8 Memory Management Unit

This chapter describes the *Memory Management Unit* (*MMU*).

Memory Management Unit

The Memory Management MMU performs two primary functions: it translates virtual addresses into physical addresses, and it controls memory access permissions. The MMU hardware required to perform these functions consists of a Translation Look-aside Buffer (TLB), access control logic, and translation table walking logic.

The MMU supports memory accesses based on Sections or Pages. Sections are comprised of 1MB blocks of memory. Two different page sizes are supported: Small Pages consist of 4KB blocks of memory and Large Pages consist of 64KB blocks of memory. (Large Pages are supported to allow mapping of a large region of memory while using only a single entry in the TLB). Additional access control mechanisms are extended within Small Pages to 1KB Sub-Pages and within Large Pages to 16KB Sub-Pages.

The MMU also supports the concept of domains - areas of memory that can be defined to possess individual access rights. The Domain Access Control Register is used to specify access rights for up to 16 separate domains.

The TLB caches 64 translated entries. During most memory accesses, the TLB provides the translation information to the access control logic.

If the TLB contains a translated entry for the virtual address, the access control logic determines whether access is permitted. If access is permitted and an off-chip access is required, the MMU outputs the appropriate physical address corresponding to the virtual address. If access is not permitted, the MMU signals the CPU to abort.

If the TLB misses (it does not contain a translated entry for the virtual address), the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. Once retrieved, the translation information is placed into the TLB, possibly overwriting an existing value. The entry to be overwritten is chosen by cycling sequentially through the TLB locations.

When the MMU is turned off (as happens on reset), the virtual address is output directly onto the physical address bus.

8.1 MMU Program Accessible Registers

The following ARM810 System Control Coprocessor (CP15) registers, in conjunction with page table descriptors stored in memory, determine the operation of the MMU.

Register	Number	Bits
Control Register	1	M,A,S,R
Translation Table Base	2	31 .. 14
Domain Access Control	3	31 .. 0
Fault Status	5	8 .. 0
Fault Address	6	31 .. 0
TLB Operations	8	31 .. 0
TLB Lock down Control	10	31 & 5 .. 0

Table 8-1: CP15 register functions

All of these registers except register 8 contain state and can be read using MRC instructions and written using MCR instructions. Registers 5 and 6 are also written by the MMU when a data abort is signaled to record the cause of, and address associated with, an Abort. Writing to Register 8 causes the MMU to perform one of the TLB operations “Invalidate TLB” or “Invalidate TLB Entry”. Register 8 does not contain state and cannot be read.

Depending on the coprocessor instruction used, writing to register 8 with an MCR instruction causes one of the TLB operations “Invalidate TLB” or “Invalidate TLB Entry” to be performed by the MMU. Register 8 does not contain state and cannot be read.

System Control Coprocessor is described in ***Chapter 5, Configuration***. The details of register format and the coprocessor instructions to access them are given there.

A brief description of these registers is provided below. Each register will be discussed in more detail within the section that describes its use.

The **Control Register** contains bits to enable the MMU (M bit), enable Alignment checks (A bit), and to control the access protection scheme (S bit and R bit).

The **Translation Table Base Register** hold the physical address of the base of the translation table maintained in main memory. Note that this base must reside on a 16KB boundary.

The **Domain Access Control Register** consists of sixteen 2-bit fields, each of which defines the access permissions for one of sixteen Domains (D15-D0).

The **Fault Status Register** indicates the cause of an abort and the domain number of the aborted access when a data abort occurs. Bits 7:4 specify which of the sixteen domains (D15-D0) was being accessed when a fault occurred. Bits 3:1 indicate the type of access being attempted. The encoding of these bits is shown in ***Table 8-6: Priority Encoding of Fault Status*** on page 8-17.

The **Fault Address Register** holds the virtual address associated with the access that caused with abort. See ***Table 8-6: Priority Encoding of Fault Status*** on page 8-17 for details of exactly what address is stored for each type of fault.

Writing to the **TLB Operations Register** causes the MMU to perform one of the TLB operations “Invalidate TLB” or “Invalidate TLB Entry” depending on the coprocessor instruction used. For details, see the description of Register 8 in ***Chapter 5, Configuration***.

The **TLB Lock-Down Control Register** allows specific page table entries to be locked into the TLB. Locking entries in the TLB guarantees that accesses to the locked page or section can proceed without incurring the time penalty of a translation table walk. This allows the execution latency for time-critical pieces of code such as interrupt handlers to be minimised. Use of the TLB lock down facilities is described in ***Chapter 7, Instruction and Data Cache (IDC)***.

8.2 Address Translation

The MMU translates virtual addresses generated by the CPU into physical addresses to access external memory, and also derives and checks the access permission. Translation information, which consists of both the address translation data and the access permission data, resides in a translation table located in physical memory. The MMU provides the logic needed to traverse this translation table, obtain the translated address, and check the access permission.

There are three routes by which the address translation (and hence permission check) takes place. The route taken depends on whether the address in question has been marked as a section-mapped access or a page-mapped access; and there are two sizes of page-mapped access (large pages and small pages). However, the translation process always starts out in the same way, as described below, with a Level One fetch. A section-mapped access only requires a Level One fetch, but a page-mapped access also requires a Level Two fetch.

8.3 Translation Process

8.3.1 Translation table base

The translation process is initiated when the on-chip TLB does not contain an entry for the requested virtual address. The Translation Table Base (TTB) Register points to the base of a table in physical memory which contains Section and/or Page descriptors. The 14 low-order bits of the TTB Register are set to zero as illustrated in ***Figure 8-1: Translation table base register***, the table must reside on a 16KB boundary.

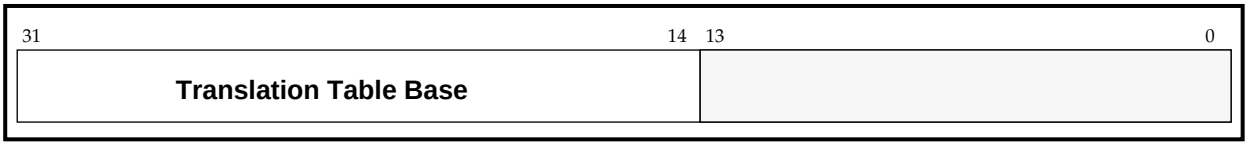

Figure 8-1: Translation table base register

8.3.2 Level one fetch

Bits 31:14 of the Translation Table Base register are concatenated with bits 31:20 of the virtual address to produce a 30-bit address as illustrated in ***Figure 8-2: Accessing the translation table first level descriptors***. This address selects a four-byte translation table entry which is a First Level Descriptor for either a Section or a Page (bit1 of the descriptor returned specifies whether it is for a Section or Page)

.

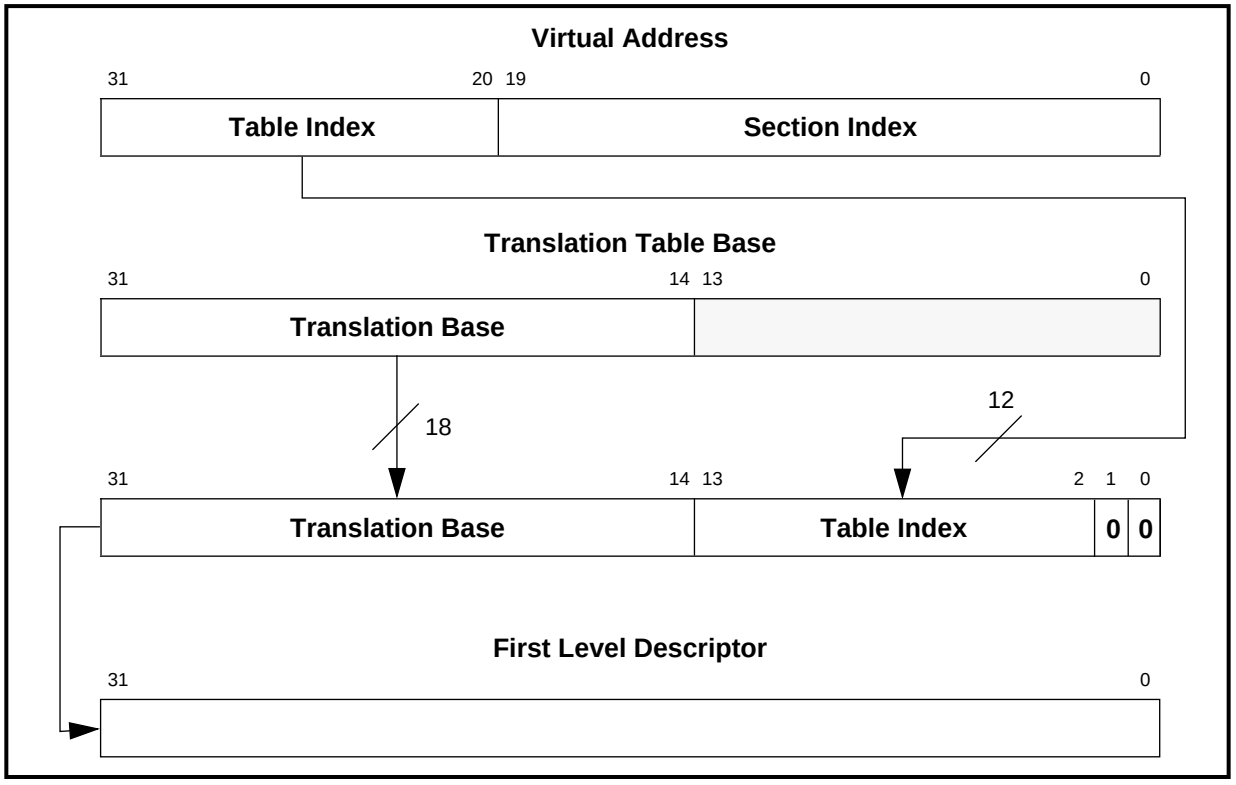

Figure 8-2: Accessing the translation table first level descriptors

8.4 Level One Descriptor

The Level One Descriptor returned is either a Page Table Descriptor or a Section Descriptor, and its format varies accordingly. The following figure illustrates the format of Level One Descriptors.

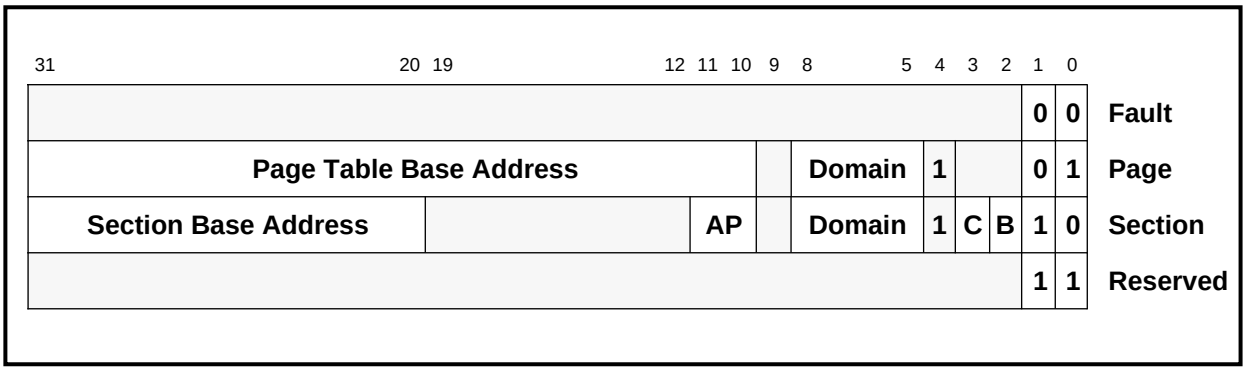

Figure 8-3: Level one descriptors

The two least significant bits indicate the descriptor type and valididty, and are interpreted as shown below..

Value	Meaning	Notes
0 0	Invalid	Generates a Section Translation Fault
0 1	Page	Indicates that this is a Page Descriptor
1 0	Section	Indicates that this is a Section Descriptor
1 1	Reserved	Reserved for future use

Table 8-2: Interpreting level one descriptor bits [1:0]

8.5 Page Table Descriptor

Bits 3:2 are always written as 0.

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 31:10 form the base for referencing the Page Table Entry. (The page table index for the entry is derived from the virtual address as illustrated in ***Figure 8-6: Small page translation*** on page 8-12).

If a Page Table Descriptor is returned from the Level One fetch, a Level Two fetch is initiated as described below.

8.6 Section Descriptor

Bits 3:2 (C, & B) The C & B bits together indicate whether the area of memory mapped by this section is treated as write-back cacheable, write-through cacheable, non cached buffered or non-cached non-buffered. Reference section 7.1.1 Cacheable and Bufferable Status of Memory Regions.

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 11:10 (AP) specify the access permissions for this section and are interpreted as shown in ***Table 8-3: Interpreting access permission (AP) Bits*** on page 8-9. Their interpretation is dependent upon the setting of the S and R bits (control register bits 8 and 9). Note that the Domain Access Control specifies the primary access control; the AP bits only have an effect in client mode. Refer to section on access permissions

AP	S	R	Permissions Supervisor	User	Notes
00	0	0	No Access	No Access	Any access generates a permission fault
00	1	0	Read Only	No Access	Supervisor read only permitted
00	0	1	Read Only	Read Only	Any write generates a permission fault
00	1	1	Reserved		
01	x	x	Read/Write	No Access	Access allowed only in Supervisor mode
10	x	x	Read/Write	Read Only	Writes in User mode cause permission fault
11	x	x	Read/Write	Read/Write	All access types permitted in both modes.
xx	1	1	Reserved		

Table 8-3: Interpreting access permission (AP) Bits

Bits 19:12 are always written as 0.

Bits 31:20 form the corresponding bits of the physical address for the 1MByte section.

8.7 Translating Section References

Figure 8-4: Section translation illustrates the complete Section translation sequence. Note that the access permissions contained in the Level One Descriptor must be checked before the physical address is generated. The sequence for checking access permissions is described below.

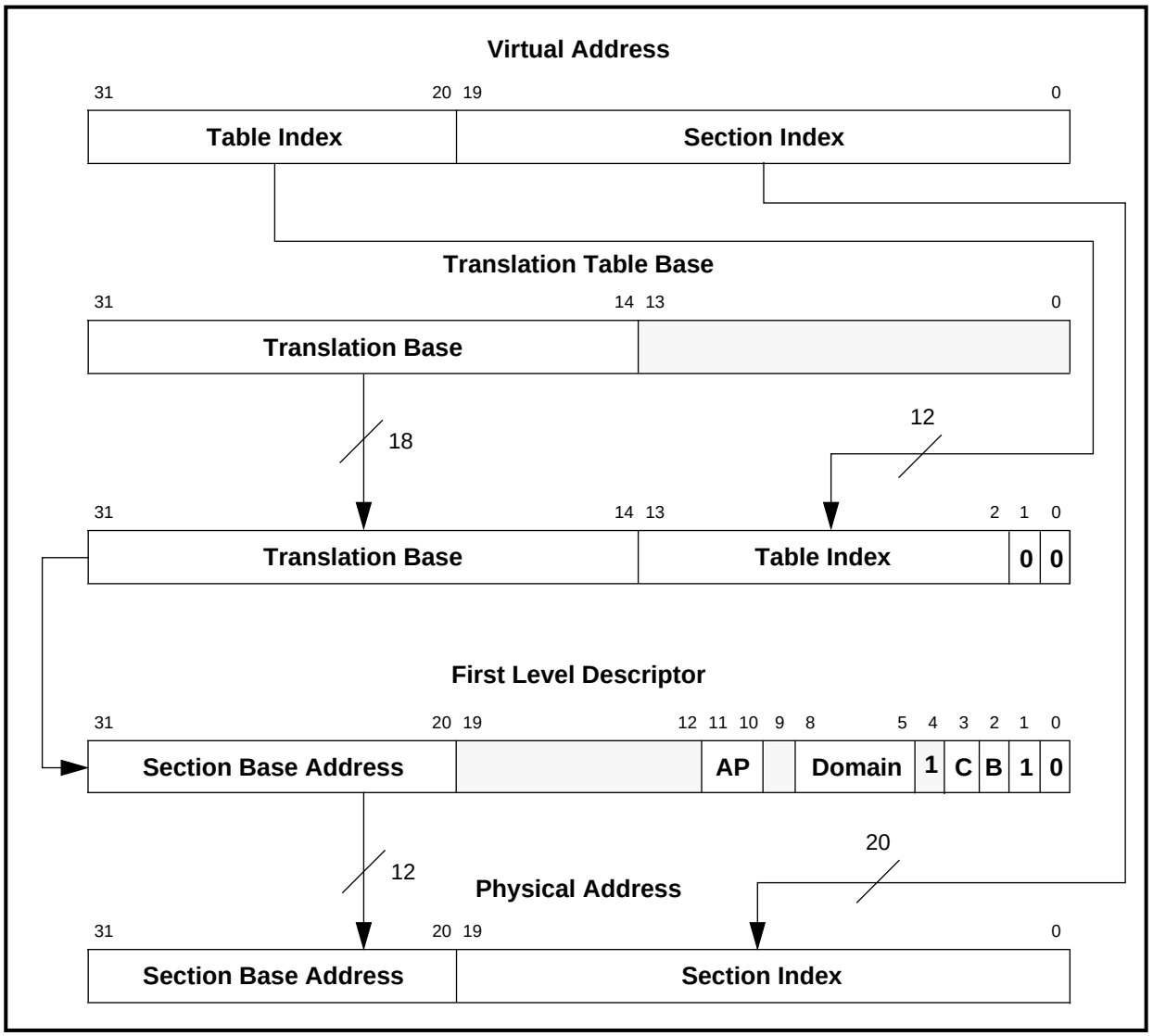

Figure 8-4: Section translation

8.8 Level Two Descriptor

If the Level One fetch returns a Page Table Descriptor, this provides the base address of the page table to be used. The page table is then accessed as described in ***Figure 8-6: Small page translation*** on page 8-12, and a Page Table Entry, or Level Two Descriptor, is returned. This in turn may define either a Small Page or a Large Page access. The figure below shows the format of Level Two Descriptors

.

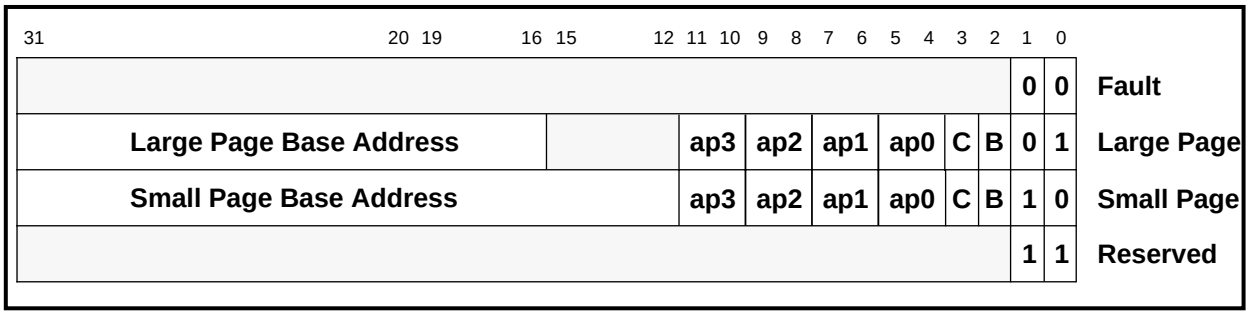

Figure 8-5: Page table entry (level two descriptor)

The two least significant bits indicate the page size and validity, and are interpreted as follows.

Value	Meaning	Notes
0 0	Invalid	Generates a Page Translation Fault
0 1	Large Page	Indicates that this is a 64 KB Page
1 0	Small Page	Indicates that this is a 4 KB Page
1 1	Reserved	Reserved for future use

Table 8-4: Interpreting page table entry Bits 1:0

Bit 3:2 (C : B) - The C & B bits together indicate whether the area of memory mapped by this section is treated as write-back cacheable, write-through cacheable, non cached buffered or non-cached non-buffered. Reference section 7.1.1 Cacheable and Bufferable Status of Memory Regions.

Bits 11:4 specify the access permissions (ap3 - ap0) for the four sub-pages and interpretation of these bits is described earlier in ***Table 8-2: Interpreting level one descriptor bits [1:0]*** on page 8-7.

For large pages, **bits 15:12** are programmed as 0.

Bits 31:12 (small pages) or bits **31:16** (large pages) are used to form the corresponding bits of the physical address - the physical page number. (The page index is derived from the virtual address as illustrated in ***Figure 8-6: Small page translation*** on page 8-12 and ***Figure 8-7: Large page Ttanslation*** on page 8-13).

8.9 Translating Small Page References

Figure 8-6: Small page translation illustrates the complete translation sequence for a 4KB Small Page. Page translation involves one additional step beyond that of a section translation: the Level One descriptor is the Page Table descriptor, and this is used to point to the Level Two descriptor, or Page Table Entry. (Note that the access permissions are now contained in the Level Two descriptor and must be checked before the physical address is generated. The sequence for checking access permissions is described later).

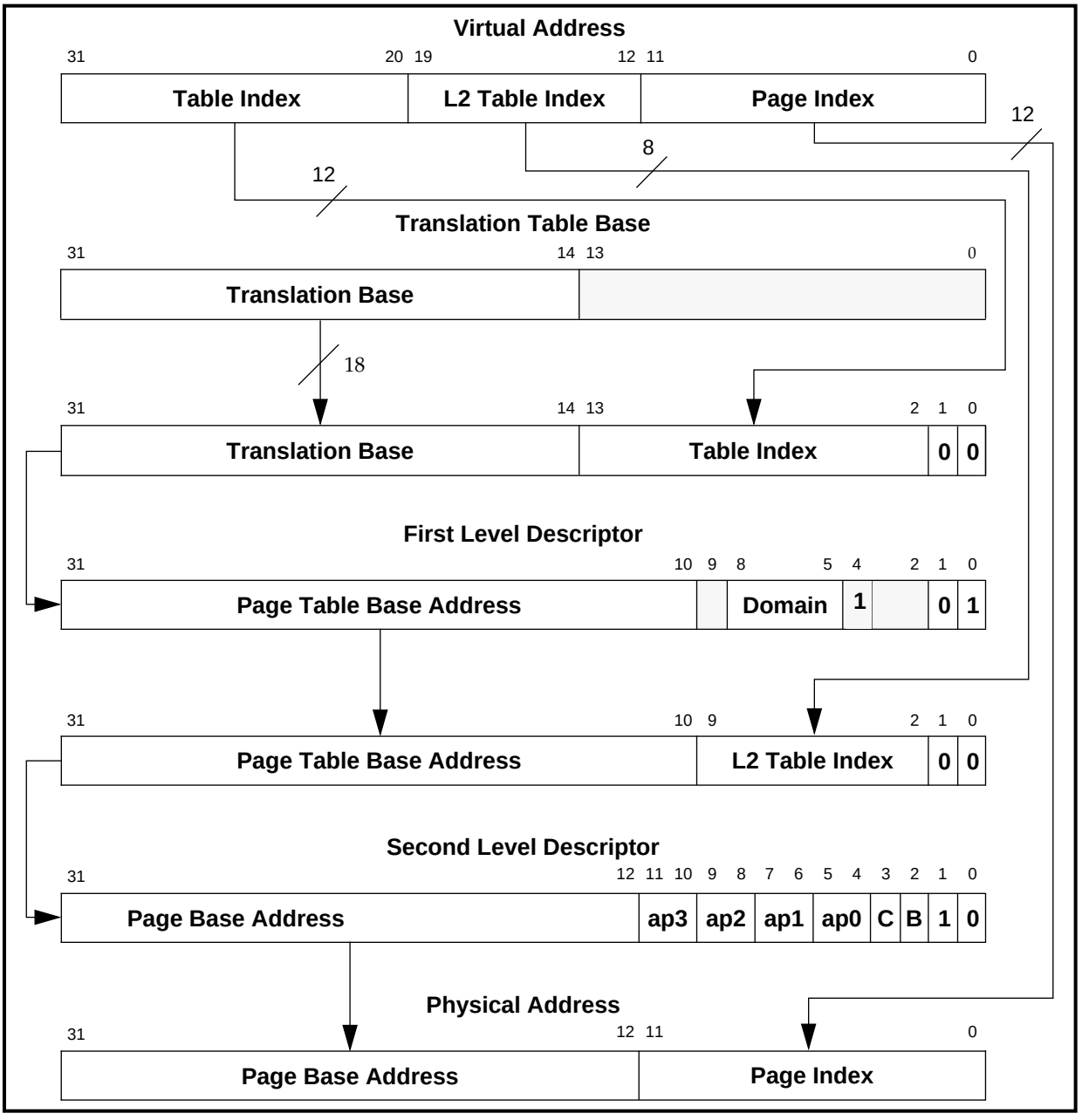

Figure 8-6: Small page translation

8.10 Translating Large Page References

Figure 8-7: Large page Ttanslation illustrates the complete translation sequence for a 64 KB Large Page. Note that since the upper four bits of the Page Index and low-order four bits of the Page Table index overlap, each Page Table Entry for a Large Page must be duplicated 16 times (in consecutive memory locations) in the Page Table.

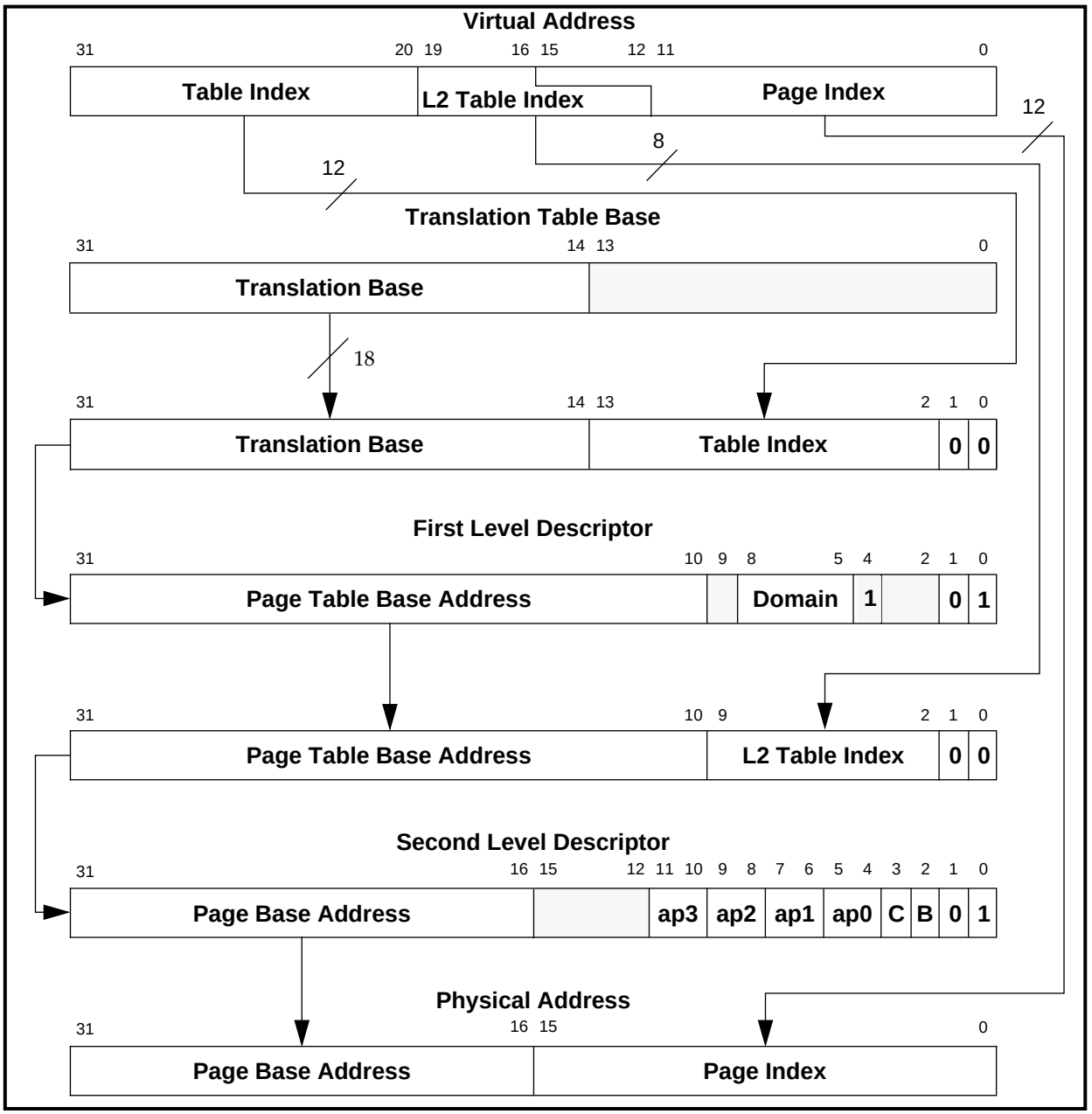

Figure 8-7: Large page Ttanslation

8.11 Cacheable and Bufferable Status of Memory Regions

For first level translation table descriptor for each Section, and the second level translation table descriptor for each Large Page, and each Small Page contain two bits—the C-bit and the B-bit—which specify whether the memory in that Section or Page will be cached or buffered, and whether it will be cached with Write-Through or Write-Back behaviour.†

In addition the cache and write buffer behaviour is controlled by the cache enable bit (C-bit) and write buffer enable bit (W-bit) in the CP15 Control Register.

To differentiate the two C bits, we shall add the subscript “tt” to the translation table bits giving us Ctt and Btt, and the subscript “cr” to the control register bits giving us Ccr and Wcr.

The Cache and Write Buffer Configuration is determined by the values of Ctt, Btt, Ccr, Wcr as shown in ***Table 8-5: Cache and write buffer configuration***.

Note † *Write-Back caches are also known as Copy-Back caches.*
“AND” means bitwise AND function.

Ctt AND Ccr	Btt AND Wcr	Cache, Writebuffer & External Abort Operation
0	0	Non-Cached, Non-Buffered (NCNB) • Reads and Writes are not cached. • Writes are not buffered. • Reads and writes may be externally aborted.*
0	1	Non-Cached Buffered (NCB) • Reads and Writes are not cached. • Writes are buffered. • Reads may be externally aborted. • Writes cannot be externally aborted.
1	0	Cached, Write-Through Mode. (WT) • Reads which hit in the cache read the data from the cache and do not perform an external access. • Reads which miss in the cache cause line fills which may be externally aborted. • All writes go off chip and are buffered. • Writes which hit in the cache update the cache. • Writes cannot be externally aborted.

Table 8-5: Cache and write buffer configuration

Ctt AND Ccr	Btt AND Wcr	Cache, Writebuffer & External Abort Operation
1	1	Cached, Write-Back Mode. (WB) • Reads which hit in the cache read the data from the cache and do not perform an external access. • Reads which miss in the cache cause line fills which may be externally aborted. • Writes which miss in the cache go off-chip and are buffered. • Writes which hit in the cache update the cache and mark the entry as dirty, and do not cause an external access. • Cache write-backs are buffered. • Writes (Cache Write-Misses & Cache Write-Backs) cannot be externally aborted.

Table 8-5: Cache and write buffer configuration (Continued)

Note that the Control Register C bit (Ccr) being zero disables all lookups in the cache, while the Translation table Register C bit (Ctt) being zero only stops new data being loaded into the cache. With Ccr = 1 and Ctt = 0 the cache will still be searched on every access to check whether the cache contains an entry for the data.

8.12 MMU Faults and CPU Aborts

The MMU generates six types of faults:

- Alignment Fault
- Translation Fault
- Domain Fault
- Permission Fault
- Terminal Fault
- Vector Fault

In addition, an external abort may be raised on external data access.

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU will abort the access and signal the fault condition to the CPU. The MMU is also capable of retaining status and address information about the abort. The CPU recognises two types of abort: data aborts and prefetch aborts, and these are treated differently by the MMU. See ***8.13 Fault Address and Fault Status Registers (FAR and FSR)***.

If the MMU detects an access violation, it will do so before the external memory access takes place, and it will therefore inhibit the access. External aborts will not necessarily inhibit the external access, as described in the section on external aborts.

8.13 Fault Address and Fault Status Registers (FAR and FSR)

Aborts resulting from data accesses (data aborts) are acted upon by the CPU immediately, and the MMU places an encoded 4 bit value FS[3:0], along with the 4 bit encoded Domain number, in the Fault Status Register (FSR). In addition, the virtual processor address associated with the data abort is latched into the Fault Address Register (FAR). If an access violation simultaneously generates more than one source of abort, they are encoded in the priority given in ***Table 8-6: Priority Encoding of Fault Status*** on page 8-17.

CPU instructions on the other hand are prefetched, so a prefetch abort simply flags the instruction as it enters the instruction pipeline. Only when (and if) the instruction is executed does it cause an abort; an abort is not acted upon if the instruction is not used (i.e. it is branched around). Because instruction prefetch aborts may or may not be acted upon, the MMU status information is not preserved for the resulting CPU abort; for a prefetch abort, the MMU does not update the FSR or FAR.

The sections that follow describe the various access permissions and controls supported by the MMU and detail how these are interpreted to generate faults.

Source	Priority	Domain[3:0]	FAR
highest priority			
Terminal Exception	0b0010	invalid	VA of start of cache line being written-back
Vector Exception	0b0000	invalid	VA of access causing abort
Alignment	0b00x1	invalid	VA of access causing abort
External Abort on Translation First level Second level	0b1100 0b1110	invalid valid	VA of access causing abort
Translation Section Page	0b0101 0b0111	invalid valid	VA of access causing abort
Domain Section Page	0b1001 0b1011	valid valid	VA of access causing abort
Permission Section Page	0b1101 0b1111	valid valid	VA of access causing abort
External Abort on linefetch Section Page	0b0100 0b0110	valid valid	VA of start of cache line being loaded
External Abort on non-linefetch Section Page	0b1000 0b1010	valid valid	VA of access causing abort
lowest priority			

Table 8-6: Priority Encoding of Fault Status

Notes

1 Alignment faults may write either 0b0001 or 0b0011 into FS[3:0].

2 Invalid values in Domain[3:0] occur because the fault is raised before a valid domain field has been selected.

3 Any abort masked by the priority encoding may be regenerated by fixing the primary abort and restarting the instruction.

4 The FS[3:0] encoding for Vector Exception breaks from the pattern that FS[0]==0 indicates an external abort.

8.14 Domain Access Control

MMU accesses are primarily controlled via domains. There are 16 domains, and each has a 2-bit field to define it. Two basic kinds of users are supported: Clients and Managers. Clients use a domain; Managers control the behaviour of the domain. The domains are defined in the Domain Access Control Register. ***Figure 8-8: Domain Access Control Register format*** on page 8-19 illustrates how the 32 bits of the register are allocated to define the sixteen 2-bit domains.

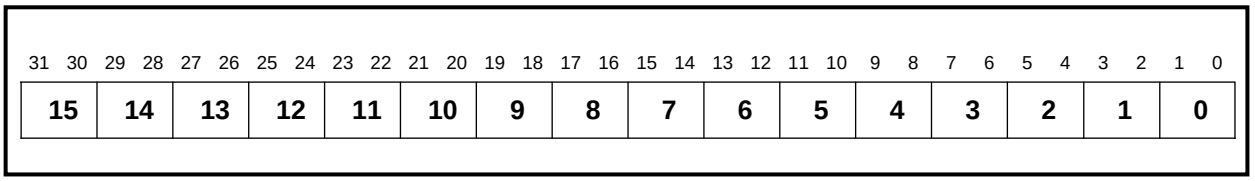

Figure 8-8: Domain Access Control Register format

Table 8-7: Interpreting access bits in Domain Access Control Register defines how the bits within each domain are interpreted to specify the access permissions.

Value	Meaning	Notes
00	No Access	Any access will generate a Domain Fault.
01	Client	Accesses are checked against the access permission bits in the Section or Page descriptor.
10	Reserved	Reserved. Currently behaves like the no access mode.
11	Manager	Accesses are NOT checked against the access Permission bits so a Permission fault cannot be generated.

Table 8-7: Interpreting access bits in Domain Access Control Register

8.15 Fault Checking Sequence

The sequence by which the MMU checks for access faults is slightly different for Sections and Pages. The figure below illustrates the sequence for both types of accesses. The sections and figures that follow describe the conditions that generate each of the faults.

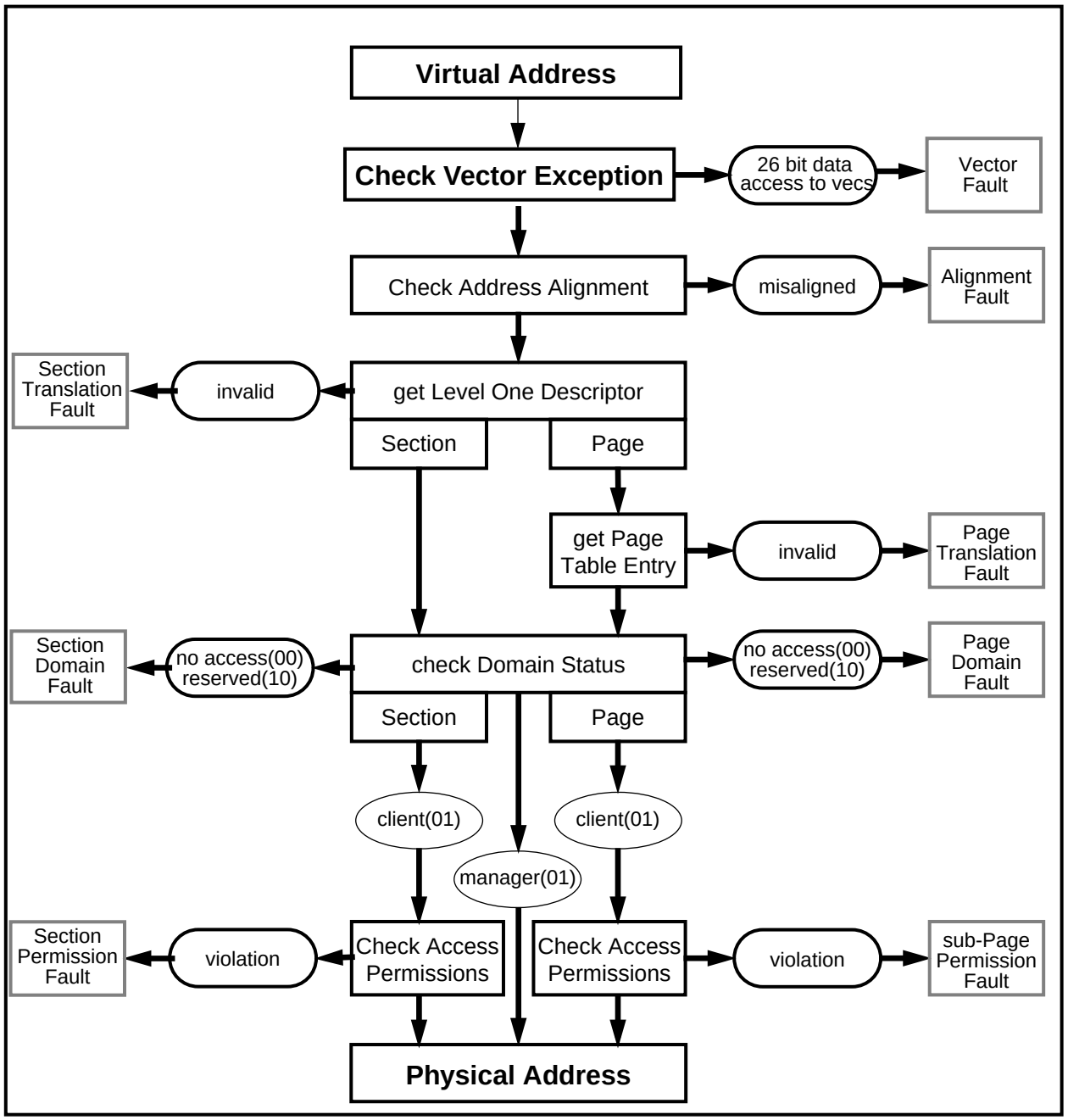

Figure 8-9: Sequence for checking faults

8.15.1 Terminal fault

A terminal fault indicates a system software error in the maintenance of the translation tables in main memory when using the Instruction-Data-Cache in Write-Back mode. It is indicated in theFault Address Register and Fault Status Register to aid debugging system software.

A terminal fault is indicated when a cache-write-back fails to translate the virtual address of the cache line to be written-back into a physical address because the associated translation table walk was aborted by the memory system or returned an invalid Level One or Level Two descriptor [A descriptor is invalid if bits[1:0] have the value “00” or “11”].

System Software must ensure that the cache contains no dirty-data for a page or section before changing the virtual-to-physical mapping of that page or section or disabling the virtual-to-physical mapping of that page or section. A Terminal Fault indicates that system software has failed to do this. When a terminal fault occurs, the data to be written-back from the cache to main memory is irrecoverably lost. A terminal fault is therefore not a reversible fault.

8.15.2 Vector fault

A Vector fault is generated by the MMU if the processor attempts a load or store data access to an address in the range &00000000 and &0000001F inclusive when operating in a 26-bit Mode. Vector faults are never generated for instruction fetches. Vector faults are generated regardless of the setting of the MMU enable bit (M-bit) in the System Control Coprocessor Control Register.

8.15.3 Alignment fault

If Alignment Fault is enabled (bit 1 in Control Register set), the MMU will generate an alignment fault on any data word access the address of which is not word-aligned irrespective of whether the MMU is enabled or not; in other words, if either of virtual address bits [1:0] are not 0. Alignment fault will not be generated on any instruction fetch, nor on any byte access. Note that if the access generates an alignment fault, the access sequence will abort without reference to further permission checks.

8.15.4 Translation fault

There are two types of translation fault: section and page.

1. A Section Translation Fault is generated if the Level One descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0 or both 1.
2. A Page Translation Fault is generated if the Page Table Entry is marked as invalid. This happens if bits[1:0] of the entry are both 0 or both 1.

8.15.5 Domain fault

There are two types of domain fault: section and page. In both cases the Level One descriptor holds the 4-bit Domain field which selects one of the sixteen 2-bit domains in the Domain Access Control Register. The two bits of the specified domain are then checked for access permissions as detailed in ***Table 8-3: Interpreting access permission (AP) Bits*** on page 8-9. In the case of a section, the domain is checked once the Level One descriptor is returned, and in the case of a page, the domain is checked once the Page Table Entry is returned.

If the specified access is either No Access (00) or Reserved (10) then either a Section Domain Fault or Page Domain Fault occurs.

8.15.6 Permission fault

There are two types of permission fault: section and sub-page. Permission fault is checked at the same time as Domain fault. If the 2-bit domain field returns client (01), then the permission access check is invoked as follows:

section:

If the Level One descriptor defines a section-mapped access, then the AP bits of the descriptor define whether or not the access is allowed according to ***Table 8-3: Interpreting access permission (AP) Bits*** on page 8-9. Their interpretation is dependent upon the setting of the S bit (Control Register bit 8). If the access is not allowed, then a Section Permission fault is generated.

sub-page:

If the Level One descriptor defines a page-mapped access, then the Level Two descriptor specifies four access permission fields (ap3..ap0) each corresponding to one quarter of the page. Hence for small pages, ap3 is selected by the top 1KB of the page, and ap0 is selected by the bottom 1KB of the page; for large pages, ap3 is selected by the top 16KB of the page, and ap0 is selected by the bottom 16KB of the page. The selected AP bits are then interpreted in exactly the same way as for a section (see ***Table 8-3: Interpreting access permission (AP) Bits*** on page 8-9), the only difference being that the fault generated is a sub-page permission fault.

8.16 External Aborts

In addition to the MMU-generated aborts, ARM810 has an external abort pin which may be used to flag an error on an external memory access. However, not all accesses can be aborted in this way, so this pin must be used with great care. The following section describes the restrictions.

The following accesses may be aborted and restarted safely. In the case of a read-lock-write sequence in which the read aborts, the write will not happen.

Reads

Unbuffered writes

Level One descriptor fetch

Level Two descriptor fetch

read-lock-write sequence

Cacheable reads (linefetches)

A linefetch may be safely aborted on any word in the transfer. If an abort occurs during the linefetch then the cache line will be invalidated. If the abort happens on a word that has been requested by the ARM8, the instruction will be aborted, otherwise the cache line will be invalidated but program flow will *not* be interrupted. The line is therefore invalidated under all circumstances.

Buffered writes.

Buffered writes cannot be externally aborted. Therefore, the system should be configured such that it does not do buffered writes to areas of memory which are capable of flagging an external abort.

Writes to Cacheable Regions

Writes to cacheable regions and cache write-backs are performed as buffered writes and cannot be externally aborted. The system design should ensure that writes to cacheable regions are not externally aborted.

8.17 Interaction of the MMU, IDC and Write Buffer

The MMU, IDC, WB and Branch prediction may be enabled/disabled independently. However, in order for the write buffer or the cache to be enabled the MMU must also be enabled. Also, Branch prediction must never be enabled when the cache is disabled. There are no hardware interlocks on these restrictions, so invalid combinations will cause undefined results.

MMU	IDC	WB
off	off	off
on	off	off
on	on	off
on	off	on
on	on	on

Table 8-8: Valid MMU, IDC and Write Buffer combinations

The following procedures must be observed.

To enable the MMU:

1 Program the Translation Table Base and Domain Access Control Registers
2 Program Level 1 and Level 2 page tables as required
3 Enable the MMU by setting bit 0 in the Control Register.

Note *Care must be taken if the translated address differs from the untranslated address as severalinstructions following the enabling of the MMU mayhave been fetched using "flat translation" and enabling the MMU may be considered as a branch with delayed execution. A similar situation occurs when the MMU is disabled. Consider the following code sequence:*

```
MOV             R1, #0x1
MCR             15,0,R1,0,0      ; Enable MMU
Fetch Flat
Fetch Flat
Fetch Translated
```

To disable the MMU:

1. Disable Branch prediction, if it is enabled, by using the code sequence given in ***6.3.3 Turning off Branch Prediction***.
2. Disable the WB by clearing bit 3 in the Control Register.
3. Disable the IDC by clearing bit 2 in the Control Register.
4. Disable the MMU by clearing bit 0 in the Control Register.
   Note that if the MMU is enabled, then disabled and subsequently re-enabled the contents of the TLB will have been preserved. If these are now invalid, the TLB should be flushed before re-enabling the MMU.

Disabling of all three functions described in steps 2, 3 and 4 may be done simultaneously.

8.18 Effect of Reset

See ***3.7 Reset*** on page 3-12.

9 Write Buffer

This chapter describes the *Write Buffer* (*WB*).

Write Buffer

The ARM810 write buffer is provided to improve system performance. It can buffer up to 8 words of data, and 4 independent addresses. It may be enabled or disabled via the W bit (bit 3) in the ARM810 Control Register and the buffer is disabled and flushed on reset. The operation of the write buffer is further controlled by the C and B bits which are stored in the Memory Management Page Tables. For this reason, in order to use the write buffer, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register. For a write to use the write buffer, both the W bit in the Control Register and either the C or B bit in the corresponding page table must be set.

It is not possible to abort buffered writes externally; the abort pin will be ignored. Areas of memory which may generate aborts should be marked as unbufferable in the MMU page tables.

9.1 Cacheable and Bufferable bits

These bits controls whether a write operation may or may not use the write buffer. Typically main memory will be cacheable and bufferable and I/O space unbufferable. The C and B bits can be configured for both pages and sections. This is decribed in section ***8.11 Cacheable and Bufferable Status of Memory Regions*** on page 8-147.

9.2 Write Buffer Operation

9.2.1 Bufferable write

If the write buffer is enabled and the processor performs a write to a bufferable area, the data is placed in the write buffer at **FCLK** (**MCLK** if running with fastbus extension) speeds and the CPU continues execution. The write buffer then performs the external write in parallel. If however the write buffer is full (either because there are already 8 words of data in the buffer, or because there is no slot for the new address) then the processor is stalled until there is sufficient space in the buffer.

9.2.2 Unbufferable writes

If the write buffer is disabled or the CPU performs a write to an unbufferable area, the processor is stalled until the write buffer empties and the unbufferable write completes externally, which may require synchronisation and several external clock cycles.

9.2.3 Read-lock-write

The write phase of a read-lock-write sequence is treated as an Unbuffered write, even if it is marked as buffered.

Note: *A single write requires one address slot and one data slot in the write buffer; a sequential write of n words requires one address slot and n data slots. The total of 8 data slots in the buffer may be used as required. So for instance there could be 3 non-sequential writes and one sequential write of 5 words in the buffer, and the processor could continue as normal: a 5th write or a 6th word in the 4th write would stall the processor until the first write had completed.*

9.2.4 To enable the Write Buffer

To enable the write buffer, ensure the MMU is enabled by setting bit 0 in the Control Register, then enable the write buffer by setting bit 3 in the Control Register. The MMU and write buffer may be enabled simultaneously with a single write to the Control Register.

9.2.5 To disable the Write Buffer

To disable the write buffer, clear bit 3 in the Control Register.

Note *Any writes already in the write buffer will complete normally.*

10 Coprocessors

This chapter describes use of coprocessors with the ARM810.

Coprocessors

10.1 Overview

The ARM810 has no external coprocessor interface, so it is not possible to add external coprocessors to ARM810.

ARM810 has an internal coprocessor, called the System Control Coprocessor designated as coprocessor number 15. The System Control Coprocessor is used to control the configuration of the device, including the endianness setting, enabling of the Cache, MMU, Writebuffer, Branch Prediction, and the control of the Cache and MMU.

The System Control coprocessor is documented in detail in ***Chapter 5, Configuration*** and in the chapters on those parts of the ARM810 it controls: ***Chapter 7, Instruction and Data Cache (IDC)***, ***Chapter 8, Memory Management Unit***,***Chapter 9, Write Buffer***, ***Chapter 6, The Prefetch Unit***.

11 ARM810 Clocking

This chapter describes the bus interface clocking:

ARM810 Clocking

The ARM810 uses two clock signals:

- bus clock
- fast clock

These clocks are derived from external inputs to the processor with configurations defined by external pins and the on-chip programmable registers.

The fast clock can be selected from three sources:

- bus clock
- on-chip PLL
- external reference clock

When the fast clock is sourced from the bus clock, operation is equivalent to ARM710a's Fastbus mode. When the fast clock is sourced from the external reference clock, the operation is equivalent to ARM710a's Standard bus mode.

The following sections explain how these clocks are made and describe their expected usage. In particular, note the addition of a clock multiplier (PLL) in this design.

11.1 The Bus Clock

The external bus clock is used to cycle the external bus interface. This clock is sourced directly from external input pins of the device. See ***Figure 11-1: Generating the external bus interface clock***.

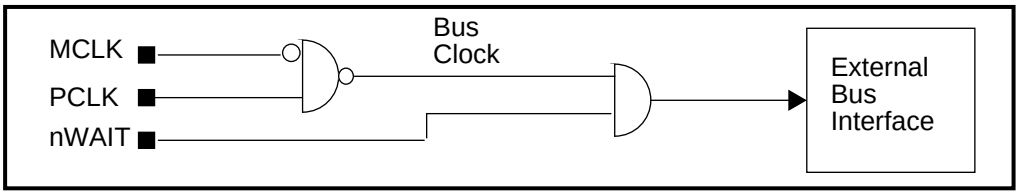

Figure 11-1: Generating the external bus interface clock

The bus clock is gated with **nWait** to provide the external bus clock itself. This allows external bus cycles to be extended if system timing requires it (see ***Figure 12-9: Use of the nWAIT pin to stop ARM810 for 1 MCLK cycle*** on page 12-16 for timing details).

11.1.1 External input clock: MCLK or PCLK

To provide for synchronous memory systems (eg. SDRAM, SSRAM) that use a clock which is essentially an inverted bus clock (returning data on the rising clock edge), you can choose to use the **PCLK** rather than the **MCLK** external input to avoid having to invert the clock externally. If you use **PCLK**, **MCLK** must be tied HIGH. If you use **MCLK**, **PCLK** must be tied LOW. New system designs should use **PCLK** for future compatibility. **MCLK** is provided for backwards compatibility. In future references in this document, the term *bus clock* refers to **MCLK** or **PCLK** depending on which is being used.

11.2 The Processor Clock

The processor clock is used to cycle the internals of the processor, see ***Figure 11-2: Generating the Processor Clock***. The processor clock can be sourced by one of two input clock signals to the synchroniser:

- bus clock
- fast clock

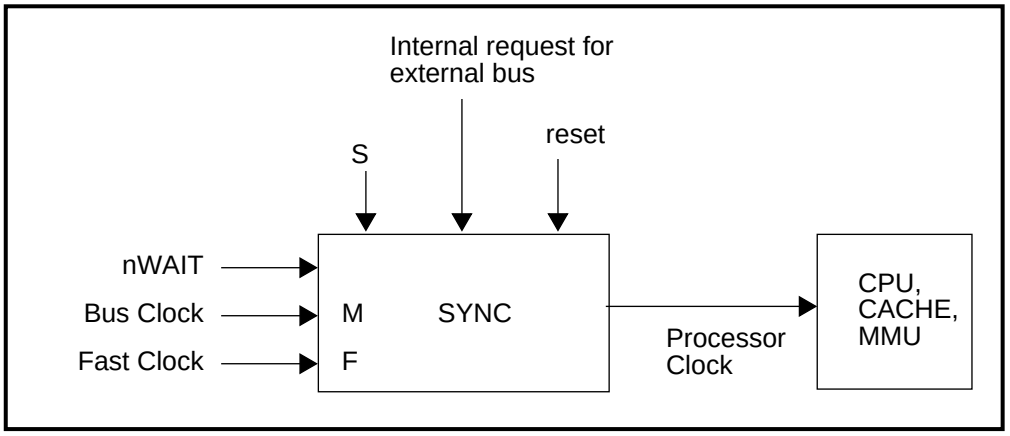

Figure 11-2: Generating the Processor Clock

When the processor is not performing external memory accesses, the fast clock (F) input to the synchroniser is the source for the processor clock (See ***11.3 Generation of the Fast Clock*** on page 11-6 for details of generating the fast clock). When external memory accesses are being made by the processor, the bus clock (M) input to the synchroniser is the source for the processor clock (See ***11.1 The Bus Clock*** on page 11-3 for details of generating the bus clock). Which of the sources to use is determined by the internal request for external bus signal during normal operation (see ***11.4 Forced Processor Clock from the Bus Clock*** on page 11-9 for details during RESET). When changing between F and M inputs, the synchroniser may perform re-synchronisation.

Note When a buffered write is made, the processor clock continues to run from the fast clock source at highest performance.

11.2.1 Synchronous/asynchronous operation

The state of the S bit (from Coprocessor 15, Register 15, bit 1) determines whether any synchronisation occurs between the bus clock (M) and fast clock (F) inputs to the synchroniser when the processor clock is changed from one to the other before and after external memory access cycles.

Synchronous operation

If the S bit is HIGH, there must be a tightly defined relationship between the bus clock and the fast clock (if this relationship is not obeyed, then the S bit should be set LOW). With the S bit HIGH, the Synchroniser will not perform any synchronisation, and the bus clock may only make transitions on the falling edge of the fast clock. Please refer to Section 15.2 for the timing requirements.

Asynchronous Operation

If the S bit is LOW, there is no defined relationship between the bus clock and the fast clock - they are asynchronous. The synchroniser introduces a synchronisation penalty whenever the internal core clock switches between the two input clocks (bus clock (M) and fast clock (F)). This penalty is symmetric, and varies between nothing and a whole period of the clock to which the core is synchronising. For example, when changing from the fast clock to the bus clock, the average synchronisation penalty is half a bus clock period, and when changing from the bus clock to the fast clock, it is half a fast clock period.

11.3 Generation of the Fast Clock

The fast clock input to the synchroniser can be selected from three sources. These are all configured internally using Coprocessor 15, Register 15, bits 2 and 3: F0 and F1. See ***11.5 Low Power Idle and Sleep*** on page 11-10 further details. During RESET, the bus clock is selected as the initial source for the fast clock.

11.3.1 Fast clock from the bus clock (Fastbus mode)

This configuration (F0=0, F1=0) makes the bus clock the source for the fast clock. This guarantees a defined relationship between the fast clock and the bus clock, and so synchronous operation (S=1) can be used for improved performance. This configuration is selected at RESET.

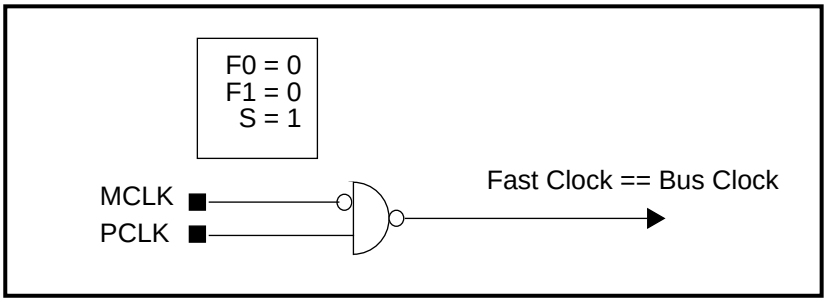

Figure 11-3: Fast clock the same as the bus clock

11.3.2 Fast clock from the output of the PLL

This configuration (F0=1, F1=1) makes the output of the PLL clock multiplier the source for the fast clock (see ***Figure 11-4: Fast clock from the output of the PLL***) When operating in this configuration, the S bit must be set LOW for asynchronous operation (S=0).

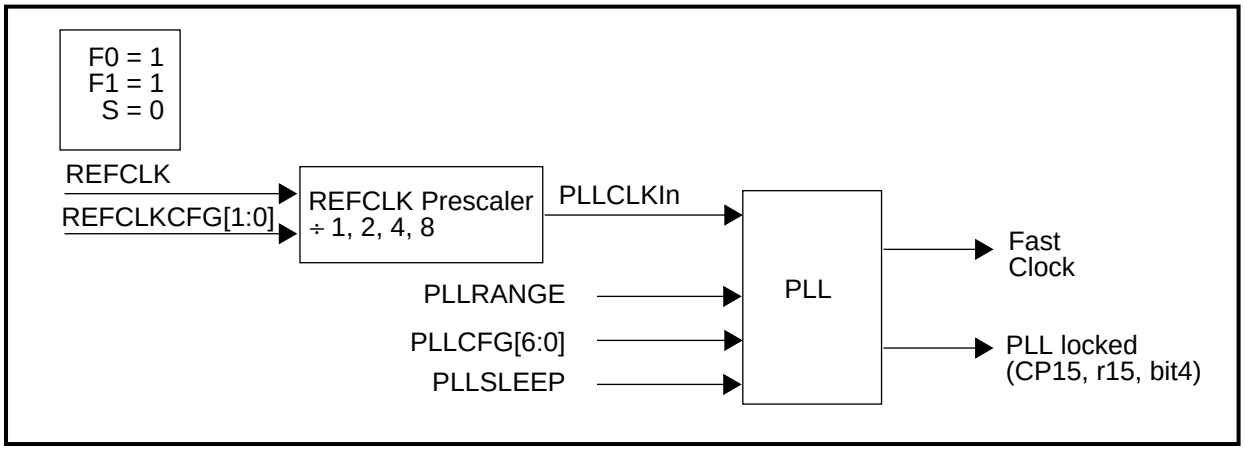

Figure 11-4: Fast clock from the output of the PLL

The PLL and input clock prescaler can be used to produce a fast clock frequency in the range 45MHz to 100MHz (or 22.5MHz to 50MHz if **PLLRANGE** is HIGH) from a **REFCLK** frequency of 1MHz to 80MHz.

The **REFCLK** prescaler divides the **REFCLK** frequency by 1, 2, 4 or 8 to produce **PLLCLKIn** under control of **REFCLKCFG** as shown in ***Table 11-1: Prescaler divide ratios***.

REFCLKCFG		Divide ratio
0	0	1
0	1	2
1	0	4
1	1	8

Table 11-1: Prescaler divide ratios

PLLCLKIn must be in the range 1MHz to 10MHz.

The fast clock output frequency is defined according to the following equation:

$$f_{FastClock} = f_{PLLCLKIn} * M/2$$

where:

$f_{FastClock}$ is the frequency of the fast clock output

$f_{PLLCLKIn}$ is the frequency of **PLLCLKIn**, which is the frequency of **REFCLK** divided by 1, 2, 4 or 8.

M is the value of the **PLLCFG** bus if interpreted as normal unsigned binary reporesentation. M is defined for the range M = 5, 6, 7 ..., 127. Values of M less than 5 are invalid.

The output frequency range of the PLL must reside between certain limits. These limits are determined by the **PLLRANGE**pin shown in ***Table 11-2: Output frequency range***.

PLLRANGE	Min Fast Clock (MHz)	Max Fast Clock (MHz)
LOW	45	100
HIGH	22.5	50

Table 11-2: Output frequency range

11.3.3 Fast clock direct (bypassing the PLL)

This configuration (F0=1, F1=0) provides a means of directly driving the fast clock from an external pin. This configuration may operate synchronously or asynchronously depending on how the reference clock (**REFCLK**) is generated. ***Figure 11-5: Fast clock direct*** shows this configuration.

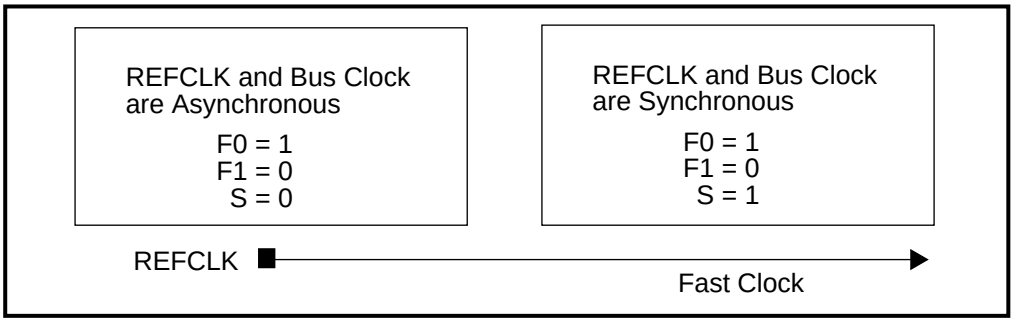

Figure 11-5: Fast clock direct

11.4 Forced Processor Clock from the Bus Clock

Coprocessor 15, Register 15, bit 0 (the **D** bit) is used to override the internal request for external bus signal to the synchroniser (see ***Figure 11-2: Generating the Processor Clock*** on page 11-4) and force the processor clock to be sourced from the bus clock. At RESET, the **D** bit is set LOW, and so the processor clock is sourced from the bus clock until this bit is changed. Once the fast clock source has been configured, and is sufficiently stable, the **D** bit should be set HIGH so the processor runs from the fast clock when not accessing the external bus.

11.5 Low Power Idle and Sleep

The **D** bit (see Section 11.4) can be employed to provide a clean transition to allow low-power idle or sleep mode. As the ARM810 is a fully static processor, stopping its clock when it has no work to do provides an ideal way to minimise power consumption and provide a fast start-up when it needs to operate again - all state is just frozen and does not need to be restored.

The easiest means of stopping the processor (and associated system) is to stop the bus clock. To allow the system to be stopped with the processor state at a precisely defined point in program execution, the processor clock must be sourced from the bus clock and the bus clock stopped. This can be achieved by setting the **D** bit LOW (writing 0 to Coprocessor 15, Register 15, bit 0) and then stopping the bus clock externally.

If the fast clock is being generated by the PLL clock multiplier and the PLL is left running while the bus clock is stopped, after restarting the bus clock, the **D** bit can be set HIGH and the processor clock sourced from an already locked fast clock PLL source. This could be implemented in the system for a fast wake-up-from-sleep interrupt response (though more power is consumed if the PLL is running continuously whilst the rest of the system is stopped).

The PLL itself can be placed in Sleep mode (using the PLLSLEEP external input), where it stops running and therefore consuming power. On wake-up, the PLL will take time to lock, and the system must take this into account - more details to be advised in future.