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

November 19, 2003
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.
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).

November 19, 2003
Agenda

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

November 19, 2003
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()`)

November 19, 2003
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

November 19, 2003
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

November 19, 2003
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

November 19, 2003
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 $G_{\text{allocation}}$
- The global memory de-allocation done by the SoCDMMU will be referred to as $G_{\text{deallocation}}$
- The PE can $G_{\text{allocate}}$ one or more than one block.
- Different PEs can issue the $G_{\text{allocation}}/G_{\text{deallocation}}$ 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.

<table>
<thead>
<tr>
<th>Physical Memory Block No.</th>
<th>Virtual Block No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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 is not enough memory, the OS requests memory allocation from the SoCDMMU

November 19, 2003
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 \textit{G\_allocated} it as read only.

- **Read Only**
  - The PE \textit{G\_allocates} the memory for read only. Other PE \textit{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

November 19, 2003
PE-SoCDMMU Interface

November 19, 2003
SoCDMMU Commands
The SoCDMMU Hardware

Address Converter

November 19, 2003
The SoCDMMU Hardware

The Basic SoCDMMU

November 19, 2003
The SoCDMMU Hardware

The Basic SoCDMMU

Basic SoCDMMU

November 19, 2003
The SoCDMMU Hardware

The Basic SoCDMMU

November 19, 2003
The SoCDMMU Hardware

The Basic SoCDMMU

Basic SoCDMMU

November 19, 2003
The SoCDMMU Hardware

The *Allocation Unit*

```c
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

November 19, 2003
## The SoCDMMU Hardware

### The Allocation Unit

<table>
<thead>
<tr>
<th></th>
<th>Area (NAND gates)</th>
<th>Worst Delay (ns)</th>
<th>Max. Clock Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized Allocator</td>
<td>5364</td>
<td>6.6 ns</td>
<td>150 MHz</td>
</tr>
<tr>
<td>Un-optimized Allocator</td>
<td>17930</td>
<td>56.3 ns</td>
<td>17.5 MHz</td>
</tr>
<tr>
<td>Comparison</td>
<td>3.3X</td>
<td>8.5X</td>
<td></td>
</tr>
</tbody>
</table>

- 256 G\_blocks.
- Synthesized using Synopsys Design Compiler\textsuperscript{TM} and a TSMC 0.25u library from LEDA Systems.

November 19, 2003
The SoCDMMU Hardware
Execution Times/Synthesis

- Synthesized using the TSMC 0.25μm.
- 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Number of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_alloc_ex</td>
<td>4</td>
</tr>
<tr>
<td>G_alloc_rw</td>
<td>4</td>
</tr>
<tr>
<td>G_alloc_ro</td>
<td>3</td>
</tr>
<tr>
<td>G_dealloc</td>
<td>4</td>
</tr>
<tr>
<td>4-Processors WCET</td>
<td>16</td>
</tr>
</tbody>
</table>
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

November 19, 2003
Introduction

The Design Productivity Gap

CAGR 58%

CAGR 21%

Logic Transistors per Chip (M)

Productivity (K) Trans./Staff-Mo.

Source ITRS


November 19, 2003
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.

November 19, 2003
DX-Gt Overview

November 19, 2003
User Specified Parameters

- The number and type of PEs
- The number and size of the global on-chip memory $G_{blocks}$
- The memory type
- The scheduling scheme to resolve concurrent SoCDMMU requests
- Memory $G_{blocks}$ initially assigned to the PEs
The SoCDMMU Generation

November 19, 2003
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

November 19, 2003
Customizing the SoCDMMU

socdmmu.vpp

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

module SoCDMMU (. . . .);

.
.
.

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

socdmmu.v

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

November 19, 2003

in[n-1:n-m-2]
Allocation Unit Optimization

- Delay over the critical path
  \[ \text{Delay} = C + k \cdot D_s + m \cdot d_1 \]

- Also, we have
  \[ n = k \cdot m : n \text{ is the no. of } G\text{-blocks} \]

- This leads to
  \[ \text{Delay} = C + k \cdot D_s + \left(\frac{n}{k}\right) \cdot d_1 \]

- The Delay is minimum when
  \[ k = \sqrt{n \cdot d_1 / D_s} : k \text{ is power of 2} \]
Agenda

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

November 19, 2003
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

November 19, 2003
Atalanta Support for the SocDMMU

New API New Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>asc_partition_create</em></td>
<td>Create a partition by requesting memory allocation from the SoCDMMU if necessary.</td>
</tr>
<tr>
<td><em>asc_partition_delete</em></td>
<td>Delete a partition and de-allocate memory block if required.</td>
</tr>
<tr>
<td><em>asc_memory_find</em></td>
<td>Find a place in the PE address space to which to map the allocated memory.</td>
</tr>
</tbody>
</table>
Agenda

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

Experiments

November 19, 2003
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 software simulator/debugger and Synopsys VCS Verilog simulator
- ARM SDT was used for software development

November 19, 2003
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

November 19, 2003
Experiment 1

- Sequence of Memory Allocations Required

<table>
<thead>
<tr>
<th>MPEG-2 Player</th>
<th>OFDM Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Kbytes</td>
<td>34 Kbytes</td>
</tr>
<tr>
<td>500 Kbytes</td>
<td>32 Kbytes</td>
</tr>
<tr>
<td>5 Kbytes</td>
<td>1 Kbytes</td>
</tr>
<tr>
<td>1500 Kbytes</td>
<td>1.5 Kbytes</td>
</tr>
<tr>
<td>1.5 Kbytes</td>
<td>32 Kbytes</td>
</tr>
<tr>
<td>0.5 Kbytes</td>
<td>8 Kbytes</td>
</tr>
<tr>
<td></td>
<td>32 Kbytes</td>
</tr>
</tbody>
</table>
# Experiment 1

Speedup of a single `malloc()`

<table>
<thead>
<tr>
<th></th>
<th>Execution Time (Average Case)</th>
<th>Execution Time (Worst Case)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDT2.5 embedded malloc()</strong></td>
<td>106 cycles</td>
<td>559 cycles</td>
</tr>
<tr>
<td>uClib <code>malloc()</code></td>
<td>222 cycles</td>
<td>1646 cycles</td>
</tr>
<tr>
<td>SoCDMMU allocation</td>
<td>28 cycles</td>
<td>199 cycles</td>
</tr>
<tr>
<td><strong>Speed up over SDT malloc()</strong></td>
<td>3.78X</td>
<td>2.8X</td>
</tr>
<tr>
<td><strong>Speed up over uClibc malloc()</strong></td>
<td>7.92X</td>
<td>8.21X</td>
</tr>
</tbody>
</table>
## Experiment 1

### Speedup of a single `free()`

<table>
<thead>
<tr>
<th></th>
<th>Execution Time (Average Case)</th>
<th>Execution Time (Worst Case)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>SDT2.5 embedded free()</em></td>
<td>83 cycles</td>
<td>186 cycles</td>
</tr>
<tr>
<td>uClib <code>free()</code></td>
<td>208 cycles</td>
<td>796 cycles</td>
</tr>
<tr>
<td>SocDMMU deallocation</td>
<td>14 cycles</td>
<td>28 cycles</td>
</tr>
<tr>
<td>Speed up over SDT <code>free()</code></td>
<td>5.9X</td>
<td>6.64X</td>
</tr>
<tr>
<td>Speed up over uClibc <code>free()</code></td>
<td>14.8X</td>
<td>28.42X</td>
</tr>
</tbody>
</table>
# Experiment 1

**Speedup in transition time**

<table>
<thead>
<tr>
<th></th>
<th>Using the SOCDMMU</th>
<th>Using SDT <code>malloc()</code> and <code>free()</code></th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Case</strong></td>
<td>280 cycles</td>
<td>1240 cycles</td>
<td>4.4X</td>
</tr>
<tr>
<td><strong>Worst Case</strong></td>
<td>1244 cycles</td>
<td>4851 cycles</td>
<td>3.9X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Using the SOCDMMU</th>
<th>Using uClibc <code>malloc()</code> and <code>free()</code></th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Case</strong></td>
<td>280 cycles</td>
<td>2593 cycles</td>
<td>9.26X</td>
</tr>
<tr>
<td><strong>Worst Case</strong></td>
<td>1244 cycles</td>
<td>15502 cycles</td>
<td>12.46X</td>
</tr>
</tbody>
</table>
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

November 19, 2003
# Experiment 2

Speedup in Execution Time

## Glibc malloc() & free()

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>E.T. (Cycles)</th>
<th>Memory Management E. T. (Cycles)</th>
<th>% of E. T. used to Memory Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>318307</td>
<td>31512</td>
<td>9.90%</td>
</tr>
<tr>
<td>FFT</td>
<td>375988</td>
<td>101998</td>
<td>27.13%</td>
</tr>
<tr>
<td>RADIX</td>
<td>694333</td>
<td>141491</td>
<td>20.38%</td>
</tr>
</tbody>
</table>

## Using the SoCDMMU

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>E.T. (Cycles)</th>
<th>Memory Management E. T. (Cycles)</th>
<th>% of E. T. used to Memory Management</th>
<th>% Reduction in Time used to Manage Memory</th>
<th>% Reduction in Benchmark E. T.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>288271</td>
<td>1476</td>
<td>0.51%</td>
<td>95.31%</td>
<td>9.44%</td>
</tr>
<tr>
<td>FFT</td>
<td>276941</td>
<td>2951</td>
<td>1.07%</td>
<td>97.10%</td>
<td>26.34%</td>
</tr>
<tr>
<td>RADIX</td>
<td>558347</td>
<td>5505</td>
<td>0.99%</td>
<td>96.10%</td>
<td>19.59%</td>
</tr>
</tbody>
</table>

November 19, 2003
Area Estimation of The SoC

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

* Using dual-port 6T SRAM Cells

November 19, 2003
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 SoCDMMU 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.

November 19, 2003
Topic Related Publications


- Hardware Software Real-Time Operating System, The δ RTOS, preparing 2 chapters
Questions

November 19, 2003