

4 Memory Management Unit

The ETRAX 100LX includes a Memory Management Unit (MMU) that manages the physical memory resources of the device. The MMU has a number of purposes:

- Protecting the code and data of one user-level application from other user-level applications.
- Permitting user-level applications to share portions of the address space.
- Protecting the kernel-level code and data from user-level applications.
- Running applications that are partially resident in main memory. Only the most recent part of such an application is normally stored in main memory; the rest of the program is stored on disk until needed. This is more commonly known as swapping.
- Reference counting in support of a garbage collection mechanism.

The MMU implements a virtual memory system where it creates the illusion of a very large amount of memory exclusively available for each application.

In ETRAX 100LX, 4 GBytes of virtual memory are made available to each application. Each application is also given a separate address space identifier that is used by the MMU to separate its memory from other applications.

The virtual memory is divided into 8 KByte pages which can be individually protected and mapped to physical memory. The upper 19-bit part of the 32-bit CPU address is known as the virtual page number (vpn), while the lower 13-bit part is used as an offset within each page. The vpn is translated into a 19-bit physical page number (pfn) while the page offset remains unchanged through the MMU. The MMU uses the CPU's *User* and *Supervisor Modes* to restrict access and select the appropriate mapping from logical to physical addresses in the virtual memory space.

Mapping from virtual address to physical address is handled by a *Translation Lookaside Buffer* (TLB). The TLB is an on-chip cache that provides translations in the form of page table entries (pte). The ptes are stored in page tables in main memory. If a virtual-to-physical address translation is not found in the TLB, the MMU will interrupt the CPU and use a software table walker to look for the translation in the main memory page table.

In some cases the MMU is not used, so it can be disabled in register `R_MMU_ENABLE`. By default the MMU is disabled after reset. When the MMU is disabled, the ETRAX 100LX is considered to be in *non-protected* mode and memory is accessed in the conventional manner.

Please see chapter 18.17 *MMU Registers* for detailed information on the MMU registers.

4.1 MMU Memory Areas

The MMU operates with two types of memory area:

- **Kernel/user area** - accessed in the CPU User and Supervisor Modes, and mainly contains user code and data structures.
- **Kernel area** - accessed in the CPU Supervisor Mode only and may include kernel code and data structures, I/O-buffers, DMA-buffers, mode registers, etc.

4.1.1 Kernel/User Address Space

The *kernel/user* address space is a uniform, virtual address area of 4 GBytes in size. It is divided into 8 KByte pages that can be individually protected and mapped to physical memory. Mapping is executed through the TLB, which translates a virtual address into a corresponding physical address. Each user process has a unique translation of virtual addresses placed in page tables in main memory. The page tables are maintained by the kernel.

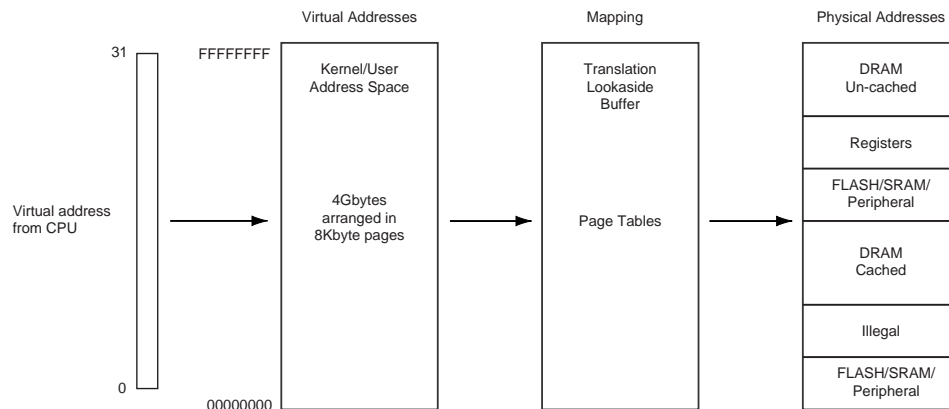


Figure 4-1 Kernel/User Virtual Address Space

4.1.2 Kernel Address Space

The kernel address space is divided into sixteen 256 MByte segments designated **seg_0** to **seg_f**. Each segment can be individually configured to use page mapping or linear segment mapping. The mapping method is determined in kernel segment register **R_MMU_KSEG**. When a bit in this register is set to 0, the corresponding segment is page-mapped via the TLB. When a segment bit is set to 1, the corresponding segment is linear mapped by means of a 4-bit offset. Linear-mapped segments do not use the TLB.

Page-Mapped Kernel Segments

Kernel address segments that use page mapping are divided into 8 KByte pages controlled by the TLB. The four-bit base fields in registers **R_MMU_KBASE_LO** and **R_MMU_KBASE_HI** are ignored for page mapping purposes.

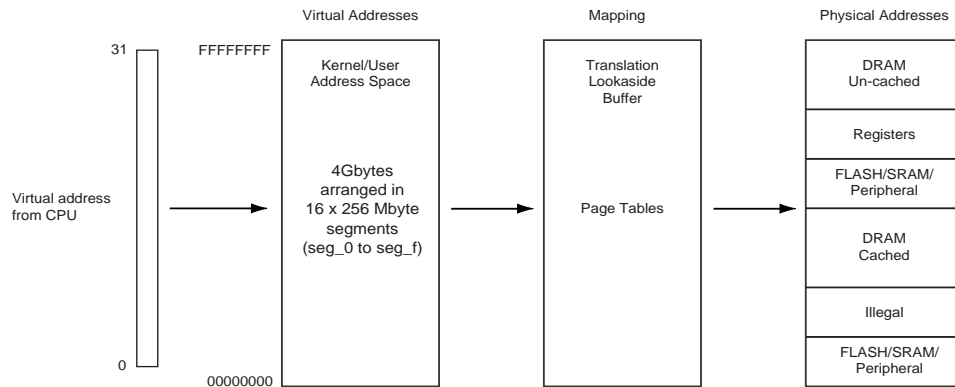


Figure 4-2 Kernel Page-mapped Virtual Address Space

Linear Segment-Mapped Kernel Areas

The virtual addresses of kernel segments that use linear mapping are converted to physical addresses by translating the four m.s.b. of the 32-bit virtual address from the CPU. The linear translation is handled in two registers, R_MMU_KBASE_HI and R_MMU_KBASE_LO. The 4-bit base field in these registers becomes address bits 31 to 28 when translating to physical addresses. Bits 27 to 0 of the virtual address are unchanged when translated to a physical address.

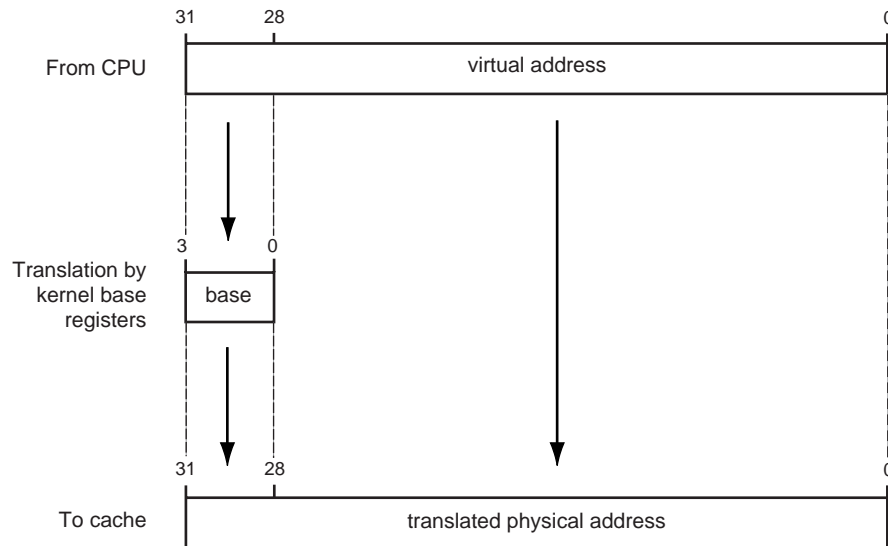


Figure 4-3 Linear Segment Address Translation

4.2 Translation Lookaside Buffer

The Translation Lookaside Buffer is a 64-entry cache that maps a virtual address to a physical address. If a translation cannot be found in the TLB (a *TLB miss*), the CPU is interrupted (chapter 2, *refers*) and the software is required to load a new translation into the TLB.

4.2.1 TLB Memory Sets

The TLB comprises four 16-entry memory sets designated 0 to 3. The TLB is thus four-set associative. This means that a vpn from the CPU can be stored at only one location in each of the four TLB memory sets, and in each set the chosen location is the same (see Figure 4.4).

An example of possible locations for different addresses in the TLB is given in the table below.

| VPN | Positions in TLB | | | |
|-----|------------------|----|----|----|
| 0 | 0 | 16 | 32 | 48 |
| 1 | 1 | 17 | 33 | 49 |
| 2 | 2 | 18 | 34 | 50 |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| 14 | 14 | 30 | 46 | 62 |
| 15 | 15 | 31 | 47 | 63 |
| 16 | 0 | 16 | 32 | 48 |
| 17 | 1 | 17 | 33 | 49 |

Table 4-1 Example of Virtual Address Positions in the TLB

The selection of the TLB set in which a particular vpn will be stored must be configured in software. The location at which the vpn will be placed within the TLB set is determined by a 4-bit index comprising bits 13 to 16 of the incoming virtual address from the CPU.

In the event of a TLB miss, a 2-bit set number is provided by a random number generator. The random set number is combined with the 4-bit index from the CPU address to form a 6-bit index. This index is stored in register R_TLB_SELECT, and can be used to choose which of the 64 TLB entries to replace. It is also possible for the software to use another algorithm to select the entry to replace. As shown below, bits 4 and 5 of the index provide the TLB set number and bits 3 to 0 denote the location within the set.

| set | | location | | | |
|-----|---|----------|---|---|---|
| 5 | 4 | 3 | 2 | 1 | 0 |

Figure 4-4 Format of TLB Miss Exception Index

To accommodate the 8 KByte page size, the lower 13 bits of the 32-bit virtual address from the CPU are not changed by the translation. The 19-bit virtual page number is translated into a 19-bit physical frame number.

The principle of the TLB memory sets is illustrated below.

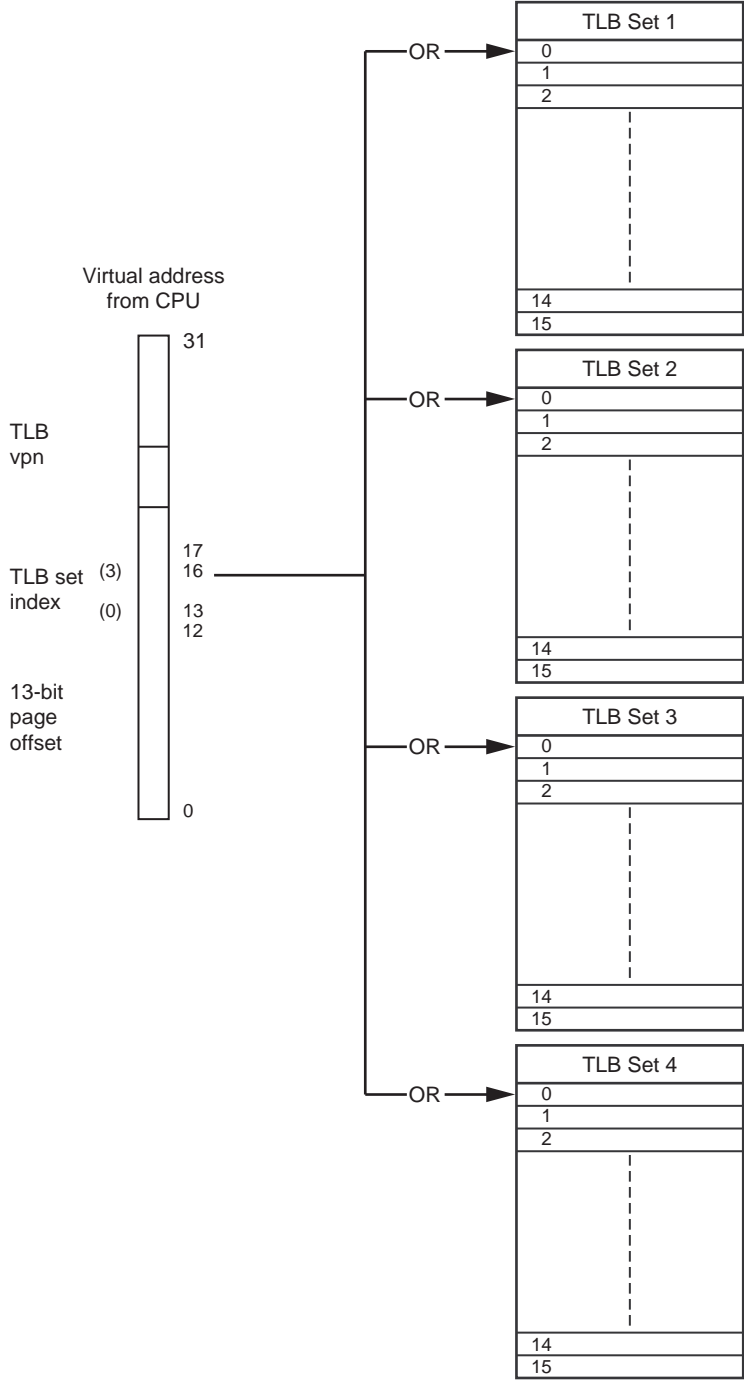


Figure 4-5 TLB Memory Sets

4.2.2 TLB Entries

Entries stored in the TLB are 44 bits in width. They consist of a virtual to physical address translation, a page identification, and control bits. The format of a TLB entry is shown below.

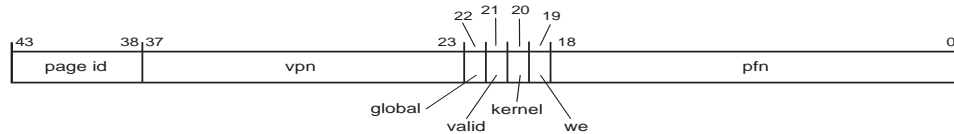


Figure 4-6 Format of a TLB Entry

- **pfn** - the translated address expressed as a 19-bit physical frame number;
- **vpn** - the virtual address expressed as a 15-bit virtual page number: the remaining 4 bits are composed from the 4-bit TLB set index.
- **page_id** - the 6-bit page identification;
- **we** - write enable bit 19 is used to write-protect the page;
- **kernel** - bit 20 is used to prevent access to the page during CPU User Mode;
- **valid** - bit 21 indicates that the TLB entry contains a valid address translation;
- **global** - bit 22 indicates whether or not the TLB ignores the page identification matching conditions for a TLB hit.

If the **global** bit is not asserted (0), the **page_id** in the TLB entry is compared to the value of the 6-bit **page_id** in register R_MMU_CONTEXT. To achieve a valid translation, the values of the **page_id** fields in the TLB entry and the register must match.

If the **global** bit is asserted (1), the **page_id** of the TLB entry is not compared to the **page_id** field in register R_MMU_CONTEXT.

The condition for a hit in one of the TLB sets is:

```
hit = (valid | inv_excp) &
      (cpu_vpn == tlb_vpn) &
      ((context_pid == tlb_pid) | global))
```

where:

| | |
|--------------------|--|
| valid | valid bit (21) in the TLB entry. |
| inv_excp | invalid page exception enabling bit from R_MMU_CTRL. |
| cpu_vpn | virtual page number from the CPU. |
| tlb_vpn | virtual page number in the TLB entry. |
| context_pid | page_id field in R_MMU_CONTEXT. |
| tlb_pid | page_id field in the TLB entry. |
| global | global bit (22) in the TLB entry. |

Table 4-2 Definitions of TLB Hit

4.2.3 TLB Register Interface

The TLB is controlled by registers R_TLB_SELECT, R_TLB_LO and R_TLB_HI, all entries to which can be read and written by the CPU. Register R_TLB_SELECT is used to choose which entry is to be read or written in the TLB.

TLB entries are 44 bits in width and therefore an entry cannot be written by the CPU in one cycle. The CPU must first write the high order part of the TLB entry to register R_TLB_HI. This contains the same fields as register R_MMU_CAUSE and, when writing to R_TLB_HI, the data are also stored at the corresponding fields of R_MMU_CAUSE. The high order part is not stored in the TLB until the low order part is written in register R_TLB_LO.

Registers R_MMU_CAUSE and R_TLB_SELECT are normally updated automatically by the MMU and do not require updating by the software. To write a new translation in the TLB, for example a TLB miss, the software only has to write into register R_TLB_LO.

4.2.4 Virtual Address from the CPU

An incoming address from the CPU is constructed as shown below.

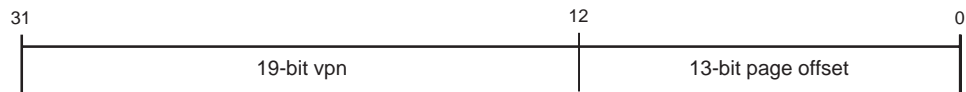


Figure 4-7 Incoming Address from the CPU

The page offset is 13 bits wide to accommodate the offset within the 8 KByte pages. The upper 15 bits of the vpn are stored in the TLB and the lower four bits are the address index for the selected TLB set.

When the CPU generates a new address, the four TLB sets are searched for a matching vpn. If an entry is found, the **page_id** is compared to the corresponding field in register R_MMU_CONTEXT. The **valid**, **kernel** and **we** bits are controlled and, if all conditions match, a valid physical address is output to the cache.

The TLB is not permitted to store more than one valid, virtual page translation with the same **page_id** or with the **global** bit set. This would cause multiple hits in the TLB and result in an MMU exception.

4.2.5 MMU Exceptions

An exception is generated if there is a mismatch in the output from one or more fields of the TLB. In the event of an exception, the MMU interrupts the CPU by means of bus fault logic (See chapter 2 *RISC CPU*). Information about the exception is stored in register R_MMU_CAUSE, which holds information about:

- The vpn that generated the exception
- The **page_id** of the application that generated the exception
- Whether the exception was caused by a write or read access

- The type of exception that occurred

Five different types of MMU exception can occur (Refer to section *4.2.1 TLB Memory Sets* for those exceptions that generate a random or actual index):

- **Miss** - The referenced address does not match any TLB entry, or the current **page_id** does not match the TLB index. A valid entry must be loaded by software.
- **Write error** - During a write operation, reference is made to a page that does not have the **we** bit asserted in the TLB entry. This exception can be used for write protection and dirty checks.
- **Access violation** - In User Mode, reference is made to a page with the **kernel** bit asserted in the TLB entry. When asserted, the **kernel** bit prevents CPU User Mode access to the page.
- **Invalid page** - Reference is made to a page with matching **vpn** and **page_id** fields in the TLB, but the **valid** bit is deasserted. This can be used for reference counting.
- **Multiple hits** - The occurrence of more than one hit in the TLB indicates a serious error. This is indicated by a bus fault signal with all exception bits deasserted.

The write error, access violation and invalid page exceptions can all be disabled in control register **R_MMU_CTRL**. Normally the invalid page exception is disabled unless a reference count is in progress. When disabled, an invalid entry in the TLB will not match any address and will, thus, cause a miss.

The nature of a miss exception is such that it does not allow any other exceptions to occur simultaneously. However, a single memory reference may cause any combination of write error, access violation and invalid page exceptions at the same time.

The conditions for the MMU exceptions are:

- Miss = \sim hit
- Write error = hit & cpu_wr & \sim we & we_exc
- Access violation = hit & user_mode & kernel & acc_exc
- Invalid page = hit & \sim valid & inv_exc

where:

| | |
|------------------|--|
| hit | Hit in one of the TLB sets. |
| cpu_wr | CPU write access. |
| we | Write enable bit in TLB entry. |
| we_exc | Write error exception enabling bit from R_MMU_CTRL. |
| user_mode | CPU User Mode access. |
| kernel | Kernel bit in TLB entry. |
| acc_exc | Access violation exception enabling bit from R_MMU_CTRL. |
| valid | Valid bit in TLB entry. |
| inv_exc | Invalid page exception enabling bit from R_MMU_CTRL. |

Table 4-3 Definition of MMU Exceptions

4.3 MMU Registers

The MMU is served by a set of dedicated registers. The table below summarizes the purpose of each register.

| Register | Function |
|--------------------------|---|
| R_MMU_CONFIG comprising: | |
| R_MMU_KSEG | Sets individual page or segment mapping method for each kernel segment 0 to f. |
| R_MMU_CTRL | Enables or disables the invalid page (valid), access (kernel) and write-enable (we) MMU exceptions. |
| R_MMU_ENABLE | Enables or disables the MMU. |
| R_MMU_KBASE_LO | Provides the 4-bit offset for linear translations in the eight lower kernel segments 0 to 7. |
| R_MMU_KBASE_HI | Provides the 4-bit offset for linear translations in the eight upper kernel segments 8 to f. |
| R_MMU_CONTEXT | Contains the 6-bit page_id of the current address map. |
| R_MMU_CAUSE | Multi-purpose, containing: <ul style="list-style-type: none"> the 19-bit vpn of the referenced address that generated an exception when an MMU exception occurs; the 6-bit page_id from R_MMU_CONTEXT when an MMU exception occurs; 4 discrete bits signifying whether a miss, invalid, access or write-enable exception has occurred; 1 bit signifying whether the exception was caused by a write or read access. |
| R_TLB_SELECT | Contains the 6-bit TLB index. |
| R_TLB_LO | Contains the lower 23 bits of the TLB entry, namely: <ul style="list-style-type: none"> 19-bit pfn; global bit; valid bit; kernel bit; we bit. |
| R_TLB_HI | Contains the upper 25 bits of the TLB entry, namely: <ul style="list-style-type: none"> 19-bit vpn; 6-bit page_id. |

Table 4-4 MMU Registers

For detailed information on the MMU registers, please refer to chapter 18.17 *MMU Registers*.

4.4 MMU Test Mode

The MMU can be set to a test mode that can be used to manually check the TLB and the bus fault logic. The MMU test mode is set by bit `mmu_test` in register `R_TEST_MODE`.

When the MMU test mode is active, the bus fault logic is operative but the bus fault signal to the CPU is gated and cannot, therefore, generate an interrupt to the CPU. Register `R_MMU_CAUSE` is updated as normal.

The address translation logic is not affected by the MMU test mode. If the MMU is enabled, mapping through the TLB and offset registers operates in the normal way. If the MMU is disabled, the output physical address is not translated.

| <code>mmu_en</code> | <code>mmu_test</code> | Interrupt CPU | Update <code>R_MMU_CAUSE</code> | Translate Address |
|---------------------|-----------------------|---------------|---------------------------------|-------------------|
| 0 | 0 | no | no | no |
| 0 | 1 | no | yes | no |
| 1 | 0 | yes | yes | yes |
| 1 | 1 | no | yes | yes |

Table 4-5 MMU Test Mode Truth Table

4.5 Example of Virtual Memory Configuration

Virtual memory configuration is essentially a matter of setting up the ratio of page-mapped kernel space to linear-mapped kernel space. For example, to set up the following virtual memory system, the MMU register configurations would resemble those described in the tables below.

- 0.5 GBytes of linear-mapped kernel area;
- 3.5 GBytes of page-mapped kernel area;
- 4 GBytes of kernel/user area.

Register R_MMU_KBASE_HI

| Bit(s) | Name | Setting | Value |
|---------|--------|---------------------------------|-------|
| 31 - 28 | base_f | Don't care. | 0 x 0 |
| 27 - 24 | base_e | Don't care. | 0 x 0 |
| 23 - 20 | base_d | Don't care. | 0 x 0 |
| 19 - 16 | base_c | Kernel cached area. | 0 x 7 |
| 15 - 12 | base_b | Kernel uncached mode registers. | 0 x b |
| 11 - 8 | base_a | Don't care. | 0 x 0 |
| 7 - 4 | base_9 | Don't care. | 0 x 0 |
| 3 - 0 | base_8 | Don't care. | 0 x 0 |

Table 4-6 Example of Kernel Base High Register Configuration

Register R_MMU_KBASE_LO

| Bit(s) | Name | Setting | Value |
|---------|--------|-------------|-------|
| 31 - 28 | base_7 | Don't care. | 0 x 0 |
| 27 - 24 | base_6 | Don't care. | 0 x 0 |
| 23 - 20 | base_5 | Don't care. | 0 x 0 |
| 19 - 16 | base_4 | Don't care. | 0 x 0 |
| 15 - 12 | base_3 | Don't care. | 0 x 0 |
| 11 - 8 | base_2 | Don't care. | 0 x 0 |
| 7 - 4 | base_1 | Don't care. | 0 x 0 |
| 3 - 0 | base_0 | Don't care. | 0 x 0 |

Table 4-7 Example of Kernel Base Low Register Configuration

Register R_MMU_KSEG

| Bit(s) | Name | Setting | Value |
|--------|-------|--------------------------------|-------|
| 15 | seg_f | Page mapping. | 0 |
| 14 | seg_e | Page mapping. | 0 |
| 13 | seg_d | Page mapping. | 0 |
| 12 | seg_c | Kernel linear segment mapping. | 1 |
| 11 | seg_b | Kernel linear segment mapping. | 1 |
| 9 | seg_a | Page mapping. | 0 |
| 8 | seg_9 | Page mapping. | 0 |
| 7 | seg_7 | Page mapping. | 0 |
| 6 | seg_6 | Page mapping. | 0 |
| 5 | seg_5 | Page mapping. | 0 |
| 4 | seg_4 | Page mapping. | 0 |
| 3 | seg_3 | Page mapping. | 0 |
| 2 | seg_2 | Page mapping. | 0 |
| 1 | seg_1 | Page mapping. | 0 |
| 0 | seg_0 | Page mapping. | 0 |

Table 4-8 Example of Kernel Segment Register Configuration

Memory maps of the virtual address spaces realized by the example configuration are illustrated below.

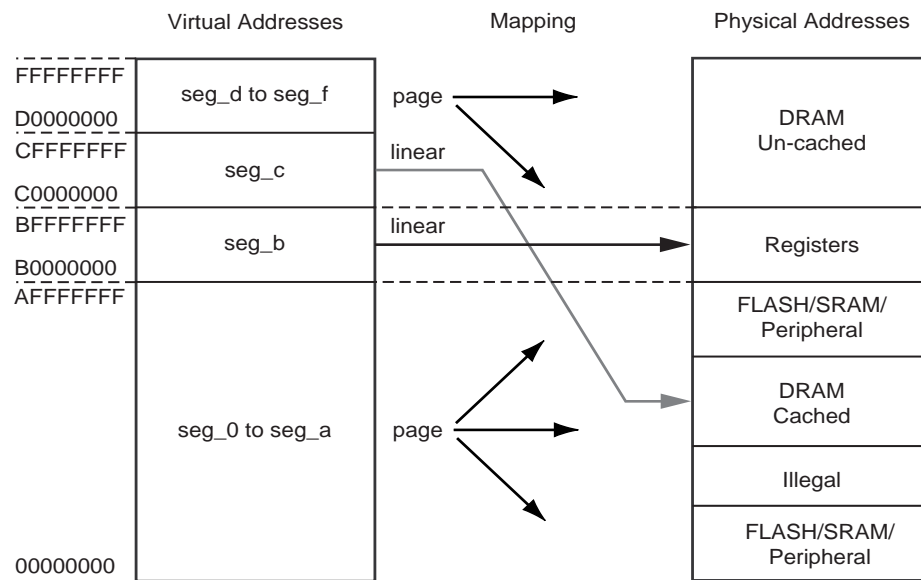


Figure 4-8 Example of Kernel Virtual Memory Map

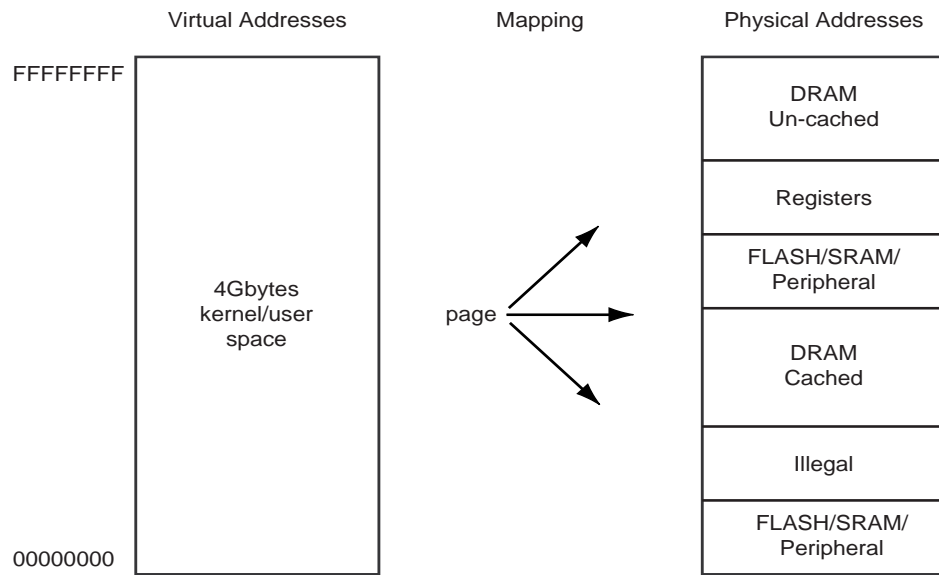


Figure 4-9 Example of Kernel/User Virtual Memory Map