Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management

Mohamed Shalan
Vincent Mooney

School of Electrical and Computer Engineering
Georgia Institute of Technology

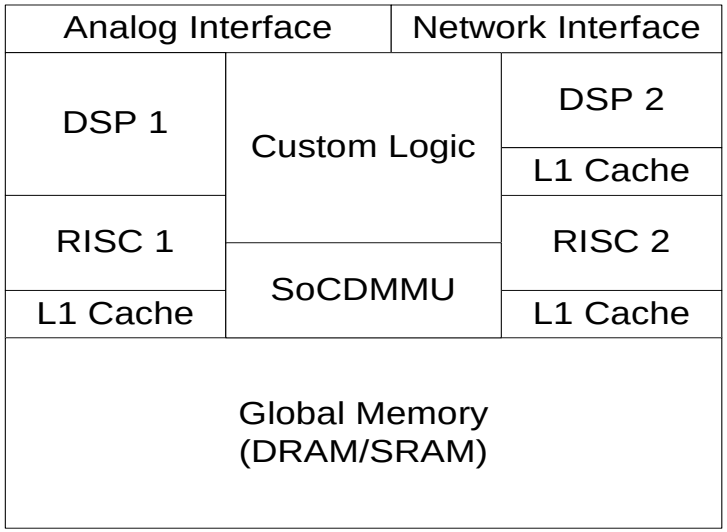
Outline

- Introduction.
- The SoCDMMU.
- RTOS Support.
- Comparison to a Fully Shared-Memory Multiprocessor System.
- 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.

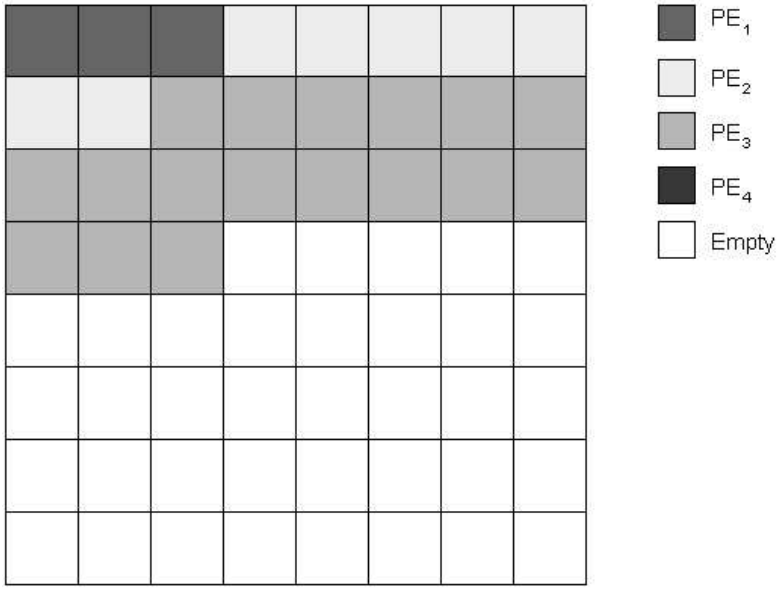
System-on-a-Chip (SoC)



- 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 (example: 16MB of global memory broken into 256 blocks each of size 64KB)



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.

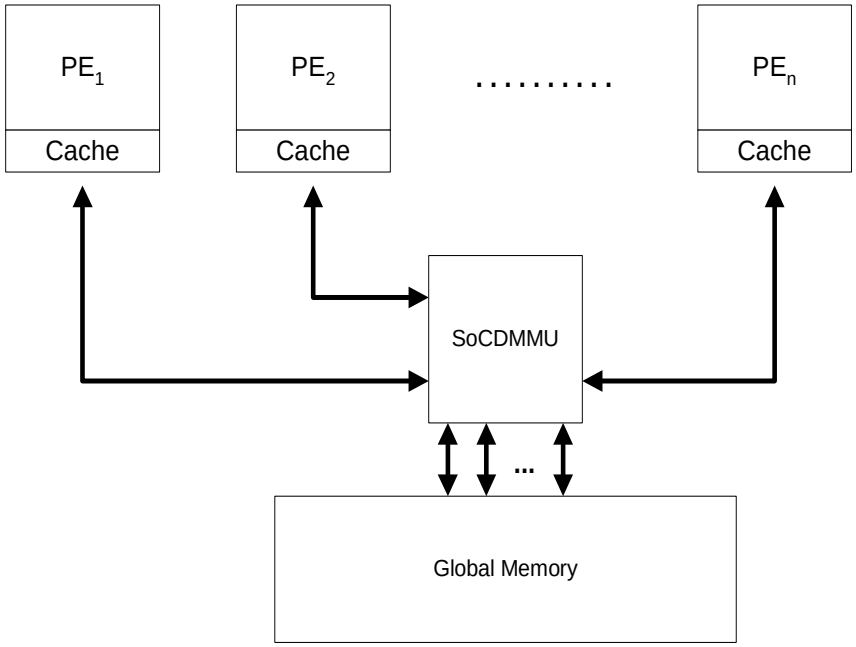
SoCDMMU

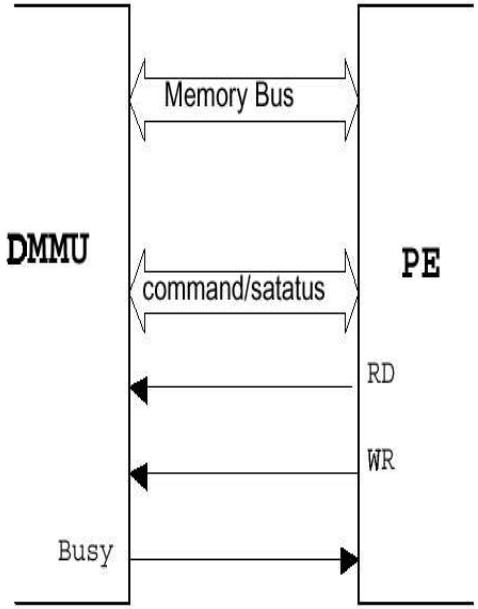
- The SoC Dynamic Memory Management Unit (SoCDMMU) is a Hardware Unit, to be a part of the SoC, that deals with the memory allocation/de-allocation among the PE's.
- The SoCDMMU allows a fast and deterministic dynamic way to allocate/de-allocate the Global Memory among the PE's.

Outline

- Introduction.
- The SoCDMMU.
- RTOS Support.
- Comparison to a Fully Shared-Memory Multiprocessor System.
- Conclusion.

PE-SoCDMMU Interface





SoCDMMU Commands

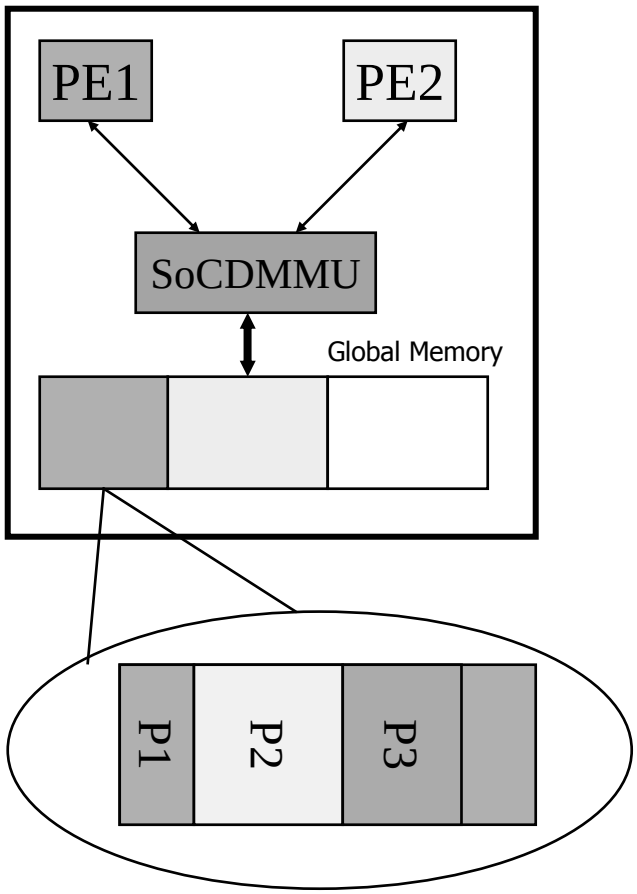
n/a	Size	Virtual Block no.	000	G_alloc_ex
SW ID	Size	Virtual Block no.	001	G_alloc_rw
SW ID	n/a	Virtual Block no.	010	G_alloc_ro
n/a	n/a	Virtual Block no.	011	G_de-alloc
New Virtual Block no.		Old Virtual Block no.	100	Move

* Note that either a processor or hardware can issue commands

Levels of Memory Management

- The SoCDMMU dynamically allocates the global on-a-chip memory among the PE's (Level 2).

- Each PE handles the local dynamic memory allocation among the processes/threads (Level 1).



Execution Times

- Synthesized using the AMI 0.5u .
- Size: 41k gates.
- Clock Speed: 100MHz.

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

Outline

- Introduction.
- The SoCDMMU.
- RTOS Support.
- Comparison to a Fully Shared-Memory Multiprocessor System.
- Conclusion.

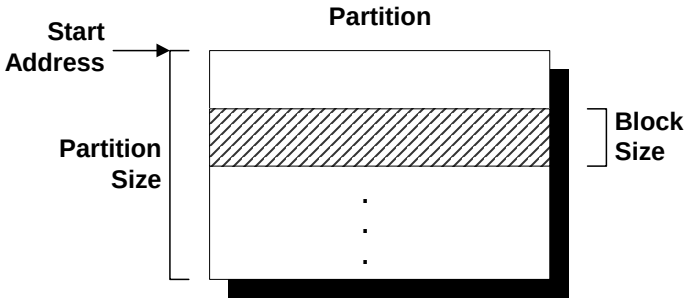
RTOS Support

- Introduction.
- Atalanta Memory Management.
- Atalanta 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. Also, most allocation algorithms usually use linked lists that does not have constant search time.
- 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 (e.g., uC-OS, eCOS, VRTXsa, pSOS, Atalanta).
- We choose to extend Atalanta, but any RTOS can be similarly extended.

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

- Extend the Atalanta RTOS with hardware support for real-time Dynamic Memory Management.
  - Best of both worlds: deterministic real-time RTOS behaviour plus dynamic allocation of global memory like general purpose operating systems
- Use the same Memory Management API Functions.
- Allow multiple real-time applications to run on the SoC, including determinstic transition times among applications (e.g., which may have just been downloaded)

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 *move* Command (pointers problems).
  - Replicate the physical address space.

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

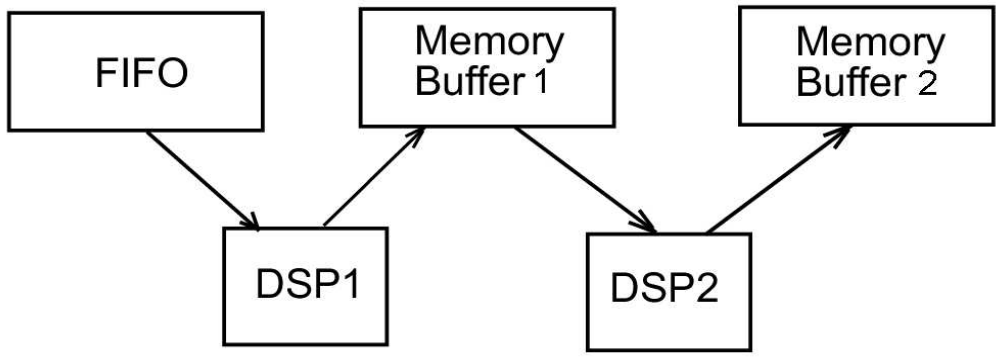
Atalanta Support for the SocDMMU

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.

Atalanta Support for the SocDMMU

Example of how to use SoCDMMU in Atalanta (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.

Atalanta Support for the SocDMMU
Example of how to use SoCDMMU in Atalanta (2)

DSP1

```
#define BUF1        0x10
.
.
SYS_ERROR e;
SYS_PARTITION p1;
SYS_MEM m1;
.
.
p1=asc_partition_create(2,1,DMMU_RW,BUF1,&e);
m1= asc_partition_gain(p1,&e);
.
.
asc_partition_free(p1,m1,&e);
```

Atalanta Support for the SocDMMU
Example of how to use SoCDMMU in Atalanta (3)

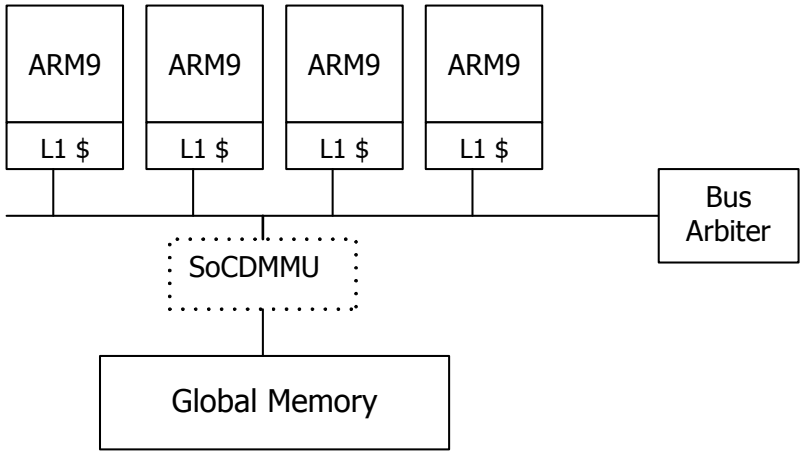
DSP2

```
#define BUF1        0x10
#define BUF2        0x20
.
.
SYS_ERROR e;
SYS_PARTITION p1;
SYS_MEM m1;
SYS_PARTITION p2;
SYS_MEM m2;
.
p1=asc_partition_create(2,1,DMMU_RO,BUF1,&e);
m1= asc_partition_gain(p1,&e);
p2=asc_partition_create(4,1,DMMU_EX,BUF2,&e);
m1= asc_partition_gain(p2,&e);
.
asc_partition_free(p2,m2,&e);
```

Outline

- Introduction.
- The SoCDMMU.
- RTOS Support.
- Comparison to a Fully Shared-Memory Multiprocessor System.
- Conclusion.

Comparison to a Fully Shared-Memory Multiprocessor System



- Global memory of 16MB; each L1 $ is 64 kB
- 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 loose the incoming message.

Comparison to a Fully Shared-Memory Multiprocessor System

- Memory Requirements

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

Comparison to a Fully Shared-Memory Multiprocessor System

Memory Management Execution time during transition from the MPEG2 player to the OFDM Receiver

	Average # of cycles/Allocation	Average # of Cycles/Deallocation	Total (Cycles)
w/o SoCDMMU	106	83	280
Using SoCDMMU**	28	14	1240
Speedup	378%	590%	440%

**** Using ARM SDT2.5 embedded *malloc()* and *free()* functions**

Note that WCET comparison would show much larger speedup.

Outline

- Introduction.
- The SoCDMMU.
- RTOS Support.
- Comparison to a Fully Shared-Memory Multiprocessor System.
- Conclusion.

Conclusion and Future Work

- 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 Atlanta RTOS to support the SoCDMMU.
- Our example shows a multiprocessor SoC that utilizes the SoCDMMU has 440% overall speedup of the application transition time over fully shared memory that does not utilize the SoCDMMU.
- For the future work, we plan to develop a tool that automatically generates (synthesizes) a customized SoCDMMU that works for a given number of processors and global memory size.

More details at the poster!

Questions