*Memory Organization on the StrongARM** SA-1100 Evaluation Platform*

Application Note

October 1998

Order Number: 278203-001

Application Note

Contents

Figures

Tables

1.0 Introduction

This document describes the memory maps for the Intel StrongARM** SA-1100 Microprocessor Evaluation Platform. The StrongARM SA-1100 Portable Communications Microcontroller has a fixed memory arrangement for static memory, microcontroller registers, and dynamic memory. This document describes these layout and implementation-specific features of the SA-1100 Microprocessor Evaluation Platform. The virtual memory map used by Angel (Version 1.0*X*) and uHAL (Version 1.0*X*) is also described.

This application note should be used in conjunction with the *SA-1100 Microprocessor Technical Reference Manual* (278088).

2.0 SA-1100 Memory Layout

The address map is fixed for most aspects of the SA-1100's external access.

2.1 Main Partitions

As shown in Table 1, the map is divided into four main partitions of 1 GB each.

- Partition 0—Located at the bottom of memory and is dedicated to static memory devices (ROM, SRAM, and Flash) and to the PCMCIA expansion bus area. It occupies addresses 0x0000 0000 through 0x3fff ffff.
- Partition 1—Occupies addresses 0x4000 0000 through 0x7fff ffff and is reserved. Any access to this reserved space results in a Data Abort Exception.
- Partition 2—Occupies addresses 0x8000 0000 through 0xbfff ffff and contains all onchip registers (except those specified by the ARM V4 architecture).
- Partition 3—Occupies addresses 0xC000 0000 through 0xffff ffff and contains DRAM memory. There are four fixed banks of 128 MB each, where DRAM memory can be populated. The next range of memory is a 128 MB block, which is mapped within the memory controller and returns zeros when read. This facilitates rapid cache flushing by avoiding external memory accesses to load new data into the cache. The top 384 MB area is reserved and accesses result in a Data Abort Exception.

Table 1. SA-1100 Address Map

Partition	Start	End	Size	Used[1]	Usage
Partition 3	0xe800 0000	0xffff ffff	384 MB	-	Reserved
	0xe000 0000	0xe7ff ffff	128 MB	1 MB	Zeros (Cache Clean) Bank
	0xd800 0000	0xdfff ffff	128 MB	4 MB	Dynamic RAM Bank Select 3
	0xd000 0000	0xd7ff ffff	128 MB	4 MB	Dynamic RAM Bank Select 2
	0xc800 0000	0xcfff ffff	128 MB	4 MB	Dynamic RAM Bank Select 1
	0xc000 0000	0xc7ff ffff	128 MB	4 MB	Dynamic RAM Bank Select 0
Partition 2	0xb000 0000	0xbfff ffff	256MB	256MB	LCD and DMA Registers
	0xa000 0000	0xafff ffff	256MB	256MB	Memory and Expansion Registers
	0x9000 0000	0x9fff ffff	256MB	256MB	System Control Module Registers
	0x8000 0000	0x8fff ffff	256MB	256MB	Peripheral Module Registers
Partition 1	0x4000 0000	0x7fff ffff	1GB	-	Reserved
Partition 0	0x3000 0000	0x3fff ffff	256MB	256MB	PCMCIA Socket 1
	0x2000 0000	0x2fff ffff	256MB	256MB	PCMCIA Socket 0
	0x1800 0000	0x1fff ffff	128MB	1MB	Static Bank Select 3
	0x1000 0000	0x17ff ffff	128MB	512KB	Static Bank Select 2 (SRAM)
	0x0800 0000	0x0fff ffff	128MB	256KB	Static Bank Select 1 (Flash)
	0x0000 0000	0x07ff ffff	128MB	256KB	Static Bank Select 0 (ROM)

1. **Used** is the amount of each area implemented on the Evaluation Platform.

2.2 SA-1100 Evaluation Platform Implementation

The SA-1100 evaluation board has two banks of 256 KB of Flash Memory, the first of which (at 0x0000 0000) cannot be programmed in-situ, so is effectively ROM. The third static bank has 512 KB of SRAM, and the last static bank is used to implement an external register to control PCMCIA voltage levels, resets, and slot enables. There are 4 MB of DRAM fitted to each DRAM bank.

3.0 Virtual Memory Map

The Angel debug monitor (Version 1.0*X*) and uHAL software library (Version 1.0*X*) implement a common virtual memory map, as well as enabling caching and write buffering. The SA-1100 memory-management unit (MMU) is used to place DRAM at 0 so that software running on Angel or using uHAL routines can write exception vectors there.

3.1 Remapped Areas

Dynamic memory is mapped from 0x000 0000 to 0x00ff ffff, allowing applications to use memory without any concerns about bank sizes. This memory is defined as cacheable and bufferable for maximum performance with the SA-1100. Dynamic memory also has a 1-1 mapping with its physical location. These areas are defined as bufferable so that sequential writes are merged to provide maximum performance, but these areas are not cacheable so that screen and DMA buffers can be written without having to worry about flushing caches.

ROM is mapped to 64 MB and is cacheable. This allows for designs based on the SA-1100 evaluation platform to use more memory without having to radically alter the memory map.

Flash is mapped to 128 MB (1-1 with its physical location) and is cacheable.

Note: Programming flash is not possible with caches enabled. The Data Caches must be disabled to allow the Flash memory to be written to with suitable delays.

The SRAM is **not** mapped at all. The SA-1100 supports only DRAM or SRAM, so the DRAM was chosen as the most useful common configuration. See Section 5.2 for a description of the source code to set up a different mapping.

1 MB of Static Bank 3 has been mapped to allow the external PCMCIA control register to be addressed. This area is defined as non-cacheable and non-bufferable.

1 MB of the Zeros Bank has been mapped to allow zero-wait cache flushing. This area is defined as cacheable and bufferable.

The full PCMCIA space and internal register spaces are mapped to the same addresses as the SA-1100 definitions. These areas are defined as non-cacheable and non-bufferable.

Access to any other area is reserved and will result in a Data Abort Exception.

Table 2. Virtual Memory Map

Address	End	Size	Cache[1]	Usage
0xe000 0000	0xe00f ffff	1 MB	C, B	Zeros (Clean) Bank
0xd800 0000	0xd83f ffff	4 MB	B	Dynamic RAM Bank 3
0xd000 0000	0xd03f ffff	4 MB	B	Dynamic RAM Bank 2
0xc800 0000	0xc83f ffff	4 MB	B	Dynamic RAM Bank 1
0xc000 0000	0xc03f ffff	4 MB	B	Dynamic RAM Bank 0
0x8000 0000	0xbfff ffff	1 GB	-	Internal Registers
0x3000 0000	0x3fff ffff	256 MB	-	PCMCIA Socket 1
0x2000 0000	0x2fff ffff	256 MB	-	PCMCIA Socket 0
0x1800 0000	0x180f ffff	1 MB	-	Static Bank 3 (PCMCIA)
0x0800 0000	0x0803 ffff	256 KB	C	Static Bank 1 (Flash)
0x0400 0000	0x0403 ffff	256 KB	C	Static Bank 0 (ROM)
0x00c0 0000	0x00ff ffff	4 MB	C, B	Dynamic RAM Bank 3
0x0080 0000	0x00bf ffff	4 MB	C, B	Dynamic RAM Bank 2
0x0040 0000	0x007f ffff	4 MB	C, B	Dynamic RAM Bank 1
0x0000 0000	0x003f ffff	4 MB	C, B	Dynamic RAM Bank 0

1. Cache indicates the cacheable/bufferable state of each area:
   - "C, B" means the area is cacheable and bufferable.
   - "B" means the area is bufferable, but non-cacheable.
   - "-" means the area is non-cacheable and non-bufferable.

4.0 Power On and Memory Configurations

When the StrongARM microprocessor is reset, execution starts from 0 and memory management is disabled, so the values at the start of Static Bank 0 (ROM) are executed.

4.1 ROM Execution and System Initialization

On the SA-1100, the software can set the core clock frequency. Intel recommends to only set the core clock frequency immediately following a hard reset or immediately upon wake-up from sleep mode. The core clock frequency must be set before enabling clock switching.

The executable at the start of ROM is a version of the Angel ARM** debugger with an image "bootloader" included in the startup sequence. After setting the CPU core clock to 200 MHz and the GPIOs so that GPIO1 and GPIO0 can be read, the bootloader uses these switches to select the image to be executed. The ROM and Flash are divided into logical blocks. Each image occupies one or more blocks. Due to the simple nature of the bootloader, an image cannot currently be divided between ROM and Flash. In Angel Version 1.0*X*, each block is 64 KB.

Table 3. GPIO Switch Settings

GPIO1	GPIO0	Image	Block	Location	Start Address
0	0	0	0	ROM	0 or 0x0400 0000
0	1	1	N=1,2 or 3	ROM	Image 0 + 0x(N)0000
1	0	2	4	Flash	0x0800 0000
1	1	3	N=5,6 or 7	Flash	Image 2 + 0x(N-4)0000

The Start Address is the virtual address for the start of the given block. The Flash Management Utility adds a 64-byte header at the start of the image. If the image is linked as "binary with aif header", a further 128-byte header is added. So the image must be linked for the Start Address + 0xc0 (192 decimal). If the image to be executed is not resident at its indicated start address, then the bootloader initializes memory, sets up the MMU and copies the image to that address before jumping to it.

5.0 Changing the Memory Layout

Since any image that runs from Flash initializes its own memory layout, it is easy to alter the map.

5.1 Why Use the MMU

The SA-1100 defines the memory at address 0 as static. While developing an embedded application, an MMU shell might make the programmable Flash parts appear at zero to behave like the non-programmable parts (ROM). Any dynamic application environment requires the memory at zero to be in RAM, so that the ARM exception vectors can modified. But the primary benefit is to make all of the RAM appear to the software as a contiguous block.

5.2 MMU Source Code

The MMU is initialized by Assembler code contained in brutus/target.s (for Angel; lib/brutus/target.s for uHAL); "C" obviously cannot be used, since no memory exists yet for local variables or stack. This file is a series of ARM Assembler macros that can be called by the appropriate startup file (startrom.s for Angel; lib/boot.s for uHAL).

On the SA-1100 evaluation platform, the INIT_RAM macro calls INITMMU, which in turn calls SETUPMMU. INIT_RAM checks the SRAM/DRAM control switch and initializes the memory subsystem accordingly, but all subsequent macros currently assume that DRAM was selected. Hence, Angel does not currently run with SRAM enabled.

All of the relevant address definitions are contained in platform.s (in brutus/ for Angel; h/brutus for uHAL). The readability of this code was a primary concern, so each area is clearly commented and examples of level 2 page tables are included.

5.3 The "Dance of Death"

Applications that run from ROM when the MMU is activated are built for addresses 64 MB and above. However, the actual code for these applications executes from physical address 0, as shown in Table 1.

The virtual address space to be defined by the page tables must be consistent with the virtual ranges as described in Table 2. However, the following recommendations should be observed while building the page tables.

1. It is recommended that the page tables themselves reside in the range of addresses between 0xc0000000 and 0xd8400000. The base address of the level 1 portion of the page tables is loaded into the Table Translation Base register by the program.
2. The executing program must remain accessible immediately following enabling the MMU. If the page tables were to define virtual space exactly as shown inTable 2, then enabling the MMU would cause the system to crash since the executing program’s instruction stream would no longer be accessible. This is because the Program Counter (PC) is not remapped while enabling the MMU.

Evading the "Dance of Death" involves changing the page table entries for the current area of memory so that the ROM remains mapped from 0 when the MMU is enabled. Once the MMU is activated, the PC is moved to the mirrored virtual ROM address and the original page table entries can be restored.

The SETUPMMU macro checks that the program is executing from ROM space and that the MMU is not already enabled. If either of these tests fail, the simple final memory map is defined. Otherwise, the "Dance of Death" is performed (if the programmer gets this wrong, the program will probably be unable to find its instructions and almost certainly will crash).

Finally, flush the processor address lookup, instruction, and data caches. StrongARM caches the most recent page table entries as well as data and instructions. So in order to access the new entry correctly, this last step is crucial.

5.4 Memory Sizing

There is currently no autosizing code in the INIT_RAM macro. This macro returns the memory size to Angel and uHAL. In order to use the SRAM, SETUPMMU must check the SRAM/DRAM control switch and map the SRAM at 0 instead of the DRAM. Then INIT_RAM can check the control switch and return the smaller size if SRAM is active.

6.0 Summary

Any image that executes from dynamic memory must be sure to avoid re-initializing that memory - the uHAL library checks the MMU state to ensure that memory really needs to be initialized.

All images that execute from Flash or ROM start with no memory subsystem, no memory management enabled, and no clock switching.

Standard memory and MMU settings can be automatically set up by using the uHAL library, allowing rapid prototyping and code development.

Support, Products, and Documentation

If you need technical support, a *Product Catalog*, or help deciding which documentation best meets your needs, visit the Intel World Wide Web Internet site:
`http://www.intel.com`

Copies of documents that have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling **1-800-332-2717** or by visiting Intel's website for developers at:
`http://developer.intel.com`

You can also contact the Intel Massachusetts Information Line or the Intel Massachusetts Customer Technology Center. Please use the following information lines for support:

For Documentation and General Information	
Intel Massachusetts Information Line	
United States:	1–800–332–2717
Outside United States:	1–303-675-2148
Electronic mail address:	techdoc@intel.com

For Technical Support	
Intel Massachusetts Customer Technology Center	
Phone (U.S. and international):	1–978–568–7474
Fax:	1–978–568–6698
Electronic mail address:	techsup@intel.com

ARM POWERED
StrongARM™