A Dynamic Memory Management Unit For Embedded Real-Time System-on-a-Chip

Mohamed Shalan

Vincent Mooney

School of Electrical and Computer Engineering

Georgia Institute of Technology

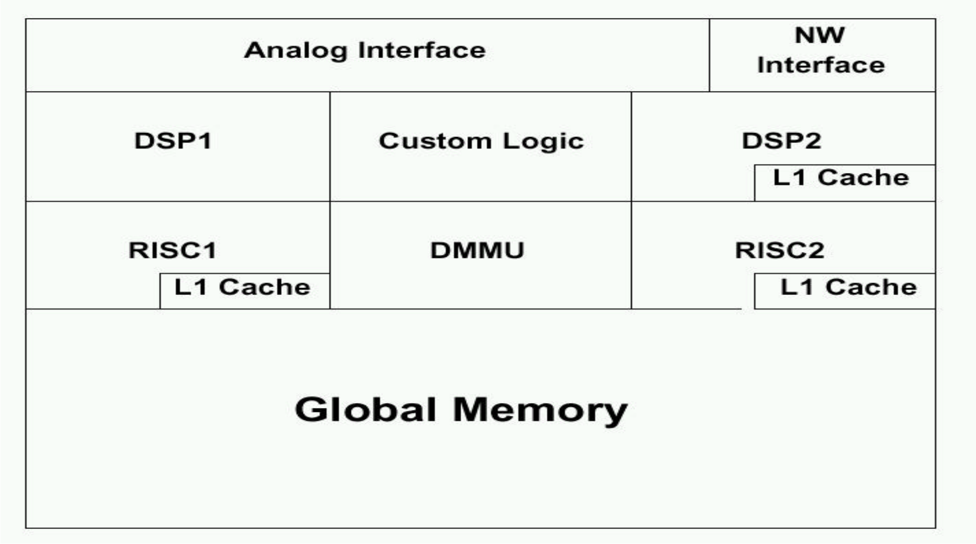
Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

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 PE’s of various types, large global on-chip memory, analog components, and network interfaces.

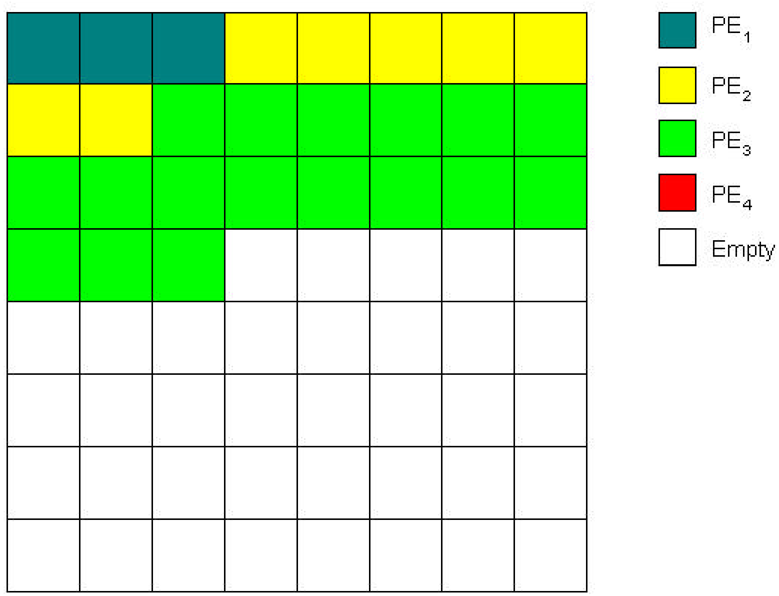
System-on-a-Chip (SoC)



- 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 PE's.



Problem

- How to deal with the allocation of the large global on-chip memory between the PE's. ?

Solution 1

- Custom Memory Configuration (Static)
  - Pros:
    - Easy.
    - Deterministic.
  - Cons:
    - Inefficient memory utilization.
    - System modification after implementation is very difficult if not impossible.

Solution 2

- Shared memory multiprocessor (Dynamic)
  - Pros
    - Flexible.
    - Efficient memory utilization.
  - Cons
    - Worst case execution time is very high if not not deterministic.

SoCDMMU

- The SoC Dynamic Memory Management Unit (SoCDMMU) is a Hardware Unit, to be a part of the SoC, that deals with the memory allocation/de-allocation among the PE's.
- The SoCDMMU allows a fast and deterministic dynamic way to allocate/de-allocate the Global Memory among the PE's.

Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

Programming Model

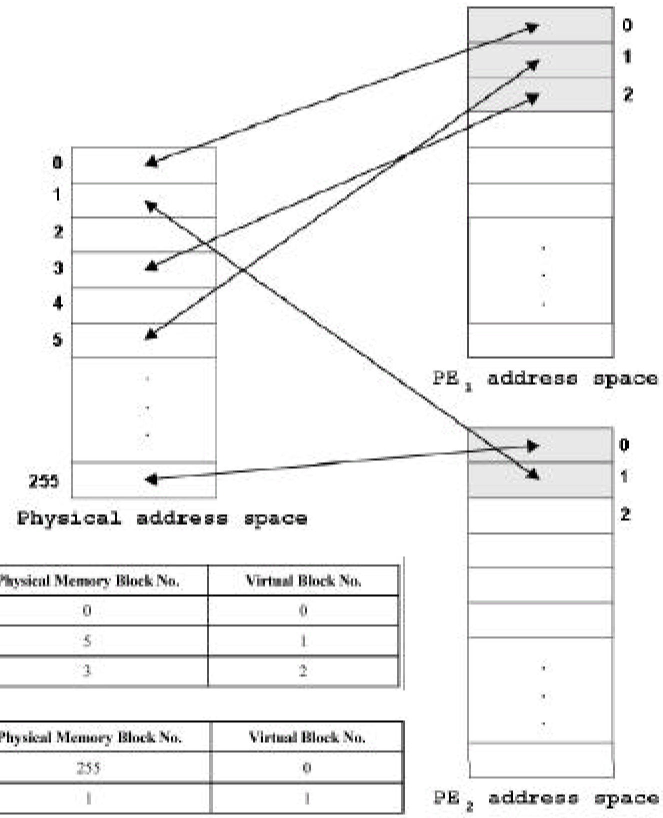
- Assumptions.
- Two-Level memory management.
- Types of allocations.

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_de-allocation.*
- The PE can *G_allocate* one or more than one block.
- Different PE's 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 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

- There is an OS that runs on each PE.
- The SoCDMMU manages the memory between the PE's.
- The OS on each PE manages the memory between the processes that run on that PE (Level 1).
- The process requests the memory allocation from the OS. If there in not enough memory, the OS requests memory allocation from the SoCDMMU (Level 2).

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.

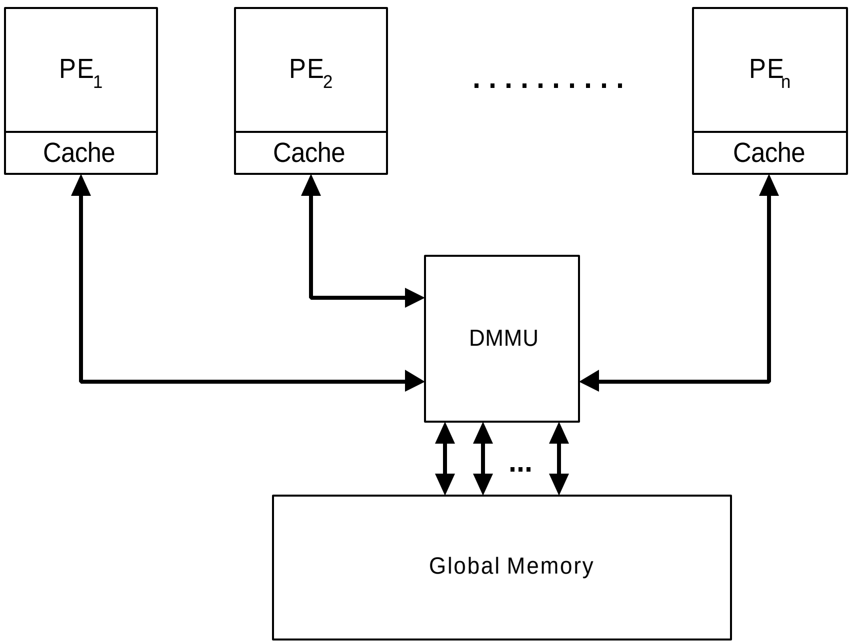
Outline

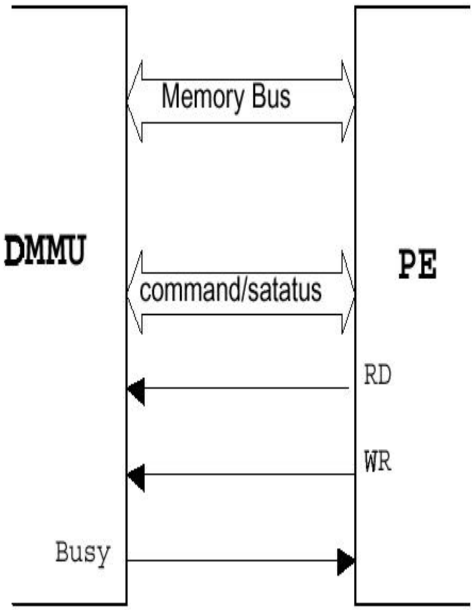
- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

The SoCDMMU Hardware

- PE-SoCDMMU Interface.
- PE-SoCDMMU Commands.
- SoCDMMU Architecture
  - Basic SoCDMMU.
  - Address Converter.

PE-SoCDMMU Interface





SoCDMMU Commands

n/a	Size	Virtual Block no.	000

G_alloc_ex

SW ID	Size	Virtual Block no.	001

G_alloc_rw

SW ID	n/a	Virtual Block no.	010

G_alloc_ro

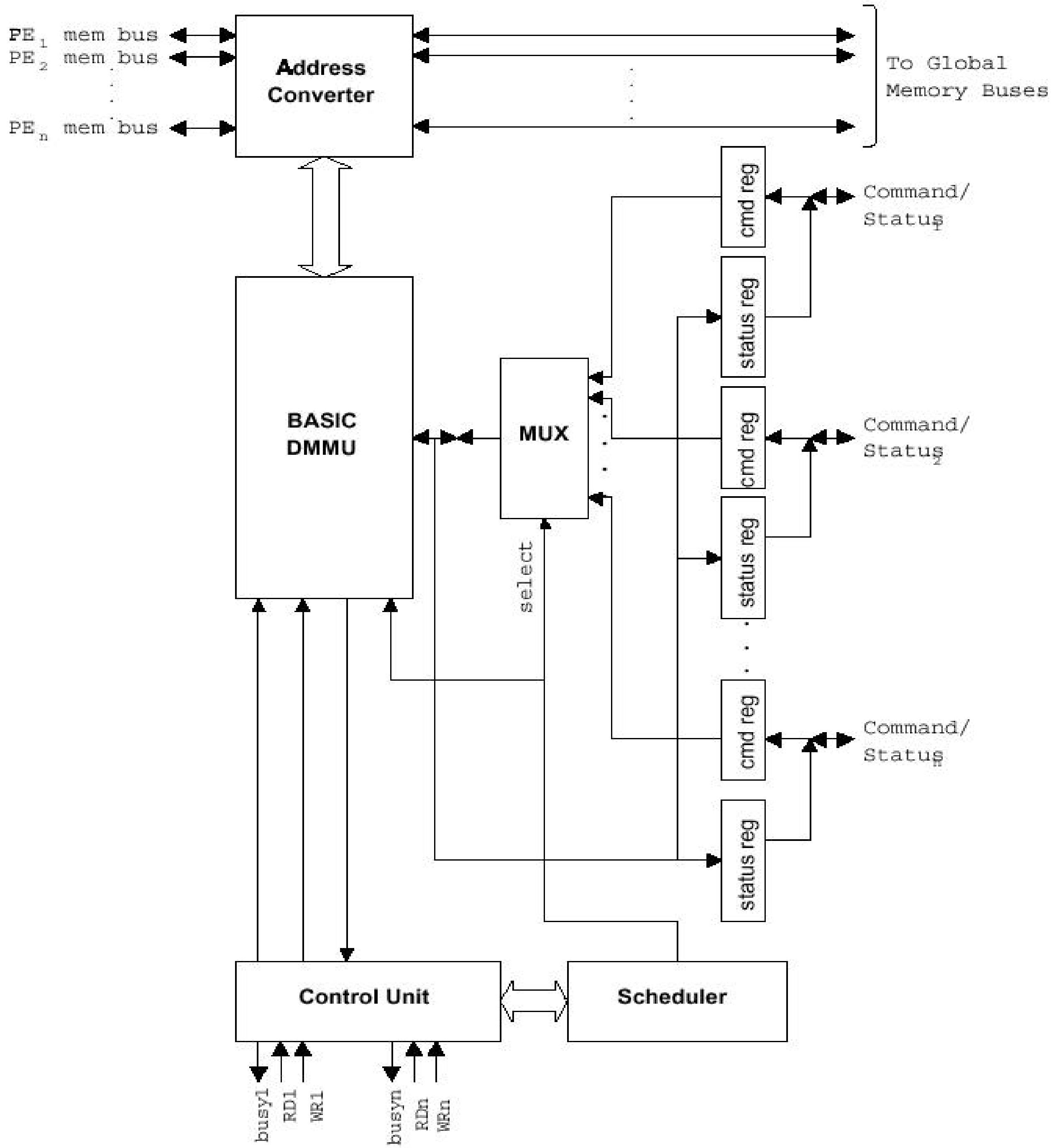
n/a	n/a	Virtual Block no.	011

G_de-alloc

New Virtual Block no.	Old Virtual Block no.	100

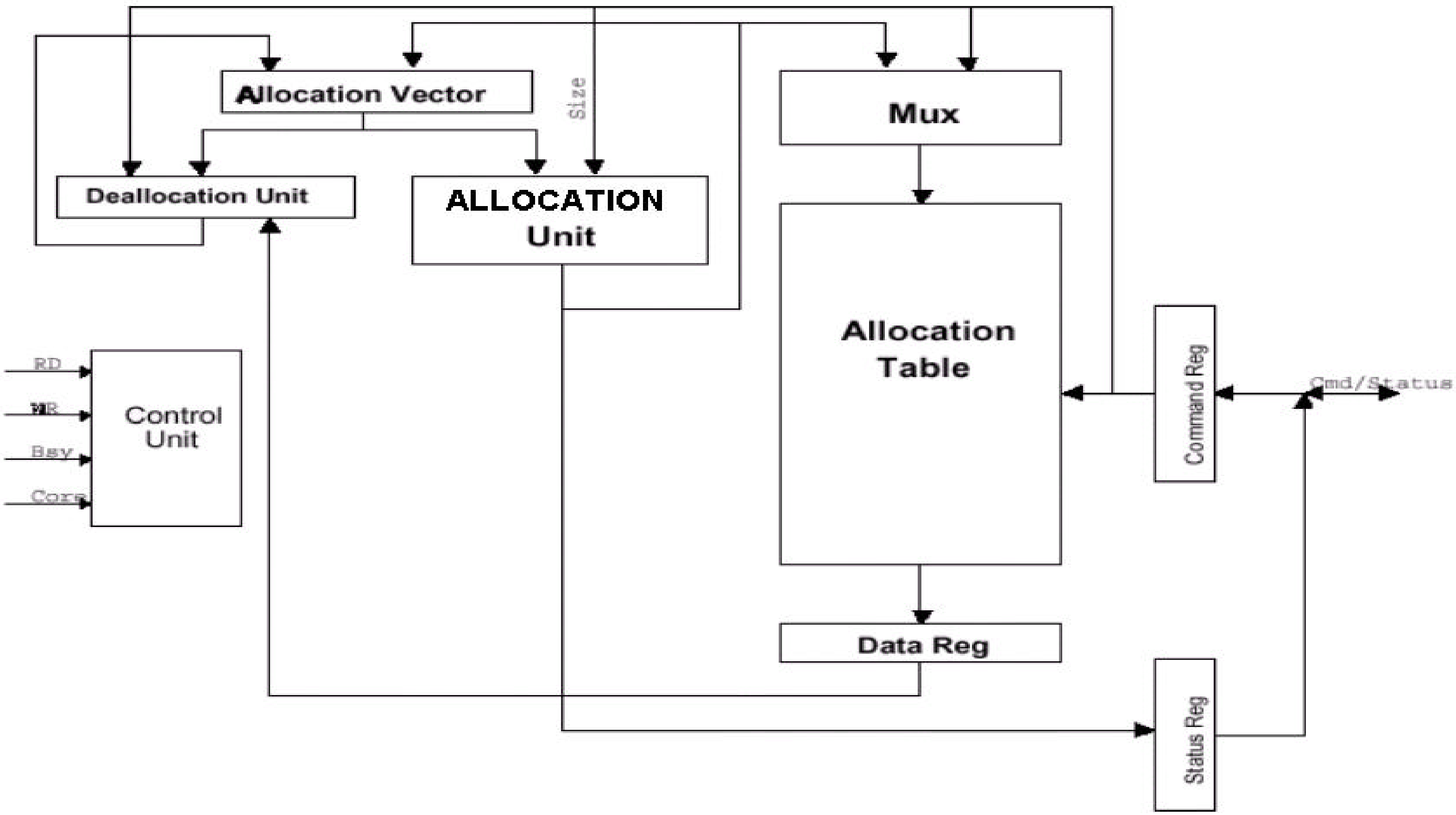
Move

The SoCDMMU Architecture

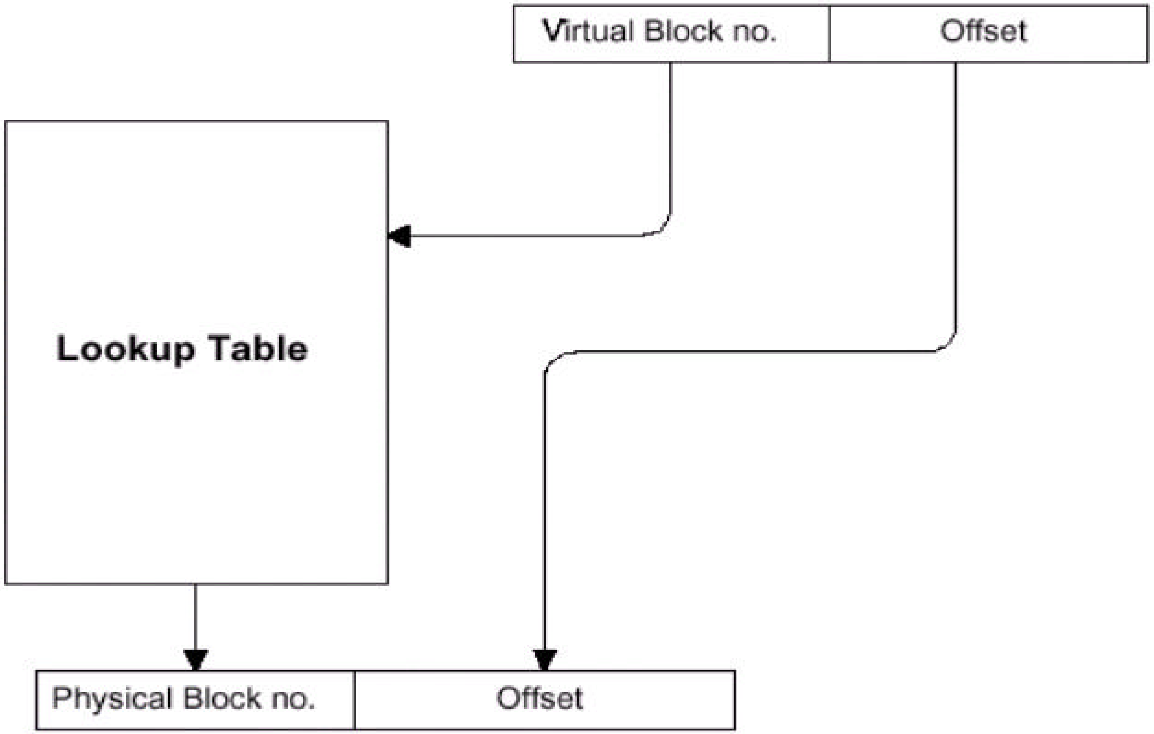


PE 1 mem bus
PE 2 mem bus
PE n mem bus
Address Converter
To Global Memory Buses
cmd reg
status reg
Command/ Status 1
Command/ Status 2
Command/ Status n
BASIC DMMU
MUX
select
Control Unit
Scheduler
busy1
RD1
WR1
busyn
RDn
WRn

Basic SoCDMMU



Address Converter



Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

Experiments and Results

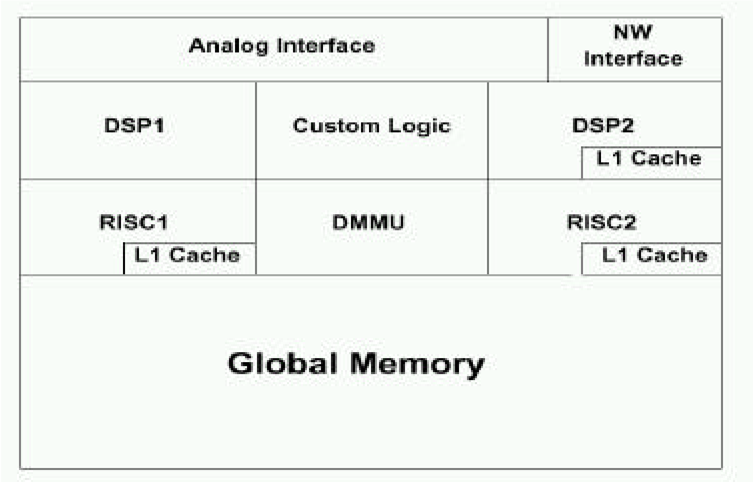
- SoCDMMU Synthesis.
- SoCDMMU Execution Times.
- Comparison with uC implementation

Synthesis

The SoCDMMU was modeled using Verilog at the *RTL* level. It was successfully synthesized using SYNOPSYS™ Design Compiler. By using AMI 0.5 micron library we got the following results.

Lines (Verilog)	*Area*
1,030 Lines	41,561.5

Execution Times



- Wireless application with voice interface.
- Global Memory 16MB.
- Allocation Block Size is 64KB.
- Allocation Vector is 256 bit
- Allocation Table has 256 entries.

Execution Times

Command	Number of Cycles
G_alloc_ex	4
G_alloc_rw	4
G_alloc_ro	3
G_dealloc	5
4-Processors WCET	20

SoCDMMU vs. uC Implementation

- To demonstrate the importance of building the SoCDMMU as a custom logic, we implemented the same functionality in software runs on PIC uC.
- Both of the custom SoCDMMU and the uC Implementation ran at 100Mhz.
- The uC code was developed using MPASM.
- The uC software is about 500 lines.

DMMU *Worst-Case Execution Time*	20 Cycles
Microcontroller Best-Case Execution Time	221 Cycles

Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

RTOS Support

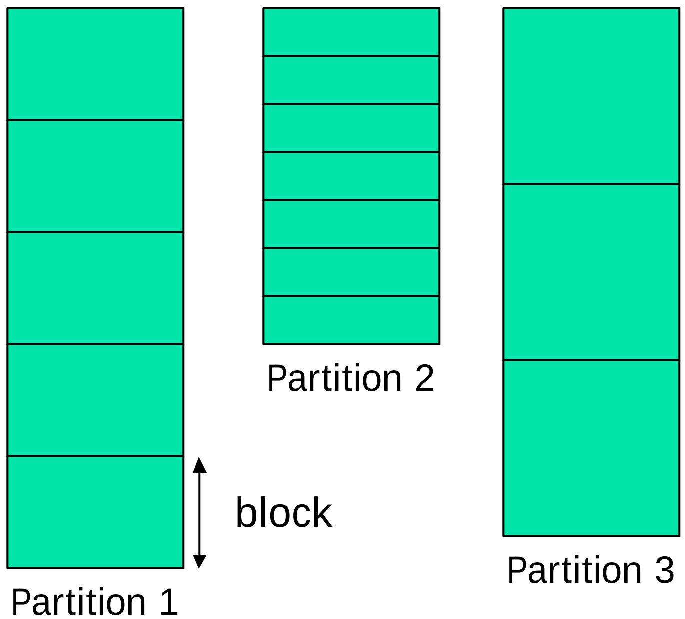
- Introduction.
- uC/OS II Memory Management.
  - Overview.
  - API Functions.
  - Data Structures.
  - Example.
- uC/OS II Support for the SocDMMU

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.
- An RTOS uses a different approach to make the allocation deterministic.
- An RTOS usually divides the memory into fixed-sized allocation units and any task can allocate only one unit at a time.

uC/OS II Memory Management

Overview



- uC/OS II 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.

uC/OS II Memory Management

API Functions

- **`OSMemCreate`**
  - Is used to create a partition.
  - It needs a pointer to a contiguous Memory partition (static array).
  - On success, it returns pointer to the allocated memory control block.
- **`OSMemGet`**
  - Is used to obtain memory block from a partition.
- **`OsMemPut`**
  - Return back a memory block to its partition.

uC/OS II Memory Management

DATA Structures

- The free blocks in each memory partition are linked together as a linked list.
- Each partition has a Memory Control Block (OS_MEM) that stores:
  - Partition base address.
  - Pointer to the free list.
  - No. of free blocks in the partition.
  - Block size of this partition.

uC/OS II Memory Management

Example

```
OS_MEM *Buf;
Unsigned char Part[100][32];
.
.
void main(void)
{
    INT8U err;
    .
    Buf=OSMemCreate(Part,100,32,&err);
    .
}

Void Task1()
{
    INT8U *x, err;
    .
    x=OSMemeGet(Buf, &err);
    .
    OSMemPut(Buf,x);
    .
}
```

uC/OS II Support for the SocDMMU

Objectives

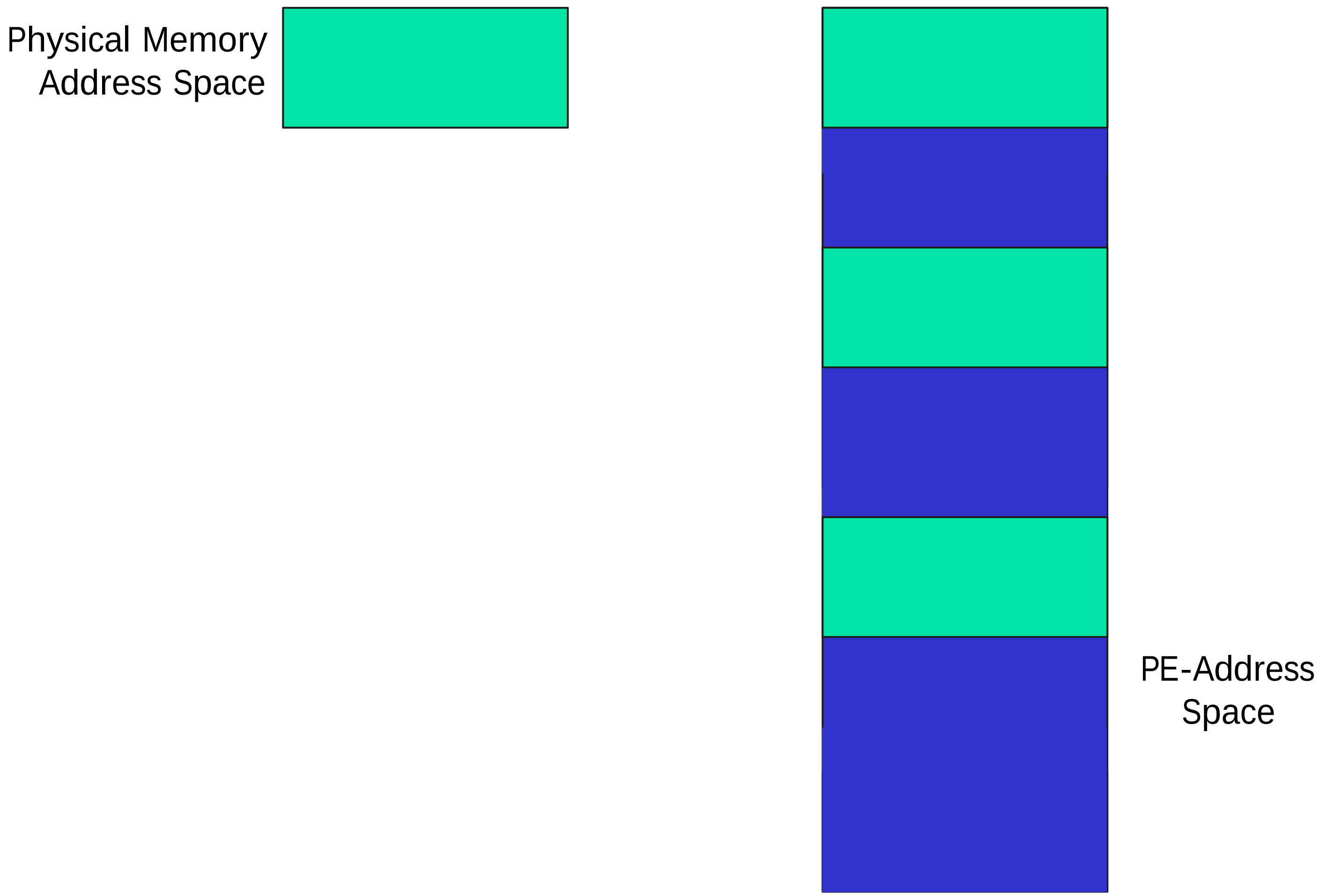
- Add Dynamic Memory Management to uC/OS II.
- Use the same Memory Management API Functions.
- Keep the Memory Management Deterministic.

uC/OS II Support for the SocDMMU

- 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 (VA) Fragmentation can be overcome by:
  - Using the SoCDMUU “Move” Command.
  - Replicate the physical address space.

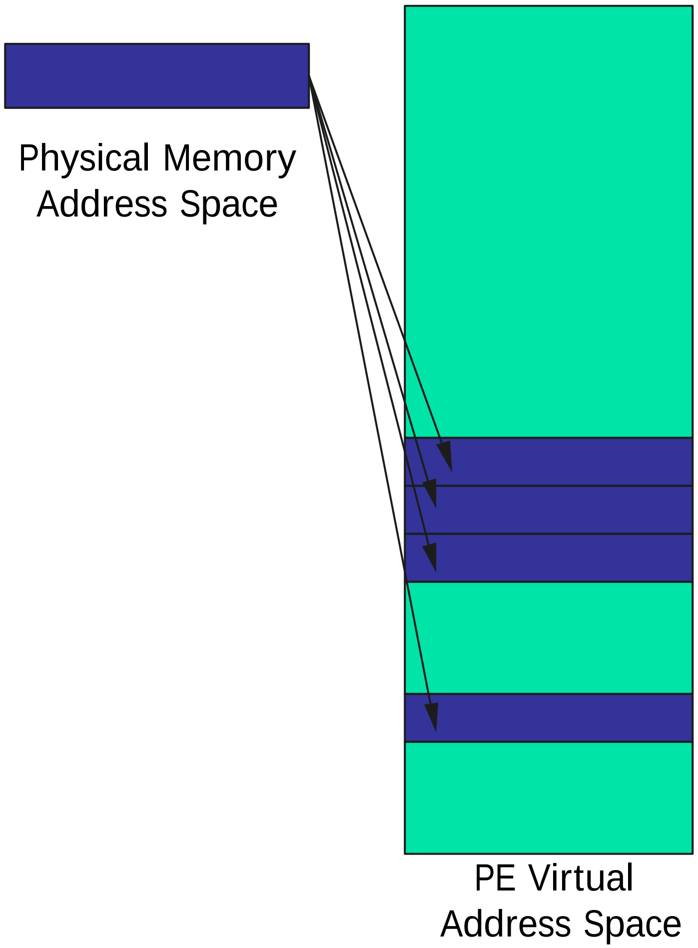
uC/OS II Support for the SocDMMU

Physical Address Space Replication (1)



uC/OS II Support for the SocDMMU

Physical Address Space Replication (2)



- This mirroring is useful to overcome the memory fragmentation.
- The first copy may be used to allocate only one block, the 2nd for allocating 2 contiguous blocks, etc..
- Also another copy may be used as a heap for different sizes allocation other than the above contiguous sizes.
- This heap can be compacted using the SoCDMMU “MOVE” command.

uC/OS II Support for the SocDMMU

New DATA Structures

- Free Blocks Array

  Array of linked list. Each linked list stores the free memory blocks (e.g., for the 2nd mirror the linked list stores the free memory chunks [of 2 blocks]).

- SoCDMMU Memory Control Table
  - Has an entry for each memory allocation done by the SoCDMMU.
  - Each entry has 2 fields
    - Starting VA.
    - Size (no. of blocks).
    - Allocation Type.
    - Pointer to the next allocation of the same type.

uC/OS II Support for the SocDMMU

New API Functions (Level 2)

- **`DMMUMemFind(size)`**
  - Returns pointer to a location in the VA Space (PE-Address Space).
- **`DMMUMemRelease(pointer to an SoCDMMU Memory Control Block entry)`**
- **`DMMUMemGet(size, VA, mode,sw id)`**
  - Returns pointer to an entry in the SoCDMMU Memory Control Block.
- **`DMMUMemPut(pointer to SoCDMMU Memory Control Block entry)`**

uC/OS II Support for the SocDMMU

New API Functions

- **`OSMemRelease`**
  - It does the opposite of the **`OSMemCreate`** function.
  - It may call the **`DMMUMemPut`** to de_allocate the physical memory blocks allocated by **`OSMemCreate`**.

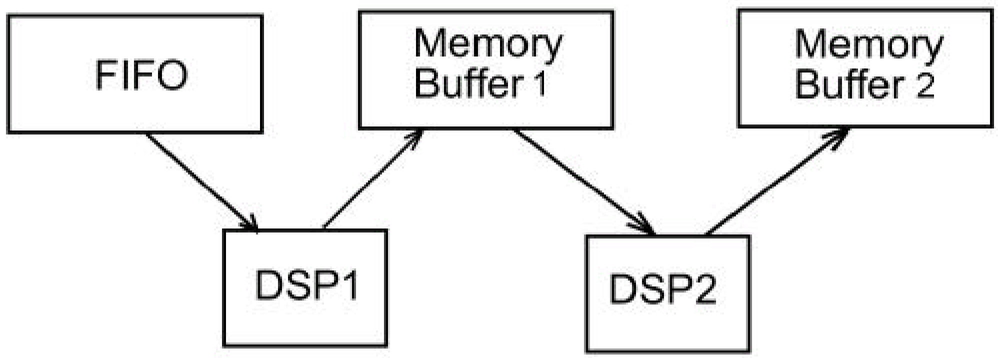
uC/OS II Support for the SocDMMU

Modified API Functions

- **`OSMemCreate(no. of blocks,block size ,mode,SW_id)`**
  - No need for static allocation.
  - It may call the **`DMMUMemGet`** function to allocate no of physical memory blocks.

uC/OS II Support for the SocDMMU

Example (1)



- DSP1 and DSP2 are used to perform the Orthogonal Frequency Division Multiplexing (OFDM).
- DSP1 reads the incoming data from the FIFO and performs FFT, then it passes it to DSP2 through the shared memory buffer 1.
- DSP2 performs the rest of the OFDM processing and then writes the modulated data into memory buffer 2.

uC/OS II Support for the SocDMMU

Example (1)

DSP1

```
#define BUF1 10
OS_MEM *Buf;
INT8U *x;
.
.
buf=OSMemCreate(1024,1,BUF1,RW);
x=OSMemGet(buf);
```

DSP2

```
#define BUF1 10
OS_MEM *buf1,*buf2;
INT8U *x,*y;
.
.
buf1=OSMemCreate(1024,1,BUF1,RO);
x=OSMemGet(buf1);
buf2=OSMemCreate(1024,1,BUF1,EX);
y=OSMemGet(buf2);
```

Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

Current Work

- Extend the SoCDMMU to support *G_alloc_rw* of the same block by multiple PE's.
  - The SoCDMMU may configure the level1 caches to un-cache certain address spaces.
- Carrying out a study comparing our multiprocessor SoC to a SoCDMMU with fully shared memory multiprocessor SoC (e.g., Hydra).
  - Seamless co-simulation of 4 ARM9TDMI cores.
  - ARM AMBA? No
  - New bus agent, bus arbiter, cache coherency controller, and snooping controller? Yes

Outline

- Introduction.
- Programming Model.
- The SoCDMMU HW.
- Experiments and Results.
- RTOS Support.
- Current Work.
- Conclusion.

Conclusion

- We Described a new approach to handle on-chip memory allocation/de-allocation among PE's on SoC. Also, we showed how to extend the ucos-ii to support the SoCDMMU.
- Our approach is based on HW SoCDMMU that allows a dynamic, fast way to allocate/de-allocate the on-chip memory.

Conclusion

- Thus, this approach fits in the gap between general-purpose fully shared memory multiprocessor SoCs and application specific SoC designs with custom memory configurations.

Acknowledgement

- We would like to acknowledge software donations from Mentor Graphics and Synopsys as well as hardware donations from Sun and Intel.

Questions