Lab4: Memory Issues

Speaker: Nelson Chang
Directed by Prof. Tian-Sheuan Chang
March, 2004, NCTU
Goal of This Lab

- Understand how to allocate data storages and arrange accesses.
- Mapping memory using scatter-loading.
Outline

- Memories in ARM Integrator
- Memory Characteristics
- Scatter-loading
- Lab4 – Memory Issues
Memories in ARM Integrator

- **Core Module**
  - 256/512KB SSRAM
  - 16~256 MB SDRAM

- **Logic Module.**
  - 1MB ZBT SSRAM

- **Asic Platform**
  - 256KB boot ROM
  - 32MB flash memory.
  - 512KB SSRAM
Core Module Memory Map

- Standalone:
  - Abort
  - SSRAM alias
  - CM registers
  - SDRAM
  - SSRAM

- Attached to a motherboard:
  - Motherboard
  - SSRAM alias
  - CM registers
  - SDRAM
  - BootROM/flash
  - SSRAM

- nMBDET = 1
  - REMAP = x

- nMBDET = 0
  - REMAP = 0

- nMBDET = 0
  - REMAP = 1
Logic Module Memory Map
System Controller FPGA Diagram
Integrator/Core Module

SOC Consortium Course Material
Integrator/Logic Module
Integrator/ASIC Platform

SSRAM

Flash

Boot ROM

Not to scale
Integrator LM Block Diagram

- ZBT SSRAM
- OSC1
- OSC2
- Multi-ICE
- Trace
- Module/motherboard connectors
- LA connector
- Push button
- Switches
- LEDs
- Prototyping grid
Integrator/AP Block Diagram
Outline

- Memories in ARM Integrator
- Memory Characteristics
- Scatter-loading
- Lab4 – Memory Issues
Memory Characteristics

- **SRAM**
  - Static cell
  - Fast: Access latency <10ns
  - Expensive
  - Single address decoding
  - On-chip memory

- **DRAM**
  - Capacitive cell
  - Slower: Access latency 200~100ns
  - High density, cheapest memory
  - Row and column address strobing phase
    - Non-sequential access takes longer time
    - Sequential access which falls within the same row is faster
  - Off-chip memory
Outline

- Memories in ARM Integrator
- Memory Characteristics
- Scatter-loading
- Lab4 – Memory Issues
Scatter-loading

- An image is made up of regions and output sections. Every region in the image can have a different load and execution address.
- The scatter-loading mechanism enables you to specify the memory map of an image to armlink.
- Scatter-loading gives you complete control over the grouping and placement of image components.
- Scatter-loading is especially important for writing codes for ROM in embedded systems.
When to use scatter-loading

- **Complex memory maps**
  - Code and data that must be placed into many distinct areas of memory require detailed instructions on which section goes into which memory space.

- **Different types of memory**
  - Many systems contain flash, ROM, SDRAM, and fast SRAM. A scatter-loading description can match the code and data with the most appropriate type of memory. For example, the interrupt code might be placed into fast SRAM to improve interrupt response time and infrequently used configuration information might be placed into slower flash memory.

- **Memory-mapped I/O**
  - The scatter-loading description can place a data section at a precise address in the memory map.

- **Functions at a constant location**
  - A function can be placed at the same location in memory even though the surrounding application has been modified and recompiled.

- **Using symbols to identify the heap and stack**
  - Symbols can be defined for the heap and stack location and the location of the enclosing module can be specified when the application is linked.

- Scatter-loading is therefore almost always required for implementing embedded systems because these use ROM, RAM, and memory-mapped I/O.
Information specified in a scatter-loading description file:

- grouping information describing how input sections are grouped into regions
- placement information describing the addresses where image regions are to be located in the memory maps.

These information are passed to armlink to construct the image.
### Image with simple memory map

<table>
<thead>
<tr>
<th>Load view</th>
<th>Execution view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero fill</td>
<td>SRAM</td>
</tr>
<tr>
<td>RW section</td>
<td>0x10000</td>
</tr>
<tr>
<td>RO section</td>
<td>ROM</td>
</tr>
<tr>
<td></td>
<td>0x0000</td>
</tr>
<tr>
<td></td>
<td>0x8000</td>
</tr>
<tr>
<td></td>
<td>0x16000</td>
</tr>
</tbody>
</table>

**Diagram: Scatter-loading description file**

- **Start address for load region**
- **Maximum size of load region**
- **Start address for exec region**
- **Maximum size of this exec region**
- **Place all code and RO data into this exec region**
- **Maximum size of this exec region**
- **Place all RW and ZI data into this exec region**

- **LOAD_ROM** 0x0000 0x8000
- **EXEC_ROM** 0x0000 0x8000
- **RAM** 0x10000 0x6000

**Name of load region**

- **Name of first exec region**
- **Use this region for all modules**

**Name of second exec region**

- **Start of second exec region**
- **Use this region for all modules**
Outline

- Memories in ARM Integrator
- Memory Characteristics
- Scatter-loading
- Lab4 – Memory Issues
Lab 4: Memory Issues

Goal
- Experience the effect due to memory issues
  - Understand how to allocate data storages and arrange accesses.
  - Mapping memory using scatter-loading.

Guidance
- Observer the pointers
- Identify which regions are the pointers pointing

Principles
- Memories on Integrator system
- DRAM characteristics
- Scatter-loading

Steps
- Comparison of memory access pattern in DRAM is practiced.

Requirements and Exercises
- Modified the scatter-loading description file to map the whole image into SRAM for performance.

Discussion
- Where is the local array allocated in the memory?
Memory Storage and Access Pattern

- Linearly (sequentially) stored data
  - Read 1 block needs: 4NS_cycle + 12S_cycle

- Block tile stored data
  - Read 1 block needs: 1NS_cycle + 15S_cycle
Stack and Heap

Stack
- Whenever a (non-trivial) function is called, a new activation frame is created on the stack containing a backtrace record, local (non-static) variables, and so on.
- When a function returns, its stack space is automatically recovered and will be reused for the next function call.

Heap
- An area of memory used to satisfy program requests (malloc()) for more memory for new data structures.
- A program which continues to request memory over a long period of time should be careful to free up all sections that are not used, otherwise the heap will run out the memory.
Scatter-loading description file & Memory mapping

- **DRAM.scf**

```plaintext
LR_1 0x00008000 0x10000000
{
    ALL +0 0x00040000
    {
        *(+RO)
    }
    RWZI 0x00040000 0xFFFF0000
    {
        *(+RW,+ZI)
    }
    HEAPS +0 UNINIT 0xFFFF0000
    {
        heaps.o(+ZI)
    }
    STACKS 0x10000000 UNINIT 0xFFFF0000
    {
        stack.o(+ZI)
    }
}
```

- **Memory mapping**

![Memory mapping diagram]
Files Descriptions

- **Source files:**
  - main.o
    - Main source code implementing block access to linearly (sequentially) stored data and block tile stored data.
  - retarget_simple.c
    - Retargets \_\_user\_initial\_stackheap() to place the stack and heap.
  - heaps.s
    - Assembly code to export variable $bottom\_of\_heaps$
  - stacks.s
    - Assembly code to export variable $top\_of\_stacks$

- **Caution:**
  - when using scatter-loading, you **must** use retarget_simple.c to retarget \_\_user\_initial\_stackheap() to place the stack and heap. If you do not, there might be link errors because the default implementation provided by the C library attempts to use Image\$$ZI$$Limit that is not defined when scatter loading is used.
References

[1] Using Scatter-loading for…, ADS Developers Guide [DUI0056D, 6.6 6.7 6.8 6.9]