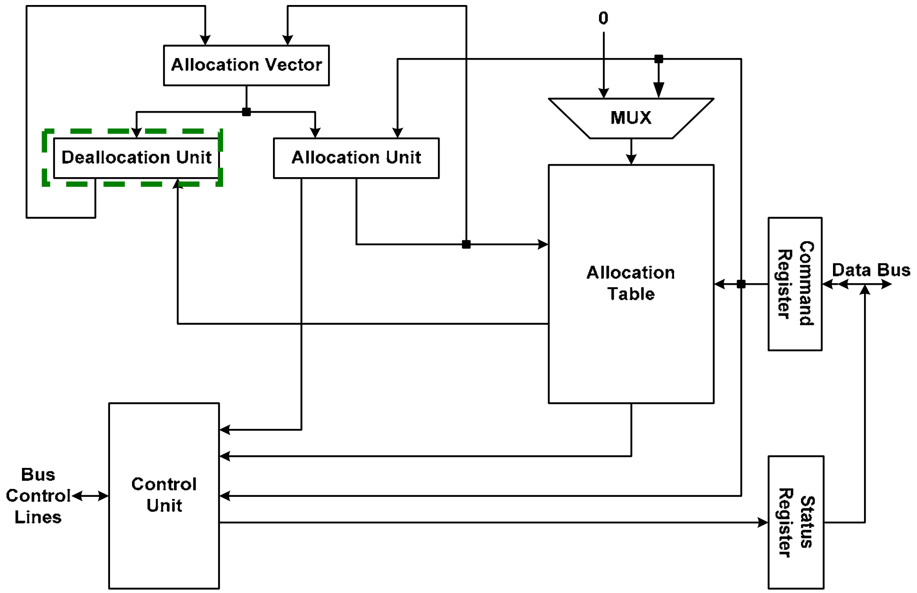
Allocation Vector

	MODE	PE	SW ID
0	Ex	PE 1	25
1	**RW**	**PE 1**	**30**
2	**RW**	**PE 1**	**30**
3	Ex	PE 2	11
.			
.			
.			
.			
255			

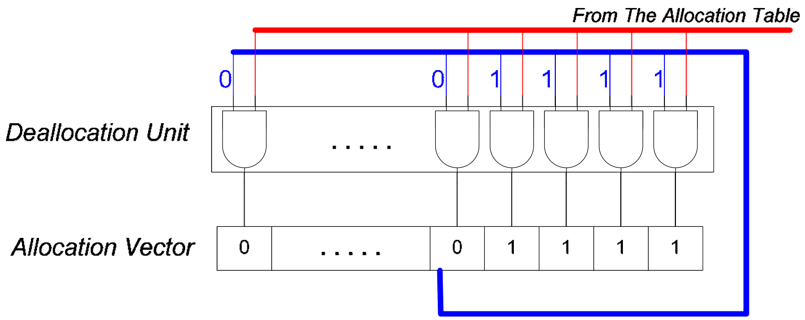
Allocation Table

The SoCDMMU Hardware

The Basic SoCDMMU

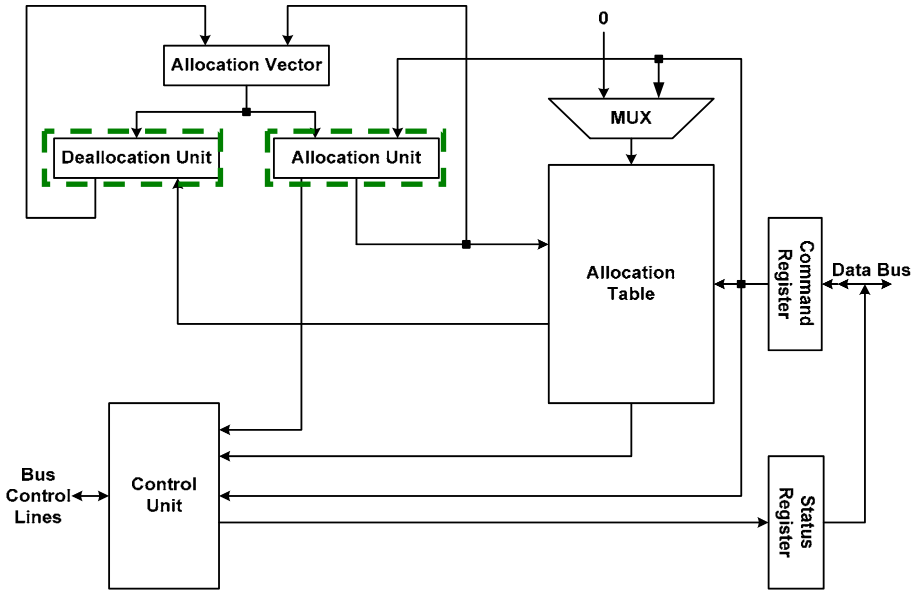


Basic SoCDMMU

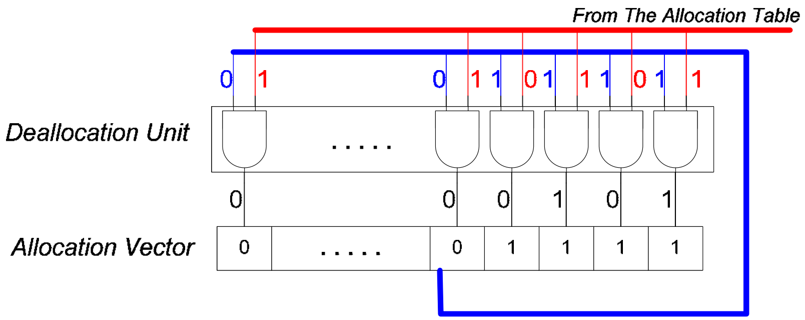


The SoCDMMU Hardware

The Basic SoCDMMU



Basic SoCDMMU



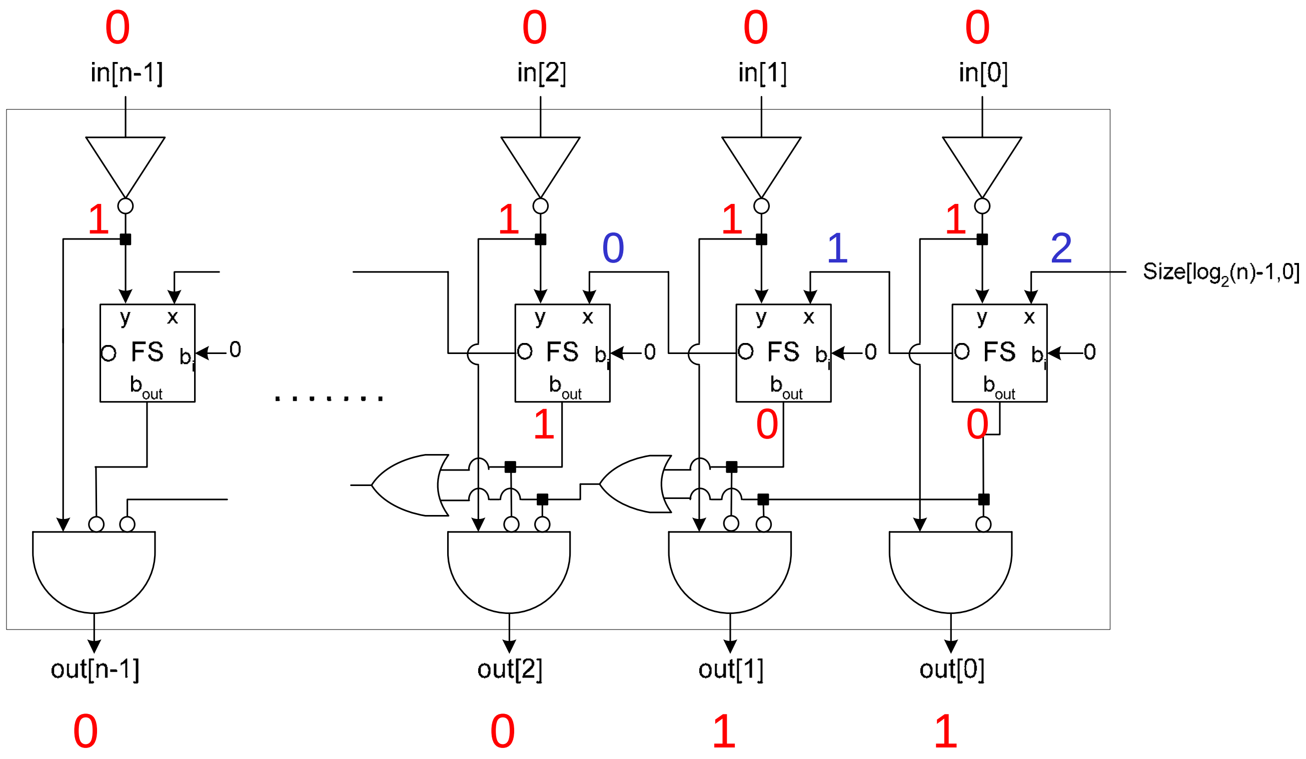
The SoCDMMU Hardware

The *Allocation Unit*

```
allocate(size,in[0:n-1]) {
    for (i:=0 to n-1) {
        if (in[i]==0 and size>0) {
            out[i]:=1;
            size:=size-1;
        } else out[i]:=0;
    }
    if (size>0) return NOT_ENOUGH_MEMORY;
    else return out;
}
```

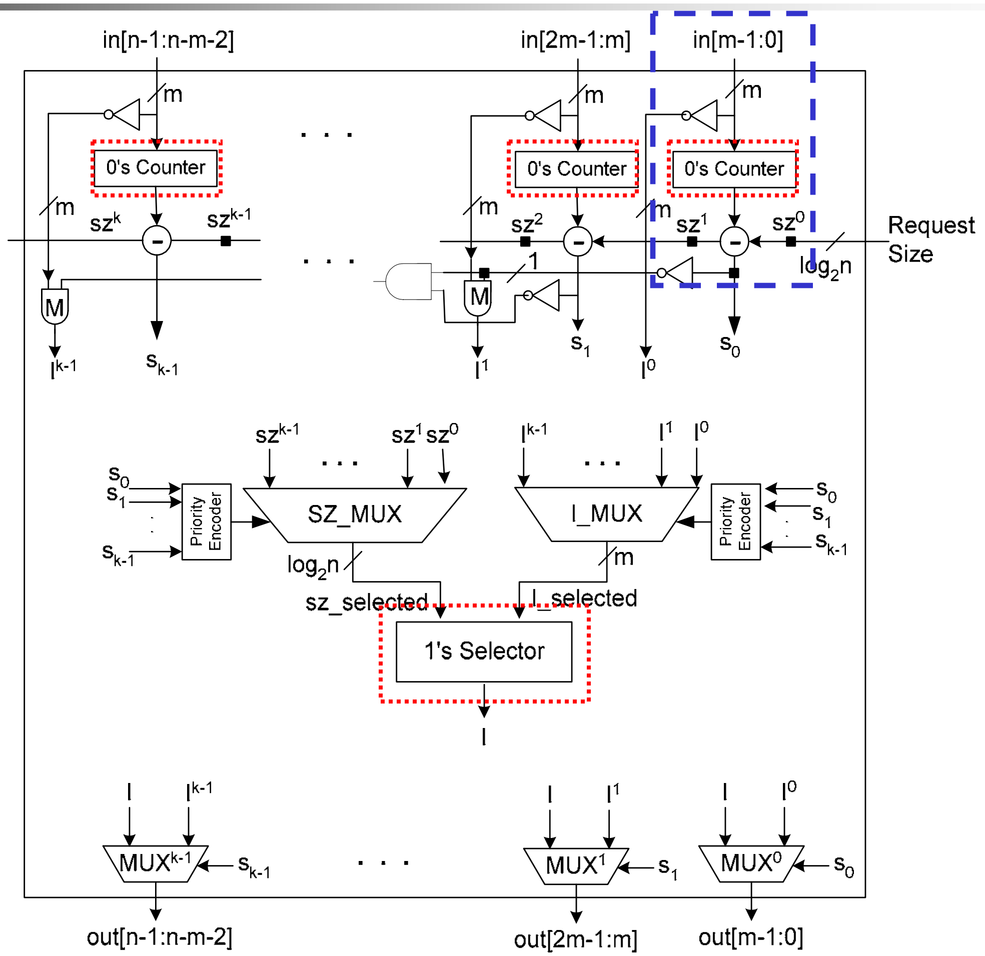
The SoCDMMU Hardware

The *Allocation Unit*



The SoCDMMU Hardware

The *Allocation Unit*



The SoCDMMU Hardware

The *Allocation Unit*

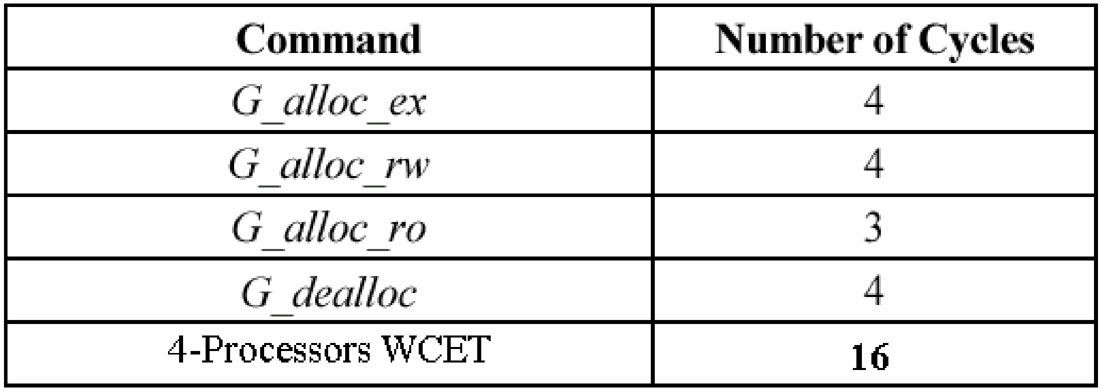
	Area (NAND gates)	**Worst Delay (ns)**	**Max. Clock Speed (MHz)**
Optimized Allocator	5364	6.6 ns	150 MHz
Un-optimized Alocator	17930	56.3 ns	17.5 MHz
Comparison	3.3X	8.5X	

- 256 *G_block*s.
- Synthesized using Synopsys Design Compiler^{TM} 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

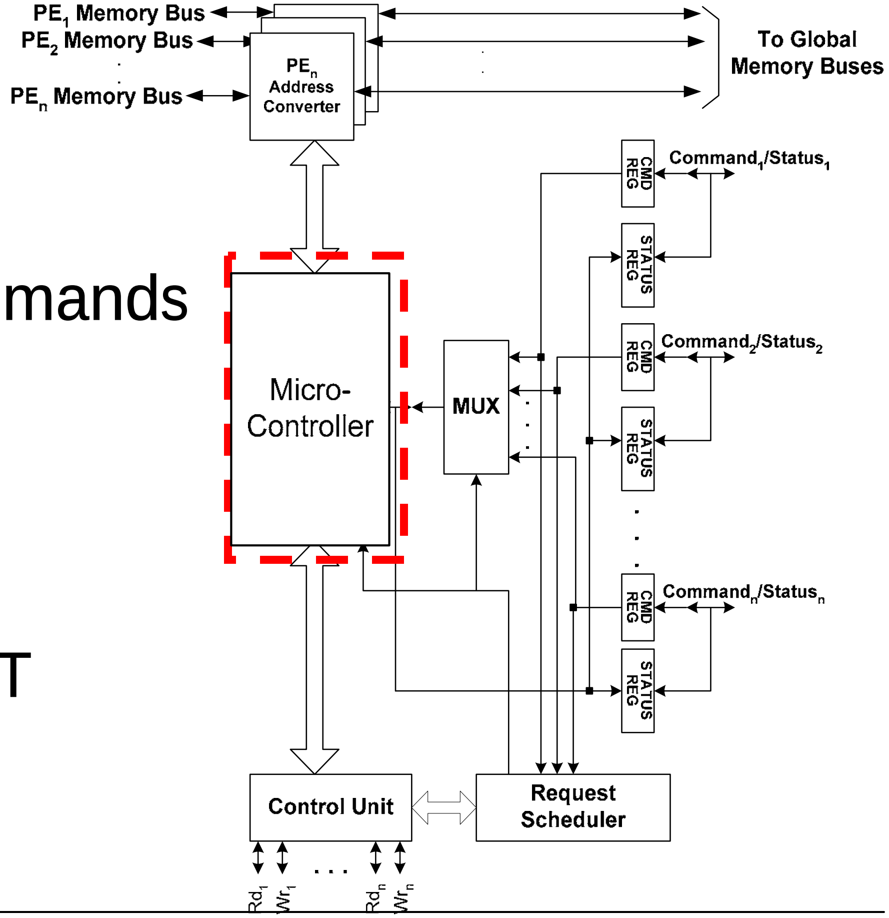


Command	Number of Cycles
G_alloc_ex	4
G_alloc_rw	4
G_alloc_ro	3
G_dealloc	4
4-Processors WCET	**16**

Microcontroller Implementation

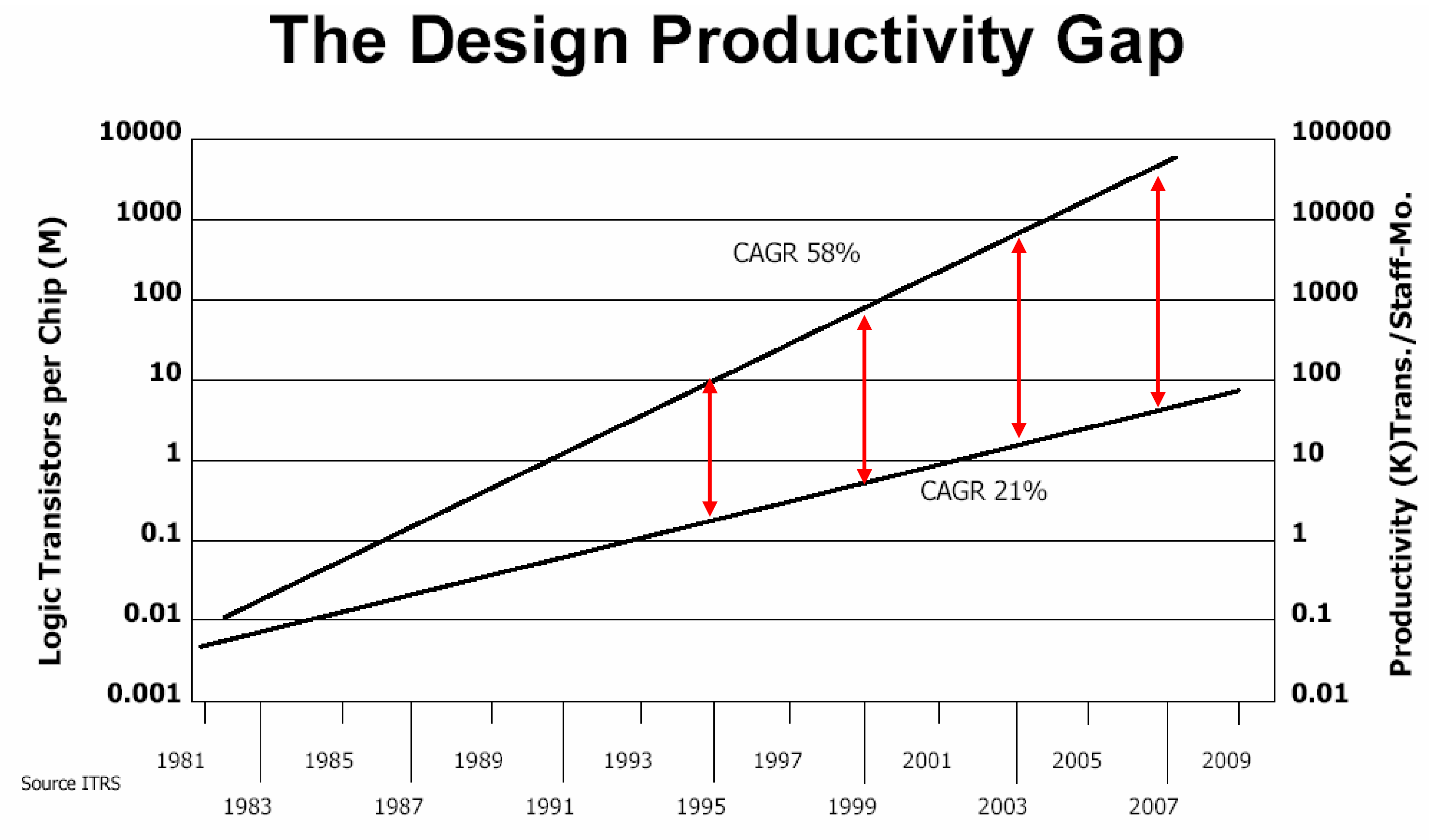
Microcontroller Roles:

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



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

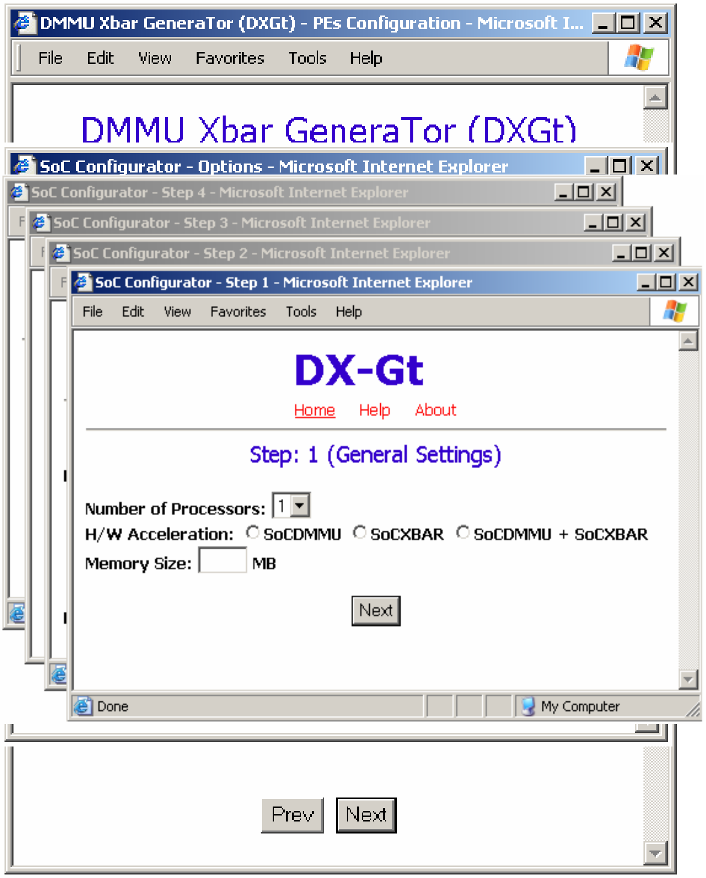
Automatic Generation: Motivation



SoCDMMU IP Generation

- 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, Virage, 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



DMMU Xbar GeneraTor (DXGt) - PEs Configuration - Microsoft I...

DMMU Xbar GeneraTor (DXGt)

SoC Configurator - Options - Microsoft Internet Explorer

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

DX-Gt

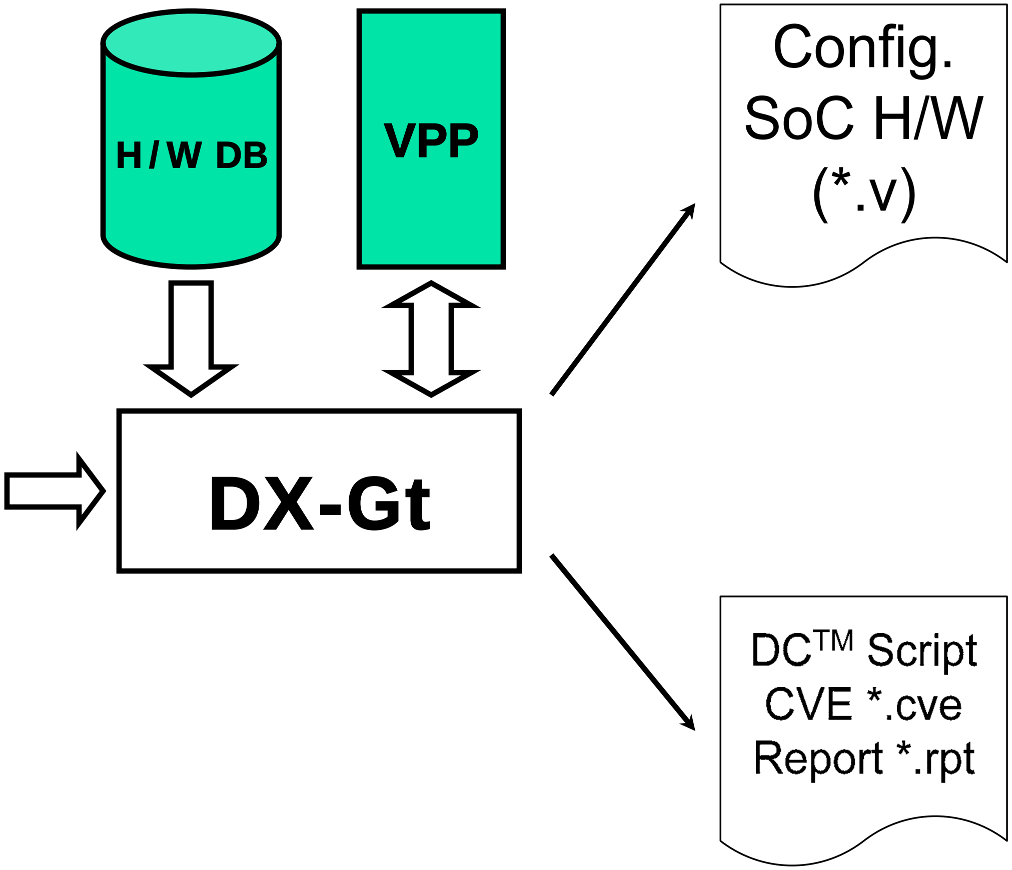
Home Help About

Step: 1 (General Settings)

Number of Processors: 1

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

Memory Size: ____ MB



H/W DB

VPP

DX-Gt

Config. SoC H/W (*.v)

DC^{TM} Script CVE *.cve Report *.rpt

RTOS Support

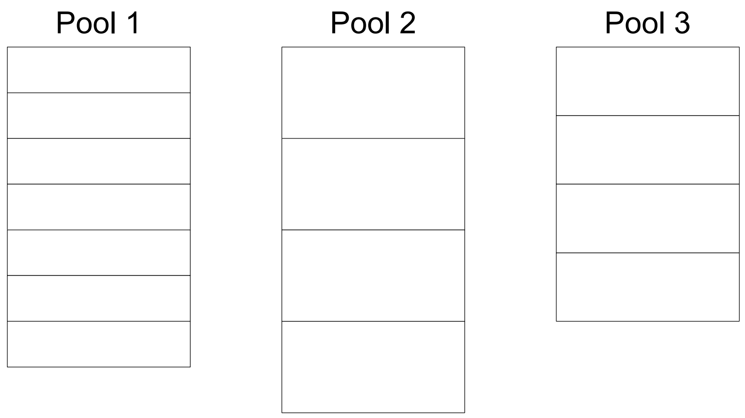
Introduction

- 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 have constant search time.
- An RTOS uses a different approach to make the allocation deterministic

RTOS Support

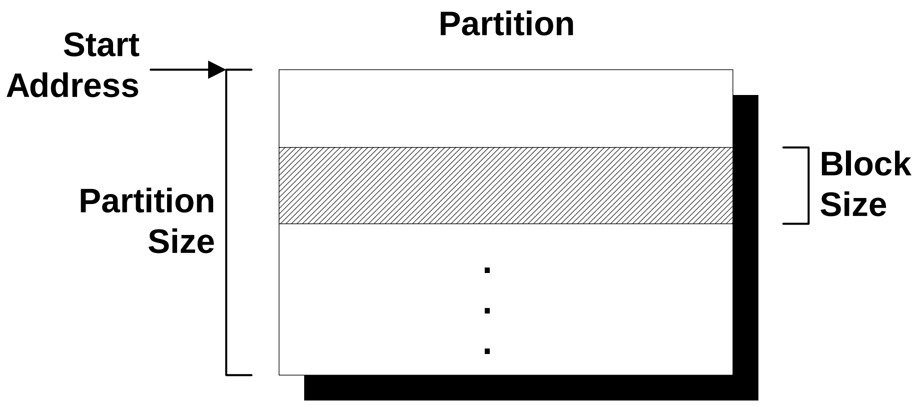
Introduction

- 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

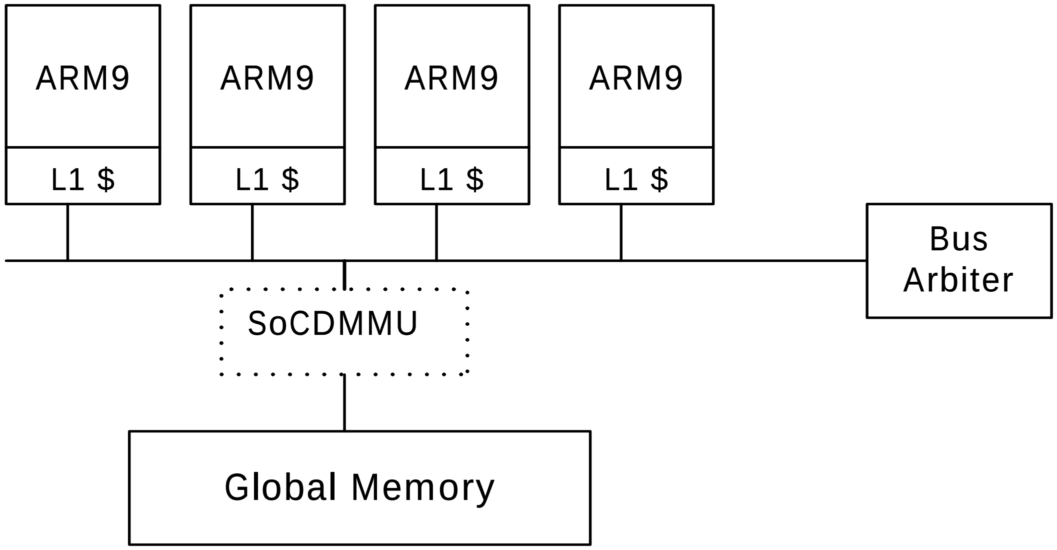
Atalanta Support for the SocDMMU

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.

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

SoCDMMU Conclusion

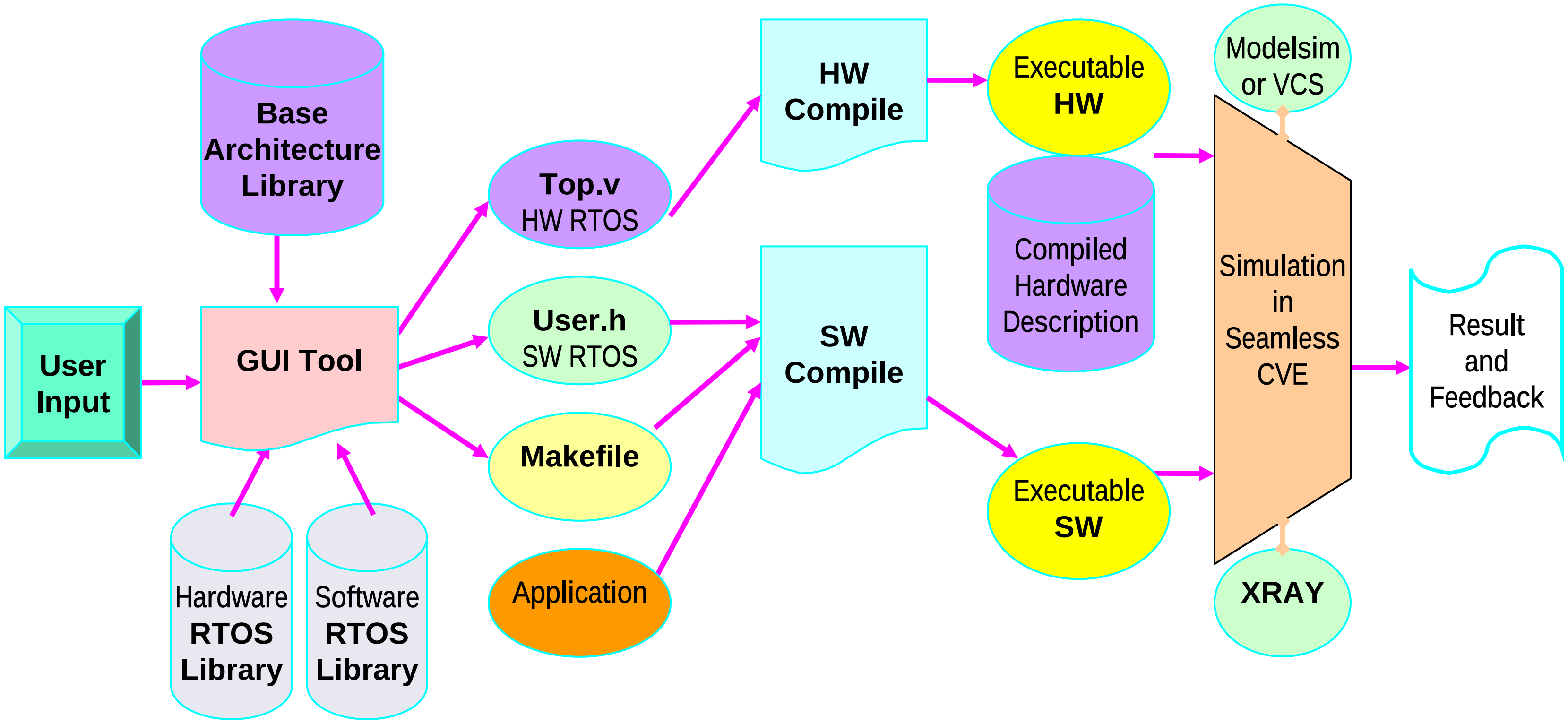
- We introduced The Two-Level memory management hierarchy for multiprocessor SoC
- We showed how Level 1 in the hierarchy can be implemented using the SoCDMMU
- We gave a sample hardware implementation of the SoCDMMU
- We introduced DX-Gt to automatically configure/customize the SoCDMU hardware
- We 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

Outline

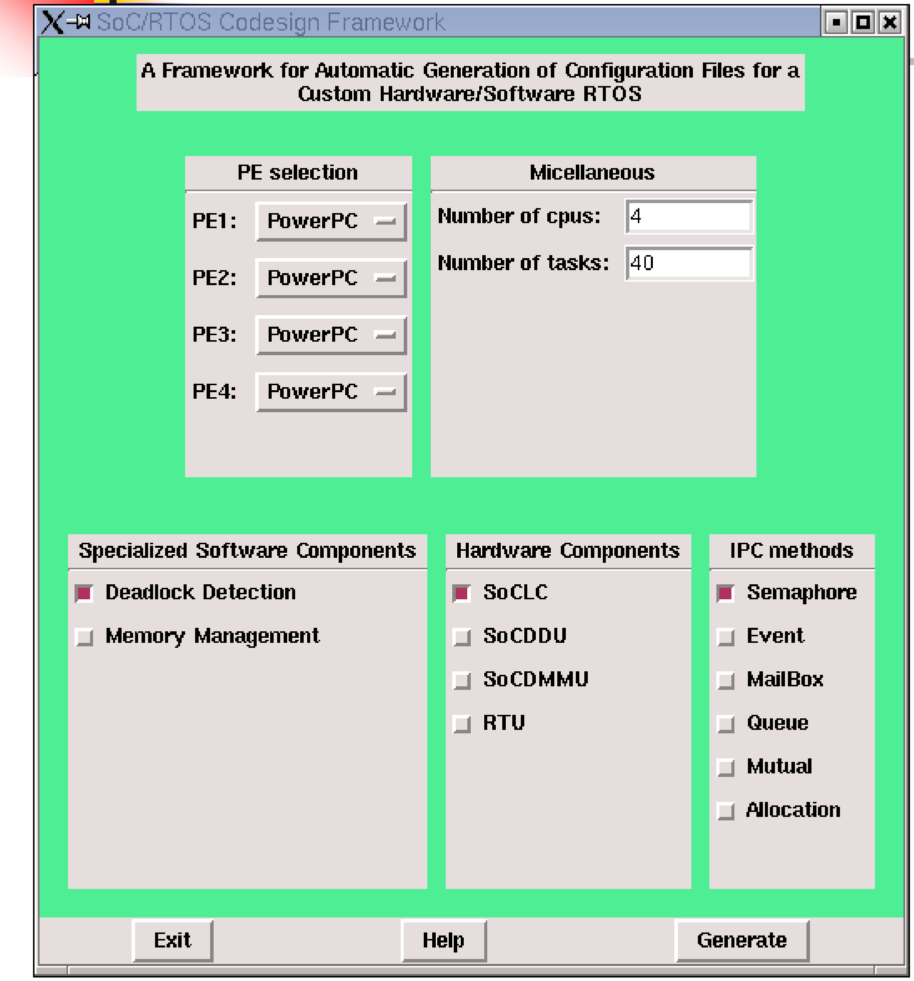
- Trends
- Custom Operating System Hardware IP
  - System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)
  - The δ Hardware/Software RTOS Generation Framework
- Compilation for Reverse Execution & Debugging
- Conclusion

δ Hardware/Software RTOS Generation Framework

and current simulation platform



The δ Hardware/Software RTOS Generation 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

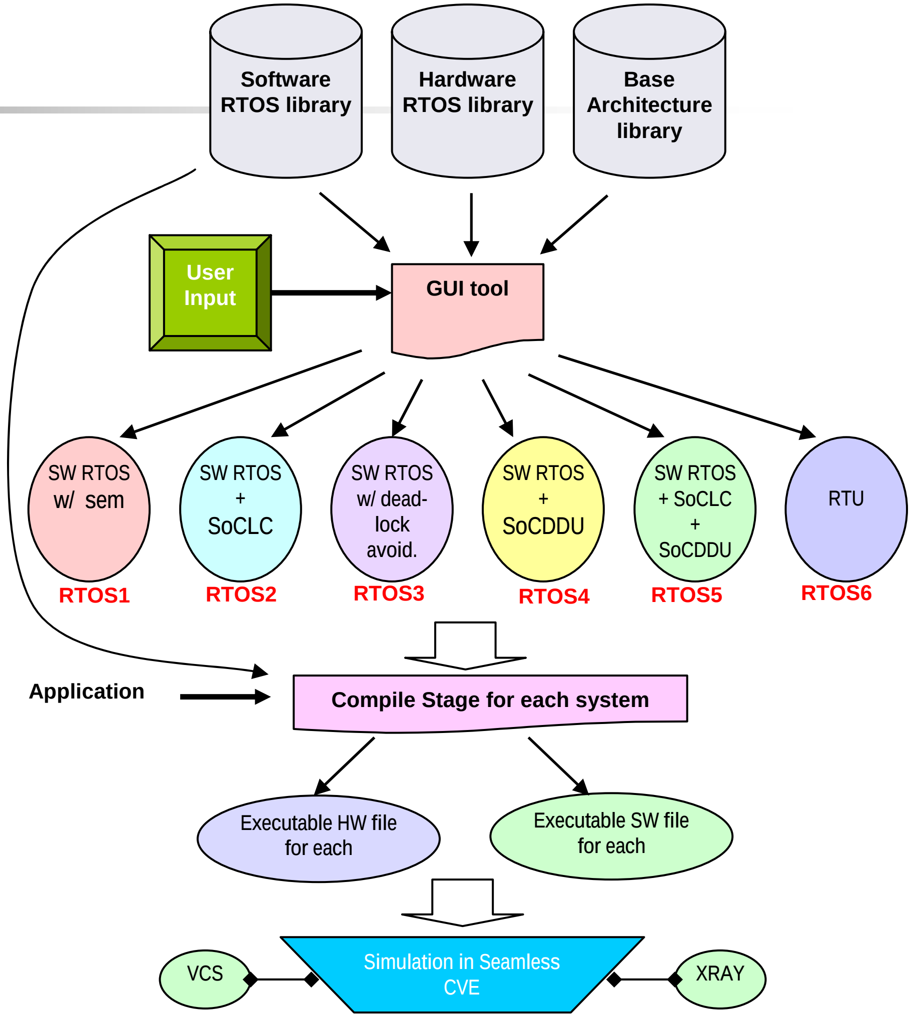
- ☑ Deadlock Detection
- ☐ Memory Management

Hardware Components

- ☑ SoCLC
- ☐ SoCDDU
- ☐ SoCDMMU
- ☐ RTU

IPC methods

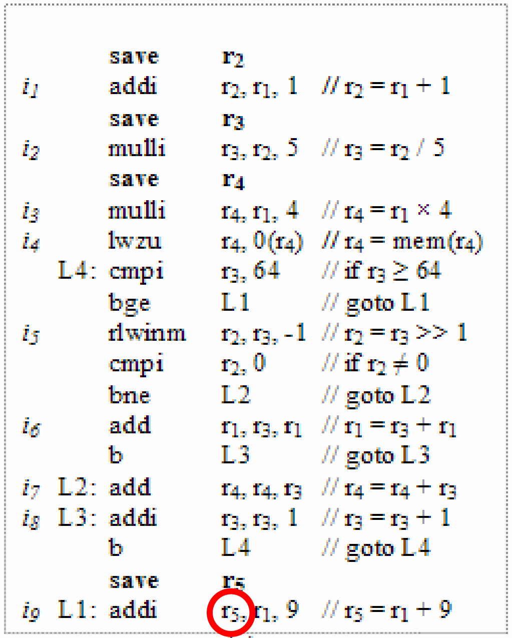
- ☑ Semaphore
- ☐ Event
- ☐ MailBox
- ☐ Queue
- ☐ Mutual
- ☐ Allocation



Outline

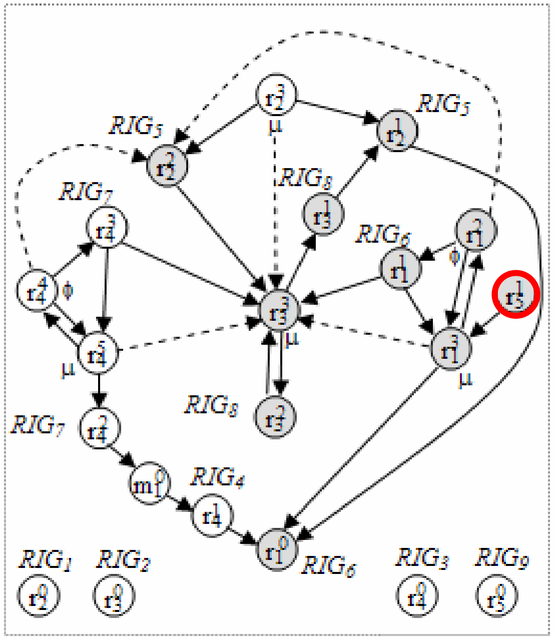
- Trends
- Custom Operating System Hardware IP
  - System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU)
  - The δ Hardware/Software RTOS Generation Framework
- Compilation for Reverse Execution & Debugging
- Conclusion

Generation of a Reverse Program

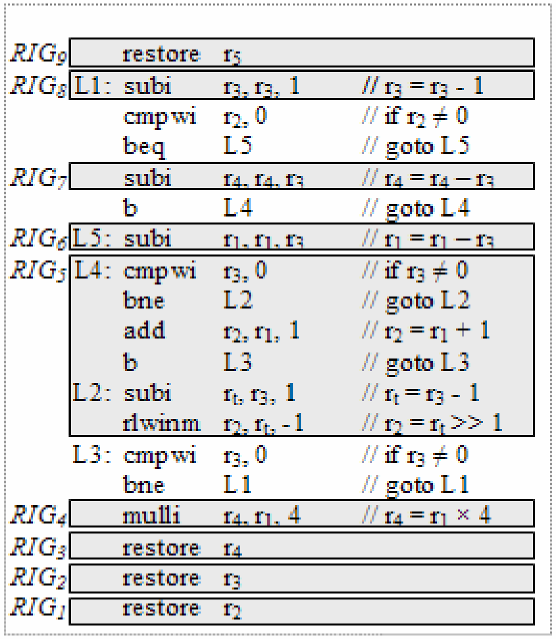


```
        save    r2
i1      addi    r2, r1, 1     // r2 = r1 + 1
        save    r3
i2      mulli   r3, r2, 5     // r3 = r2 / 5
        save    r4
i3      mulli   r4, r1, 4     // r4 = r1 × 4
i4      lwzu    r4, 0(r4)     // r4 = mem(r4)
    L4: cmpi    r3, 64        // if r3 ≥ 64
        bge     L1            // goto L1
i5      rlwinm  r2, r3, -1    // r2 = r3 >> 1
        cmpi    r2, 0         // if r2 ≠ 0
        bne     L2            // goto L2
i6      add     r1, r3, r1    // r1 = r3 + r1
        b       L3            // goto L3
i7  L2: add     r4, r4, r3    // r4 = r4 + r3
i8  L3: addi    r3, r3, 1     // r3 = r3 + 1
        b       L4            // goto L4
        save    r5
i9  L1: addi    r5, r1, 9     // r5 = r1 + 9
```

(a)

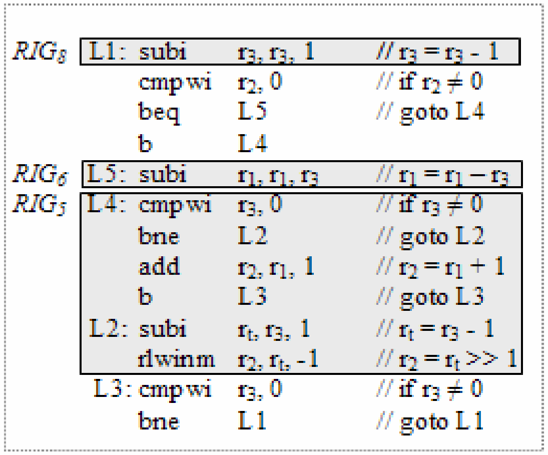


(b)



```
RIG9        restore r5
RIG8  L1:   subi    r3, r3, 1      // r3 = r3 - 1
            cmpwi   r2, 0          // if r2 ≠ 0
            beq     L5             // goto L5
RIG7        subi    r4, r4, r3     // r4 = r4 – r3
            b       L4             // goto L4
RIG6  L5:   subi    r1, r1, r3     // r1 = r1 – r3
RIG5  L4:   cmpwi   r3, 0          // if r3 ≠ 0
            bne     L2             // goto L2
            add     r2, r1, 1      // r2 = r1 + 1
            b       L3             // goto L3
      L2:   subi    rt, r3, 1      // rt = r3 - 1
            rlwinm  r2, rt, -1     // r2 = rt >> 1
      L3:   cmpwi   r3, 0          // if r3 ≠ 0
            bne     L1             // goto L1
RIG4        mulli   r4, r1, 4      // r4 = r1 × 4
RIG3        restore r4
RIG2        restore r3
RIG1        restore r2
```

(c)



```
RIG8  L1:   subi    r3, r3, 1      // r3 = r3 - 1
            cmpwi   r2, 0          // if r2 ≠ 0
            beq     L5             // goto L4
            b       L4
RIG6  L5:   subi    r1, r1, r3     // r1 = r1 – r3
RIG5  L4:   cmpwi   r3, 0          // if r3 ≠ 0
            bne     L2             // goto L2
            add     r2, r1, 1      // r2 = r1 + 1
            b       L3             // goto L3
      L2:   subi    rt, r3, 1      // rt = r3 - 1
            rlwinm  r2, rt, -1     // r2 = rt >> 1
      L3:   cmpwi   r3, 0          // if r3 ≠ 0
            bne     L1             // goto L1
```

(d)

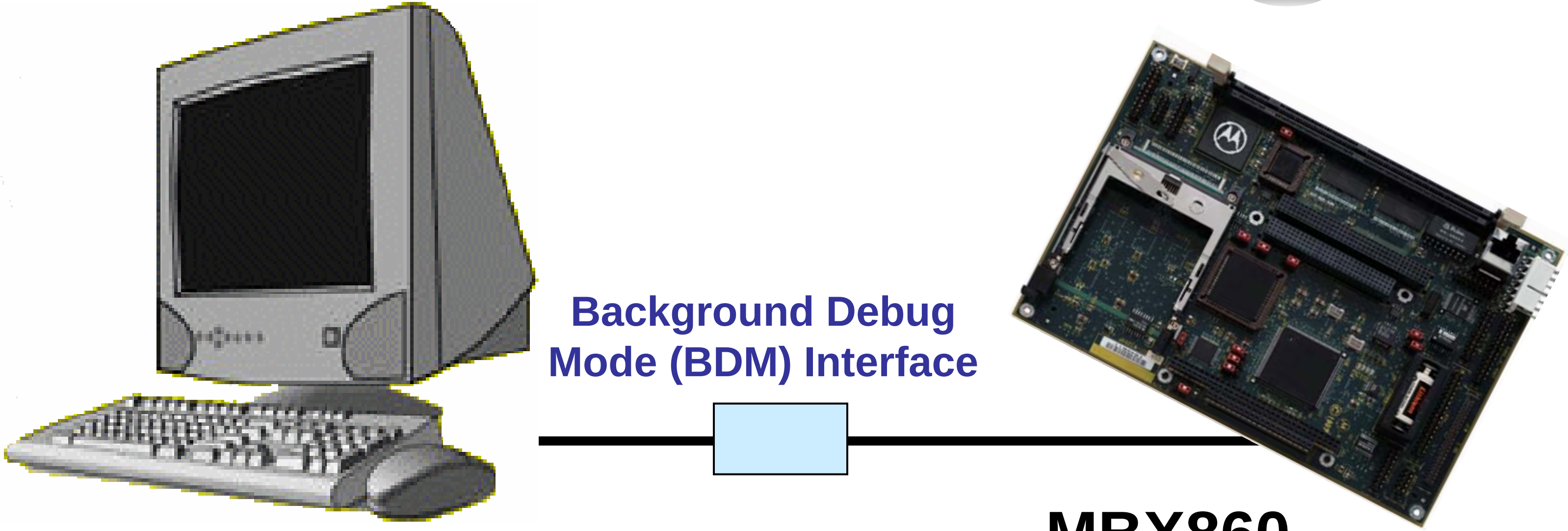
(a): original program
(b): corresponding MVG
(c): reverse program
(d): reduced reverse program

* See thesis by T. Akgul available from http://codesign.ece.gatech.edu

Experimentation Platform

Host

Target



Background Debug Mode (BDM) Interface

PC

Windows 2000

MBX860

MPC860 processor

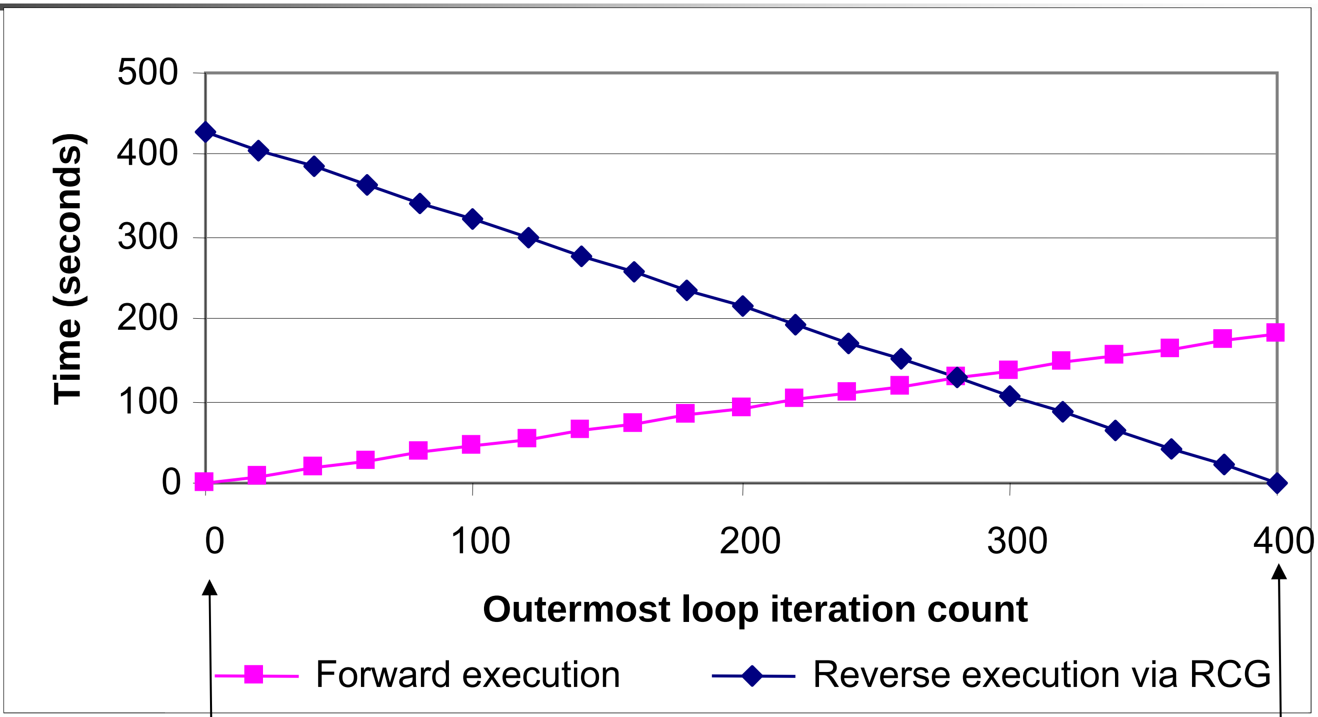
4MB DRAM, 2MB Flash

RTC, four 16-bit timers, watchdog

Memory Usage for State Saving

	ISS (KB)	ISSDI (KB)	RCG (KB)	ISS / RCG	ISSDI / RCG
Selection sort (100 inputs)	68.2	46.9	7.5	9X	6.3X
Selection sort (1000 inputs)	6032	4065	151	40X	27X
Selection sort (10000 inputs)	593389	397913	7237	82X	55X
Matrix multiply (4x4)	3.6	2.35	0.17	21X	14X
Matrix multiply (40x40)	2820	1801	12.6	224X	143X
Matrix multiply (400x400)	2756883	1755006	1250	2206X	1404X
ADPCM (32KB input data)	1544	1192	616	2.5X	2X
ADPCM (64KB input data)	3088	2384	1232	2.5X	2X
ADPCM (128KB input data)	6175	4767	2464	2.5X	2X
LZW (1KB input data)	5630	3425	98.4	57X	35X
LZW (4KB input data)	64970	39163	351	185X	112X
LZW (16KB input data)	784336	471140	1331	589X	354X

Program Re-execute Approach vs. RCG



400x400 matrix multiply

starting point for forward execution

starting point for reverse execution

Conclusion

- Trends
- SoCDMMU
- Reverse Program Generation
- Have fun at FPGAworld 2007!