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.

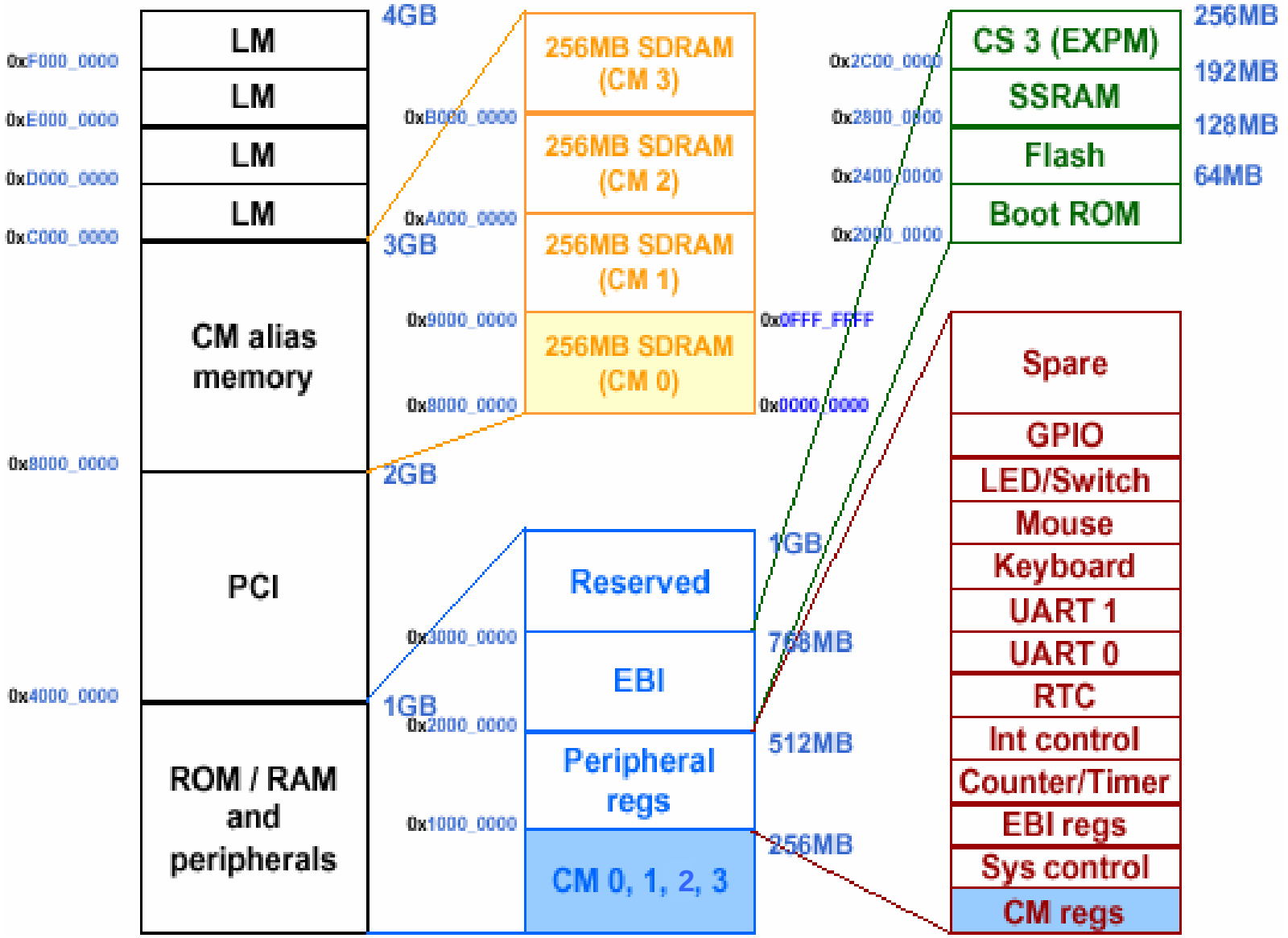
Outline

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

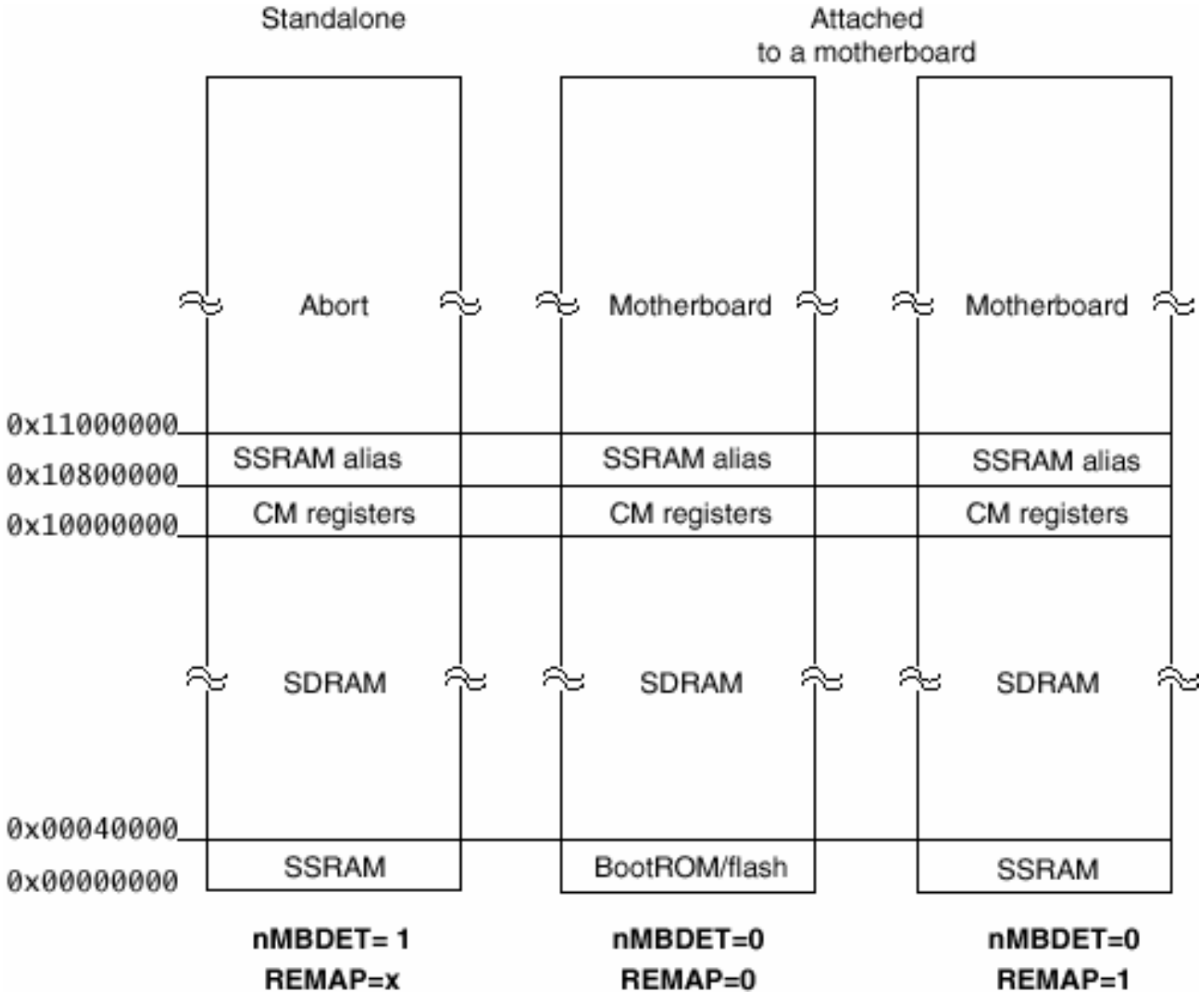
Memories in ARM Integrator

- Core Module
  - **256**/512KB SSRAM
  - 16~**256** MB SDRAM
- Logic Module.
  - **1**MB ZBT SSRAM
- Asic Platform
  - **256**KB boot ROM
  - **32**MB flash memory.
  - **512**KB SSRAM

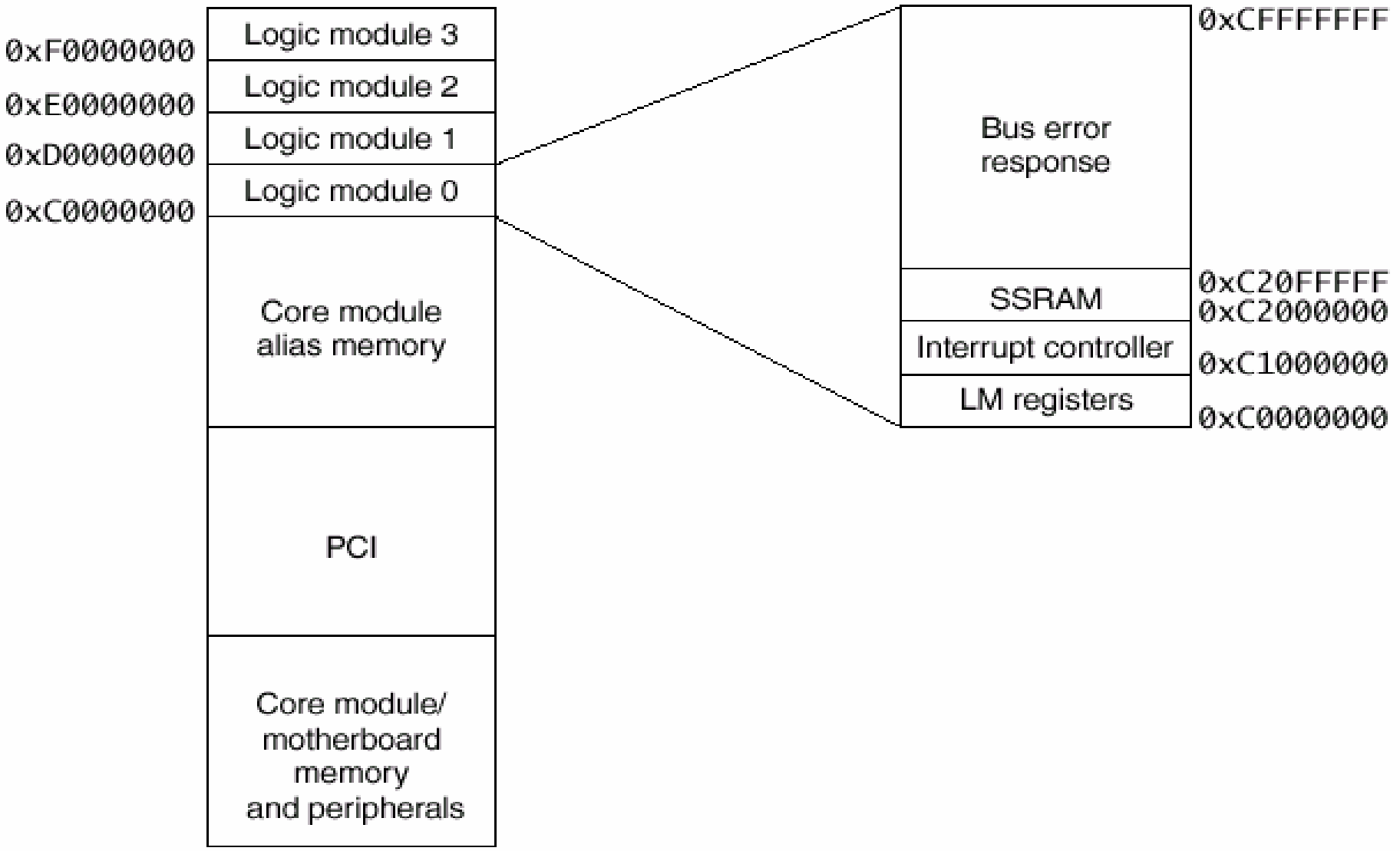
Integrator System Memory Map



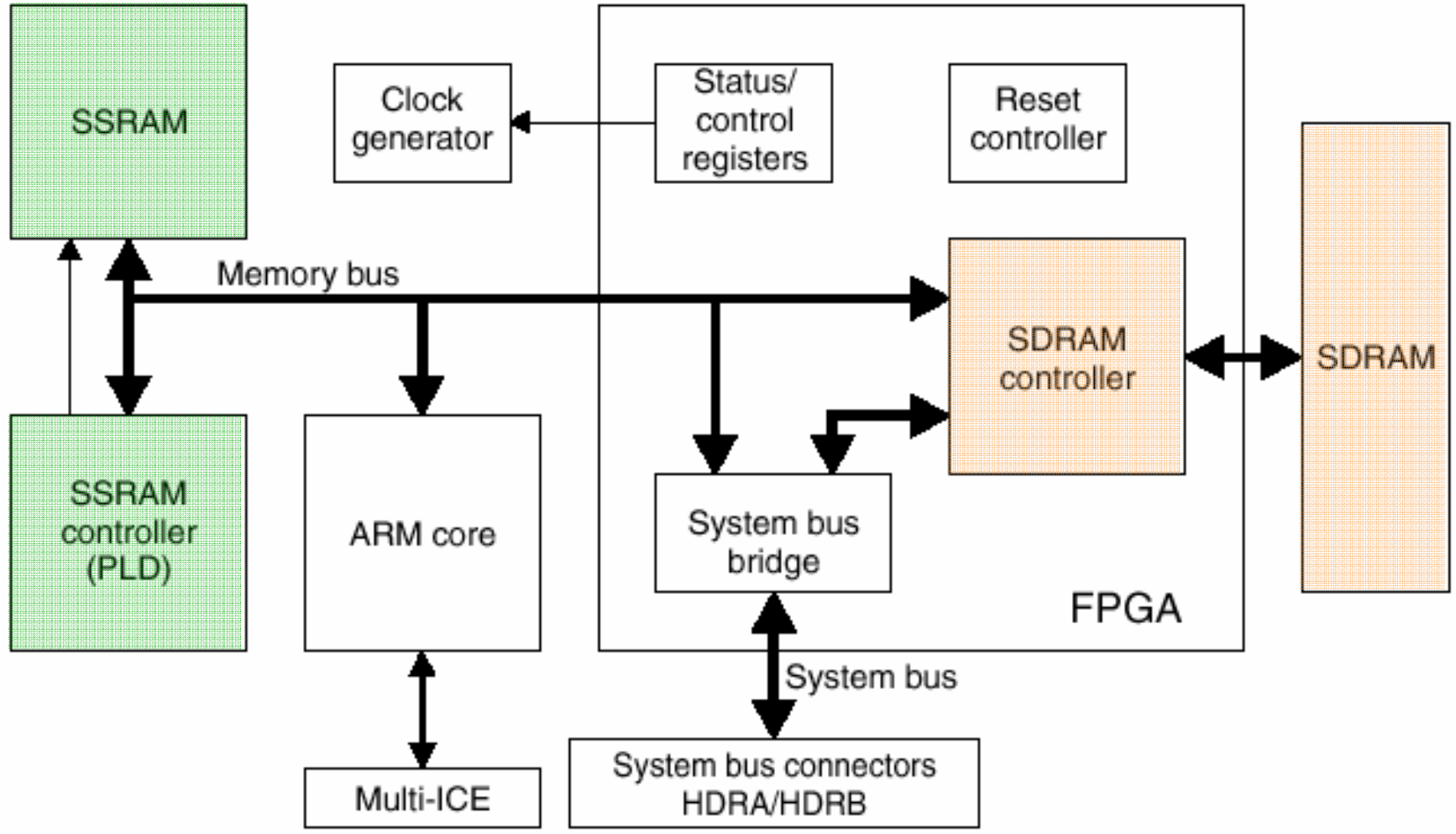
Core Module Memory Map



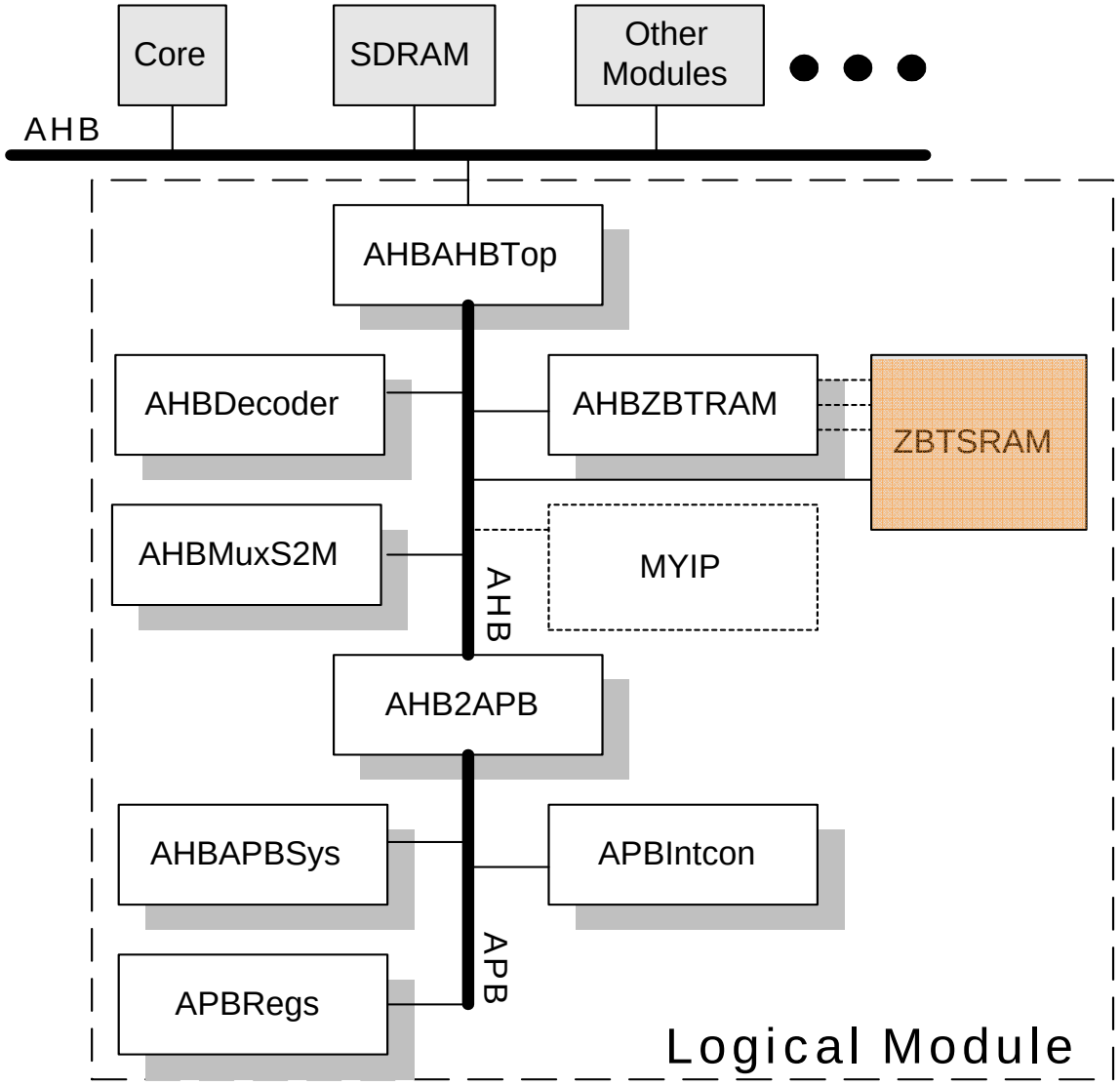
Logic Module Memory Map



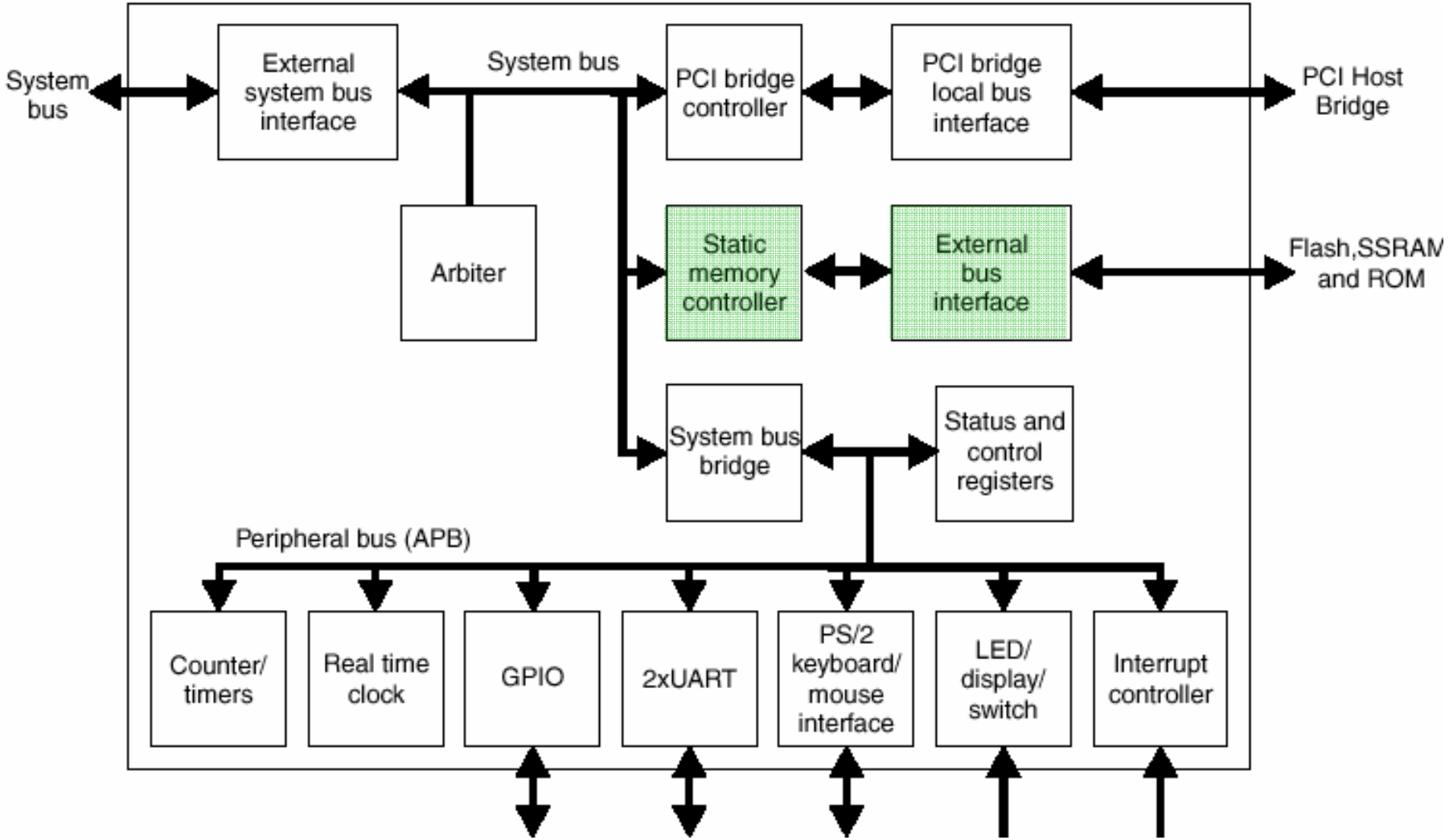
CM FPGA Diagram



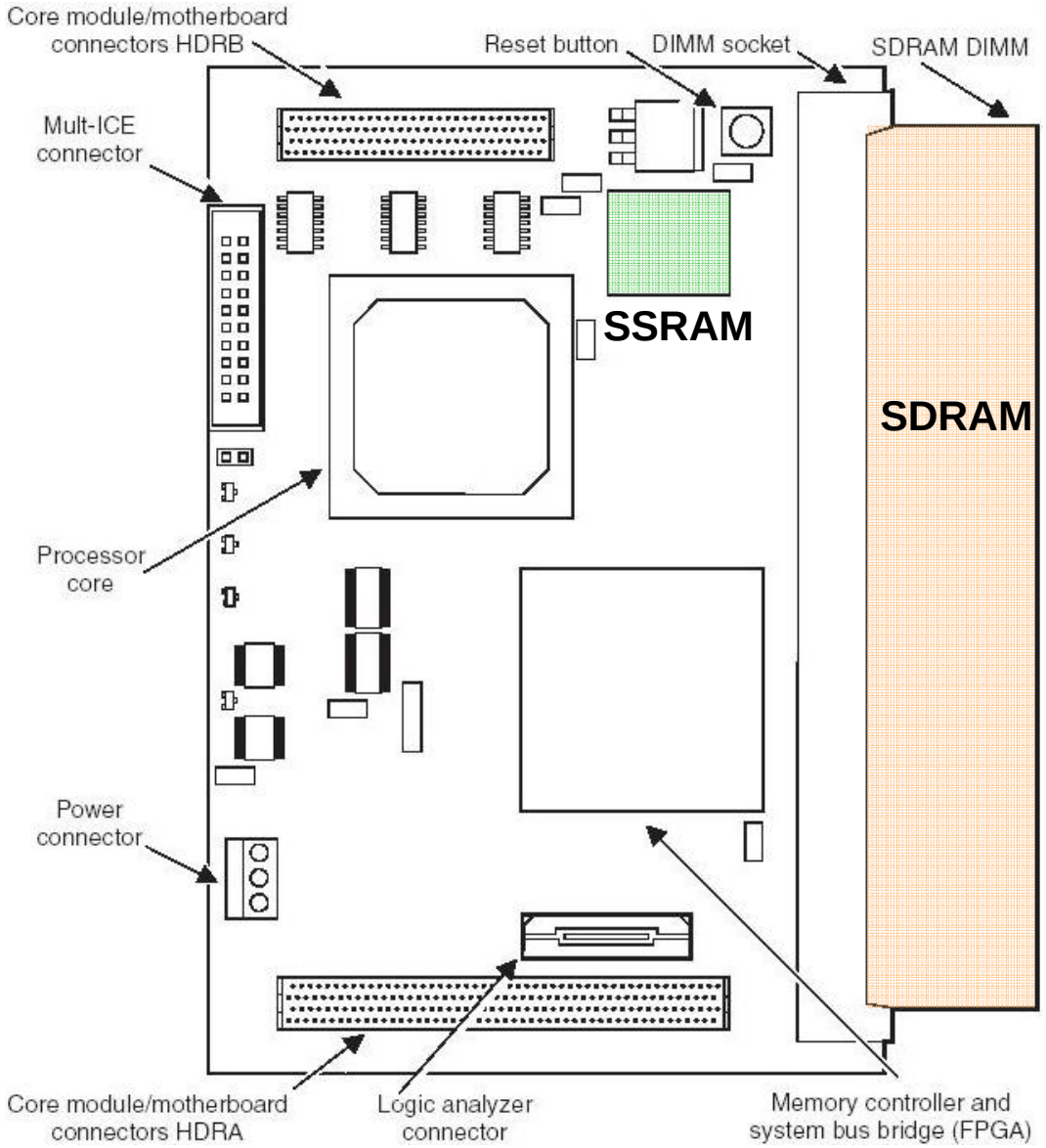
LM FPGA Diagram



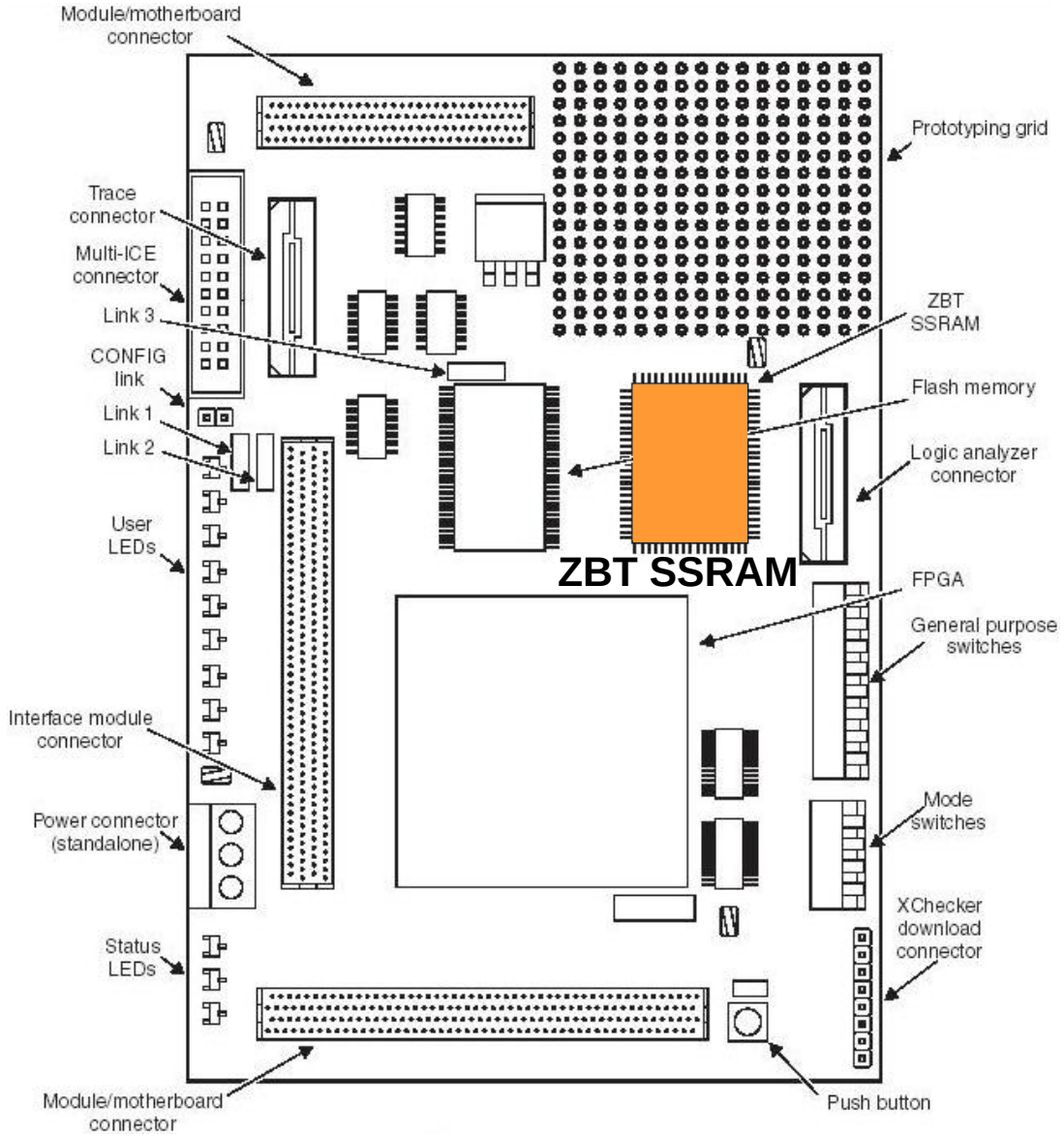
System Controller FPGA Diagram



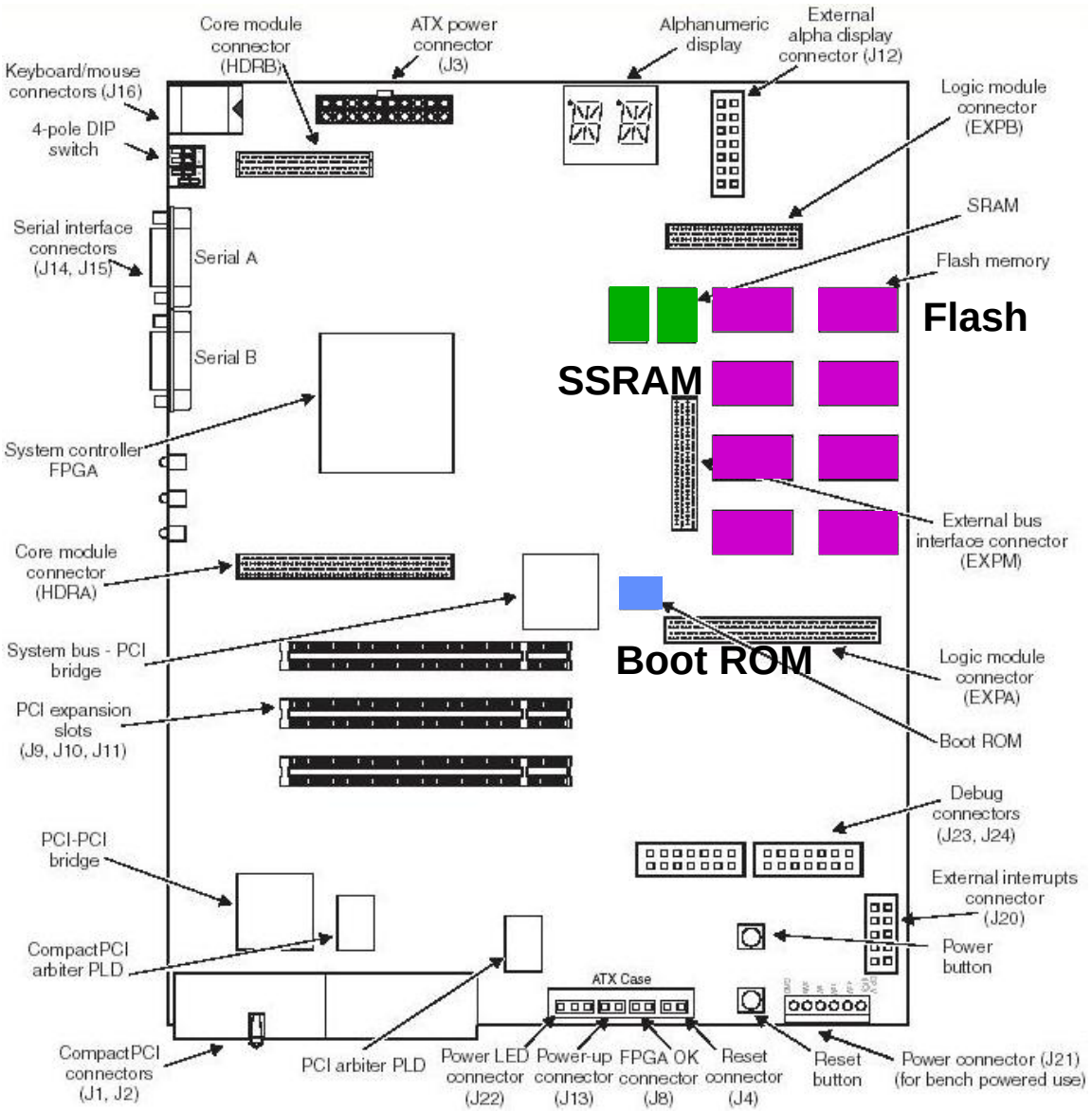
Integrator/Core Module



Integrator/Logic Module

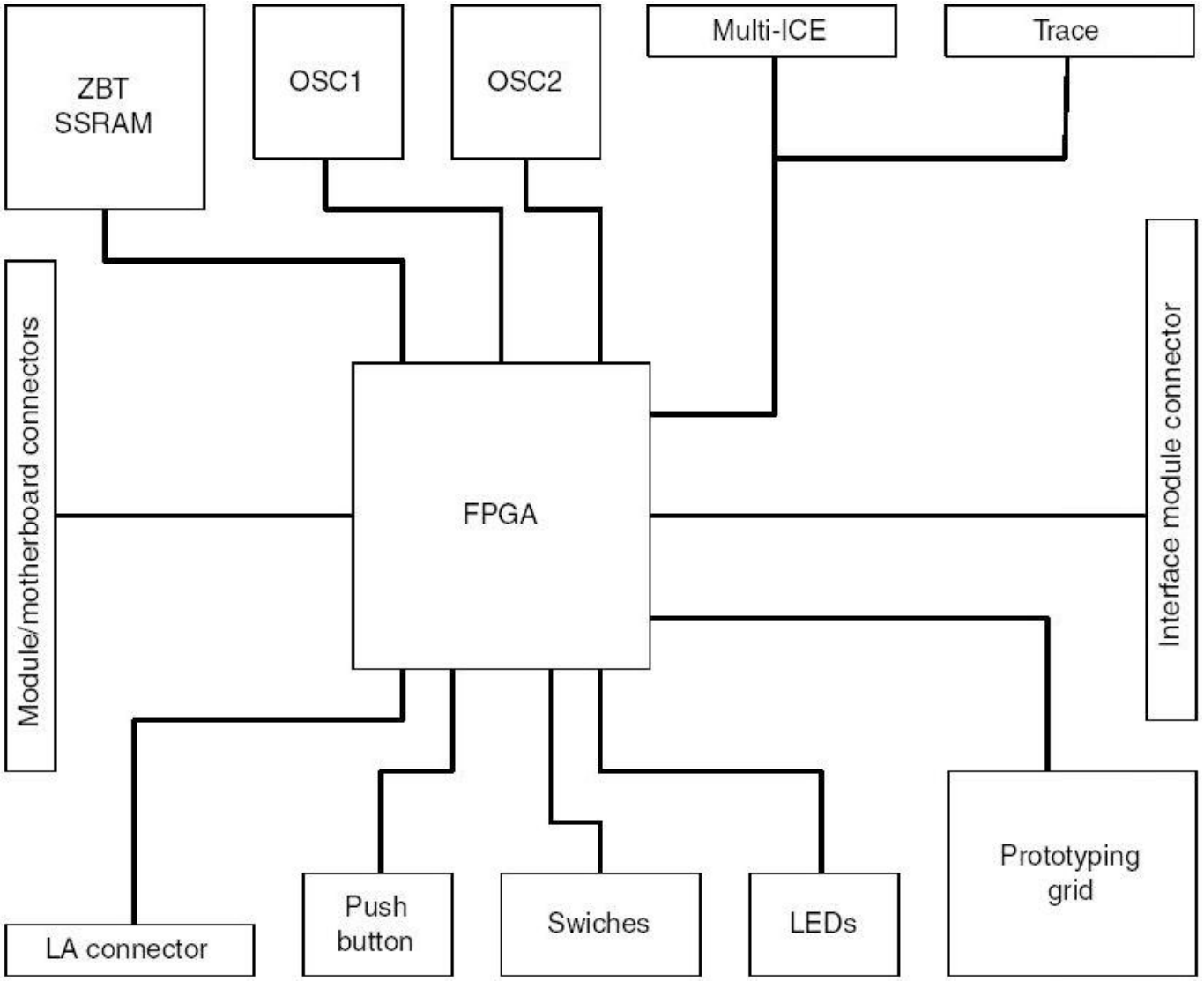


Integrator/ASIC Platform

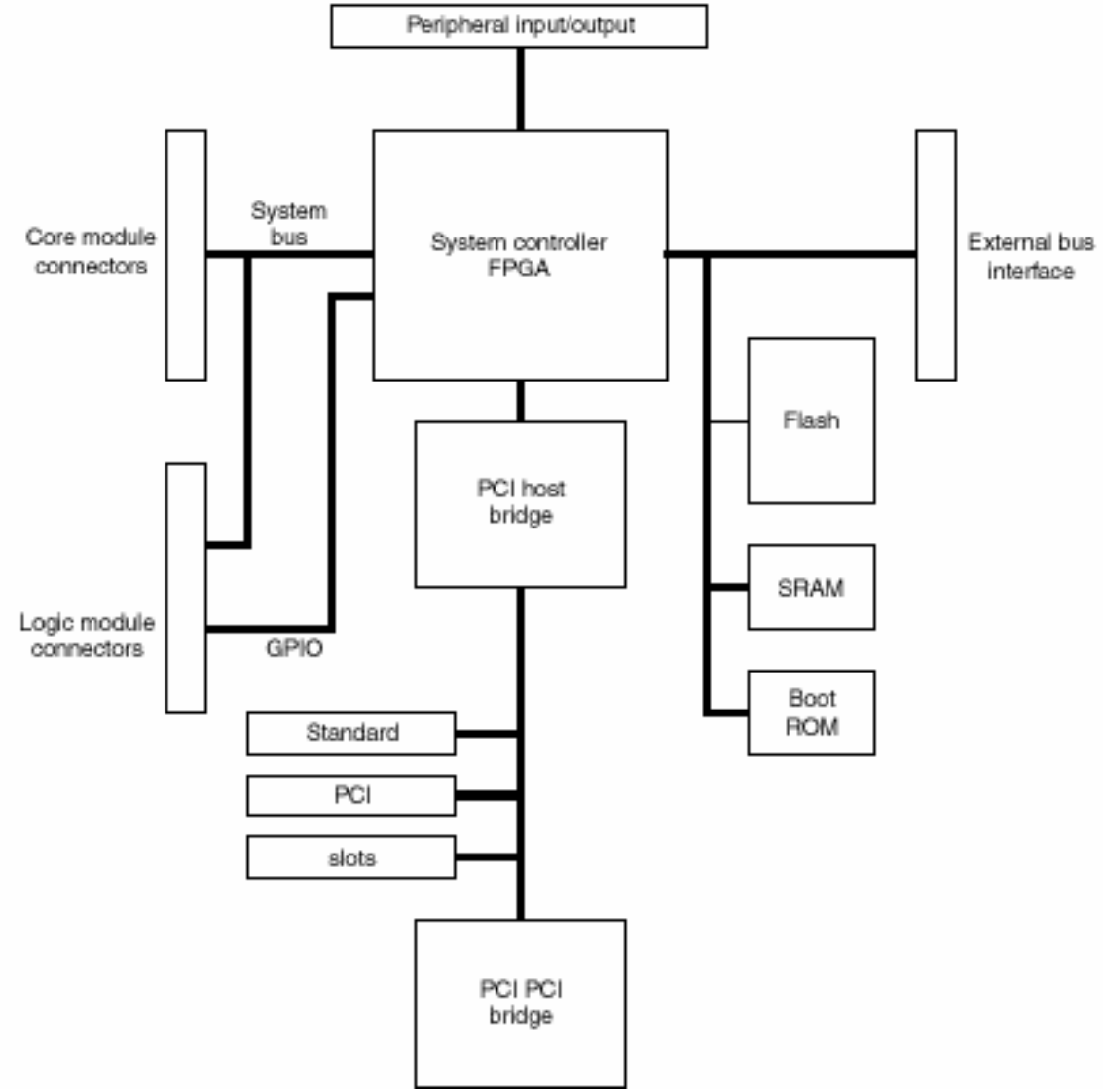


Not to scale

Integrator LM Block Diagram



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

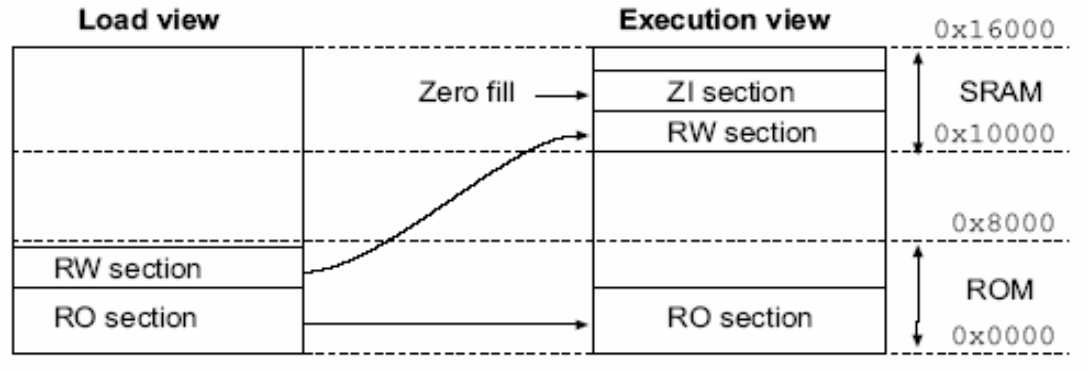
- **Complex memory maps**
  - Code and data that must be placed into many distinct areas of memory require detailed instructions on which section goes into which memory space.
- **Different types of memory**
  - Many systems contain flash, ROM, SDRAM, and fast SRAM. A scatter-loading description can 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.

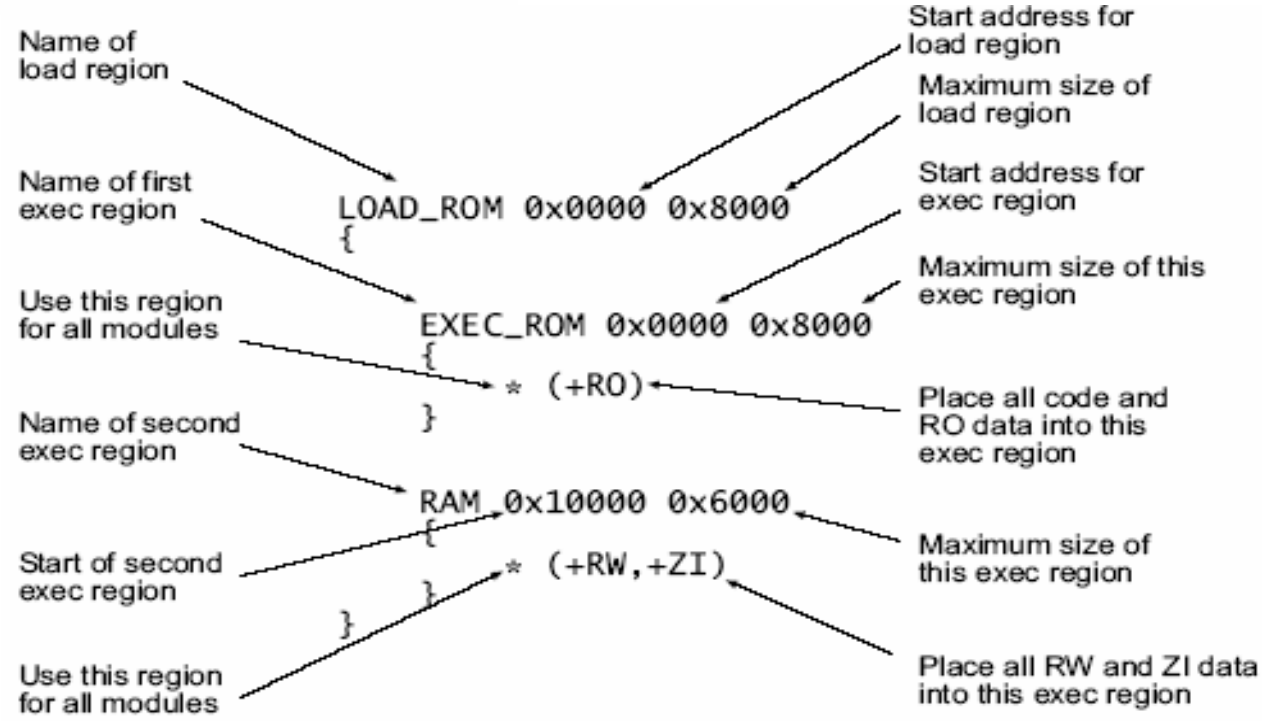
Scatter-loading description file (1/2)

- 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

Scatter-loading description file (2/2)

- Image with simple memory map





Outline

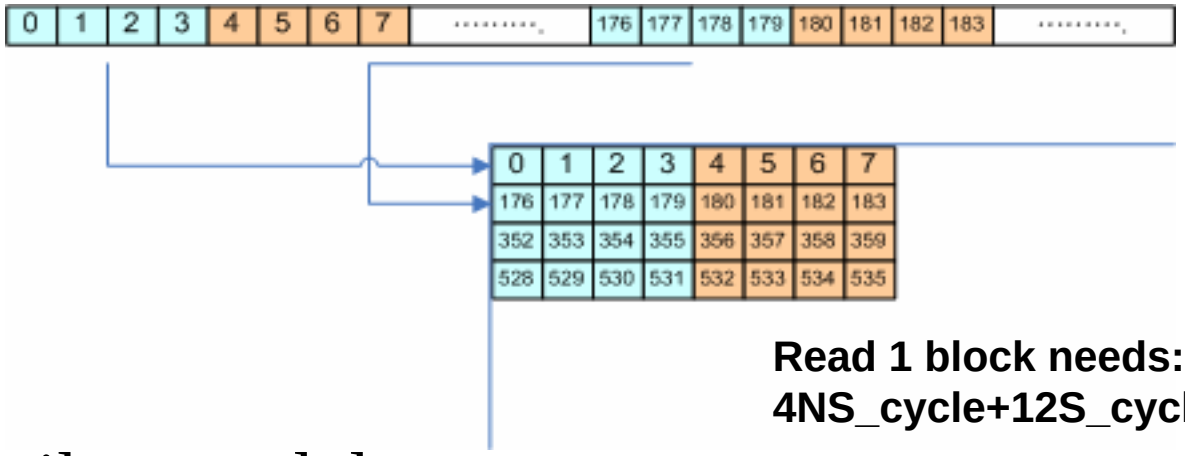
- Memories in ARM Integrator
- Memory Characteristics
- Scatter-loading
- ***Lab4 – Memory Issues***

Lab 4: Memory Issues

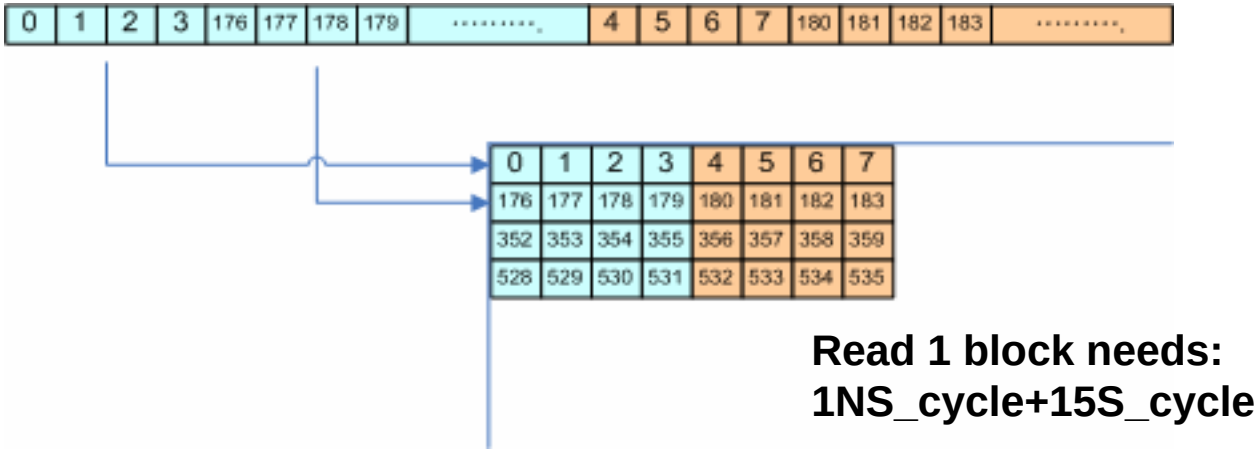
- Goal
  - Experience the effect due to memory issues
    - Understand how to allocate data storages and arrange accesses.
    - Mapping memory using scatter-loading.
- Principles
  - Memories on Integrator system
  - DRAM characteristics
  - Scatter-loading
- Guidance
  - Observer the pointers
  - Identify which regions are the pointers pointing
- 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



- Block tile stored data



Stack and Heap

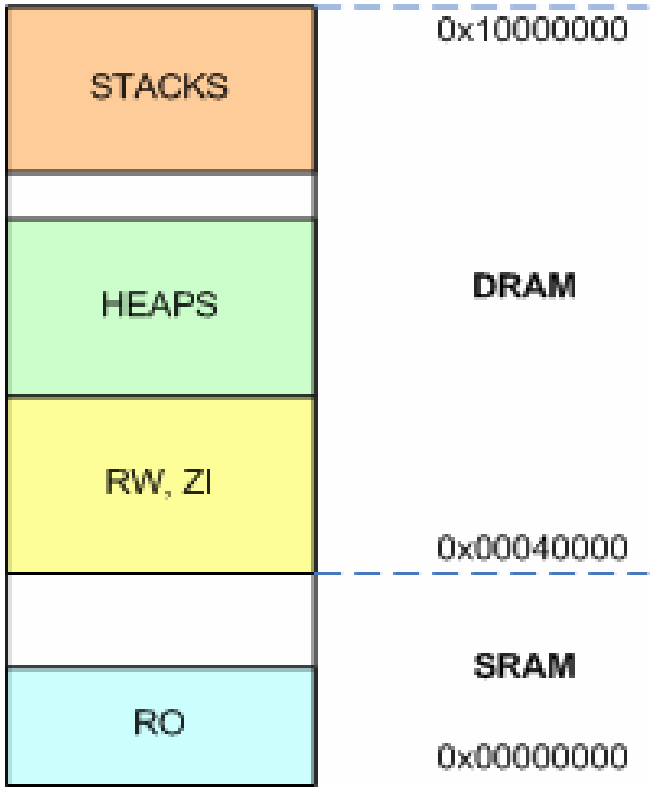
- Stack
  - Whenever a (non-trivial) function is called, a new activation frame is created on the stack containing a **backtrace record**, **local (non-static) variables**, and so on.
  - When a function returns, its stack space is automatically recovered and will be reused for the next function call.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

```
LR_1 0x00008000 0x10000000
{
        ALL +0 0x00040000
        {
                *(+RO)
        }
        RWZI 0x00040000 0x0FFC0000
        {
                *(+RW,+ZI)
        }
        HEAPS +0 UNINIT 0x0FFC0000
        {
                heaps.o(+ZI)
        }
        STACKS 0x10000000 UNINIT 0x0FFC0000
        {
                stack.o(+ZI)
        }
}
```

❑ Memory mapping



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_inital_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_stackss`
- 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]

[2] Using Scatter-loading Description Files, ADS Linker and Utilities Guide [DUI0151A, 5]

[3] Specify code from C and C++, ADS Compilers and Libraries Guide [DUI0067D]

[4] Memory Map, ADS Debug Target Guide [DUI0058D, 2.8]

[5] Steve Furber, “ARM System-on-Chip Architecture,” Addison Wesley, 6.9, 8.1, 2000.