NXP Semiconductors
Application Notes

Document Number: AN12177
Rev. 0, 04/2018

Power Architecture e200z4 and e200z7 Core Memory Protection Unit (CMPU)

by: NXP Semiconductors

Contents

1. Introduction

This application note describes the function, configuration and use of the Core Memory Protection Unit (MPU) contained in the e200z4 and e200z7 device cores. Each of these cores (with the exception of safety checker lock-step cores) incorporates an MPU which functions to protect user configured regions of memory from unwanted instruction and/or data accesses, and also regulates the caching and guarding of accesses. Since the MPU's present in both core types are largely the same this application note will consider them equivalent, however where the implementation of the MPU between the e200z4 and e200z7 cores differs in any area discussed, it will be highlighted.

The Core MPU is separate to the System MPU (SMPU) which also exists on the device, and provides extra protection coverage to ensure that all activity within the device is protected. Full SMPU configuration is not included in the scope of this application note, but details can be found in the device reference manual.

The implementation of the MPU for each specific core variant used in different products can vary slightly, but the overall principal of operation can be considered the same. For this reason and to simplify the explanations this application note will focus on the S32R274 MCU, which contains three Power Architecture® cores arranged as two independent e200z7260 for general

computation and an e200z420 core with an e200z419 checker core running in delayed lockstep configuration for safety-critical and housekeeping tasks. Therefore, the reader should ensure that they confirm the exact core type used in their product, as the implementation may have a different amount of region descriptors and address ranges than those detailed in this application note. The main described operation and use of the core MPU's however can still be considered valid.

Any MPU programming syntax used in this application note has been simplified to show the value to be programmed in one line, but for correct compilation would require the 32-bit value to be written first to a general purpose register (GPR). E.g.

```
# Disable and Invalidate core MPU
mtspr 1014, 0x00000002     # set MPUFI in MPU0CSR0 to trigger MPU invalidation
```

Should be implemented as:

```
# Disable and Invalidate core MPU
e_li r4, 0x0000
e_or2i r4, 0x0002
mtspr 1014, r4             # set MPUFI in MPU0CSR0 to trigger MPU invalidation
```

2. MPU overview

The MPU's present on the S32R274 device provide the capability of protecting regions of memory, with the following feature set:

- 24-entry region[1] descriptor table with support for -
    - Six arbitrary-sized instruction memory regions
    - 12 arbitrary-sized data memory regions
    - Six additional arbitrary-sized instruction or data memory regions
- Ability to set access permissions and memory attributes on a per-region basis
- Process ID aware, with per-bit masking of Task ID values
- Capability for masking upper address bits in the range comparison
- Capability of bypassing permissions checking for selected access types
- Per-entry write-once logic for entry protection
- Hardware flash invalidation support and per-entry invalidation protection controls
- Ability to optionally utilize region descriptors for generating debug events and watchpoints
- Software managed by mpure and mpuwe instructions

[1] Check specific core implementation for exact number of region descriptors present for other devices

3. Access protection operation

The MPU provides access protection to memory regions using a set of region descriptors contained in a table structure. Each region descriptor defines an address range and a set of access protections and memory attributes. Individual region descriptors monitor either instruction accesses or data accesses. Independent access protection may be configured for supervisor-mode and user-mode reads, writes, and instruction accesses. In addition, any subset of these access types can bypass the protection system and instead rely on the external SMPU for enforcing access protections. This allows for an increased number of regions to be supported in an SoC, without requiring excessive processor resources. Access addresses which fall within the address range(s) of one or more enabled region descriptors (a match) are allowed to proceed if the access permissions for any matching descriptor are set to allow the access. If no matching region descriptor is found, the access will not be allowed, and an Instruction Storage Interrupt (ISI) or Data Storage Interrupt (DSI) will be generated. If more than one matching descriptor is found, the access permissions enforced are the least restrictive access permissions defined by the matching entries.

Allowed accesses are based only on an address compare of the first byte of data being accessed; no checking is performed that the data associated with a single access is entirely contained in an allowed region.

4. Region descriptor table and region descriptors

The e200z420 and both e200z7260 cores on the S32R274 device support one MPU implementation dedicated to that core. Each MPU contains a 24-entry region descriptor table with support for:

- Six arbitrary-sized instruction memory regions
- 12 arbitrary-sized data memory regions
- Six additional arbitrary-sized instruction or data memory region

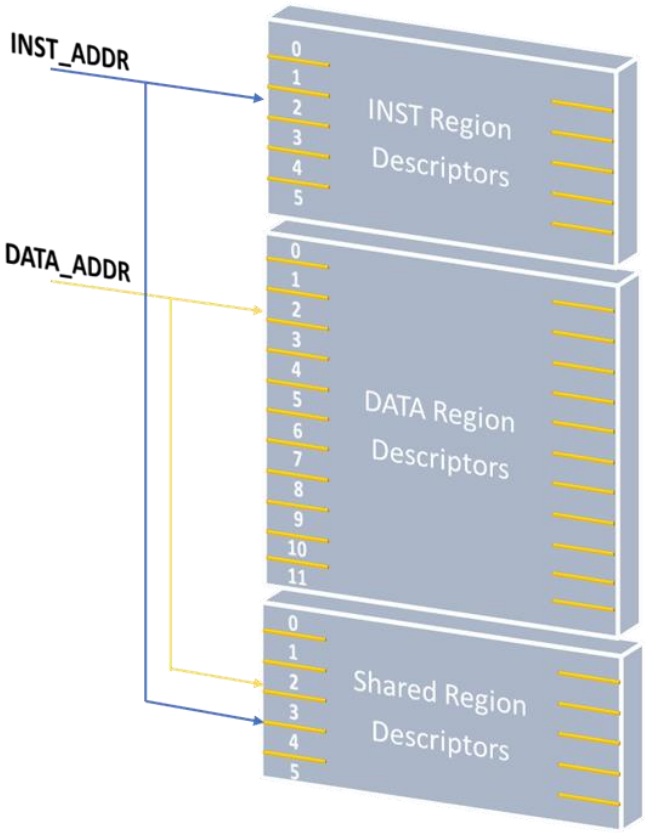

Figure 1. **Region descriptor table shows a high-level diagram of the region descriptor table structure**

Each Region Descriptor contains an Upper and Lower Bound value that defines the size (range) of the region, as well as additional information such as the Region ID, which is compared with the current process ID in the PID register to determine if a match occurs, a Valid bit indicating the region descriptor is valid and should participate in the lookup process, masks for masking upper address bits as well as portions of the region ID, protection information, access attributes, and various control fields. The table below shows the information contained in a region descriptor. The various fields are described further in subsequent sections.

Table 1. Region descriptor entry bit definitions

Field	Comments
VALID	Valid bit for entry
UPPER_BOUND	Upper address bound (compared against effective address)
LOWER_BOUND	Lower address bound (compared against effective address)
TID[0:7]	Region ID (compared against PID value or ‘0’)
TIDMSK[0:7]	Region ID Mask
UAMSK	Upper Address Mask control
INST	Instruction or Data Access entry (1=INST)
DEBUG	Entry set for debug event generation
SX, SW, SR	Supervisor execute, write, and read permission bits
UX, UW, UR	User execute, write, and read permission bits
I	Cache-Inhibited region attribute
IOVR	Cache-Inhibited region attribute override

Field	Comments
G	Guarded region attribute (not present in dedicated INST entries)
GOVR	Guarded region attribute override (not present in dedicated INST entries)
IPROT	Invalidation protect
RO	Read-only control for entry updates

5. Access monitoring, matching and masking

When the MPU is enabled, access monitoring and matching is performed by comparing the effective address (EA) for the access, and the process ID (PID) value (if) used for the access with values stored in the region descriptors.

The effective address for an instruction fetch access is the address calculated using the current value of the program counter and optionally an absolute or relative displacement value. For a data load or store access the effective address is calculated using GPR contents and optionally a displacement value.

5.1. Instruction accesses

Instruction accesses are generated by sequential instruction fetches or due to a change in program flow (branches and interrupts). Instruction accesses are monitored by the dedicated instruction region descriptors (0-5), as well as by any of the shared region descriptors where their INST bit is set to '1'.

For instruction accesses, only the effective address bits 0:28 are compared to the values in the region descriptor when the region descriptor is configured for access protections; this corresponds to a doubleword granularity. Range descriptors being used for debug breakpoint/watchpoint functions compare address bits 0:29 for a match, and then mark each halfword within the doubleword at that address with an in-range indicator, allowing halfword granularity for debug purposes.

5.2. Data accesses

Data accesses are generated by load and store instructions, and certain cache management instructions. Data accesses are monitored by the dedicated region descriptors (0-11) in the data portion of the region descriptor table, as well as by those region descriptors in the shared portion of the region descriptor table which have their INST bit set to '0'. These region descriptors do not monitor instruction access addresses for matching.

When the MPU is enabled, for data accesses, all 32 bits of the access effective address are checked for a match for both normal protection usage as well as debug breakpoint/watchpoint usage.

5.3. Access bypass and masks

5.3.1. Access bypass

Access bypass of the MPU can be enabled for specific access types in both user and supervisor modes via control information located in the MPU0 Control and Status Register 0 (MPU0CSR0). When a specific access type is configured to bypass the MPU then no access protection checking is performed

for that access type, and in addition no access attributes are supplied, unless a region descriptor match occurs.

Accesses to the Local Data memory (DMEM) may also bypass MPU protections when configured in the DMEMCTL1 register. These controls override any MPU-based settings unless configured to use the MPU protections. See the 'Local Memories' chapter in the device Reference Manual for more detail of DMEM MPU bypass functionality.

5.3.2. Address masking (UAMSK)

The MPU provides the capability of per-entry masking up to five MSB's of the upper effective address bits to zero prior to performing the address range comparison. When using upper-address bit masking, the UPPER_BOUND and LOWER_BOUND are not masked, thus should typically be programmed with '0' in the masked bit positions.

5.3.3. Process ID masking (TIDMSK)

The MPU provides the capability of per-entry masking of a subset of the TID/PID bits via the entry TIDMSK field when performing the TID portion of the address range comparison for the entry. This allows for efficient use of a minimum number of region descriptors when a portion of the address space is shared between multiple processes, but not shared across all processes (where a TID value of 0b0 can be used).

5.3.4. Process ID masking in supervisor mode (TIDCTL)

The MPU provides the capability of global masking of the TID bits in all region descriptors to 0b0 when performing the PID/TID portion of the address range comparison for the entry while in supervisor mode. This masking is controlled by the TIDCTL bit in the MPU0 control and status register (MPU0CSR0).

This allows supervisor code to utilize region descriptors loaded for user tasks regardless of the programmed TID value in those descriptors. This capability can be used in certain situations to minimize the number of supervisor region descriptors required to be active, and thus improve efficiency. The actual contents of the descriptors is not changed, the TID values are only masked to 0b0 at the comparison logic.

5.4. Permissions

It is possible to restrict access to address ranges by selectively granting permissions for user and supervisor mode read, write and execute accesses on a per-region basis. These selective permissions are controlled via the UX, SX, UW, SW, UR and SR access control bits:

- SR—Supervisor read permission. Allows loads and load-type cache management instructions to access the region while in supervisor mode.
- SW—Supervisor write permission. Allows stores and store-type cache management instructions to access the region while in supervisor mode.

- SX—Supervisor execute permission. Allows instruction fetches to access the page and instructions to be executed from the region while in supervisor mode.
- UR—User read permission. Allows loads and load-type cache management instructions to access the region while in user mode.
- UW—User write permission. Allows stores and store-type cache management instructions to access the region while in user mode.
- UX—User execute permission. Allows instruction fetches to access the page and instructions to be executed from the region while in user mode.

Note that the permissions checks are generally based on the address of the first byte of the operand or instruction which is accessed. Therefore an access which crosses a region boundary may possibly be allowed regardless of whether the second portion of the access falls within a more restrictive region, or falls outside of a defined region. If an access is misaligned across a doubleword boundary, the permissions checks will be repeated for the second portion of the access.

6. Protection flow

A high-level protection flow is shown in the figure below.

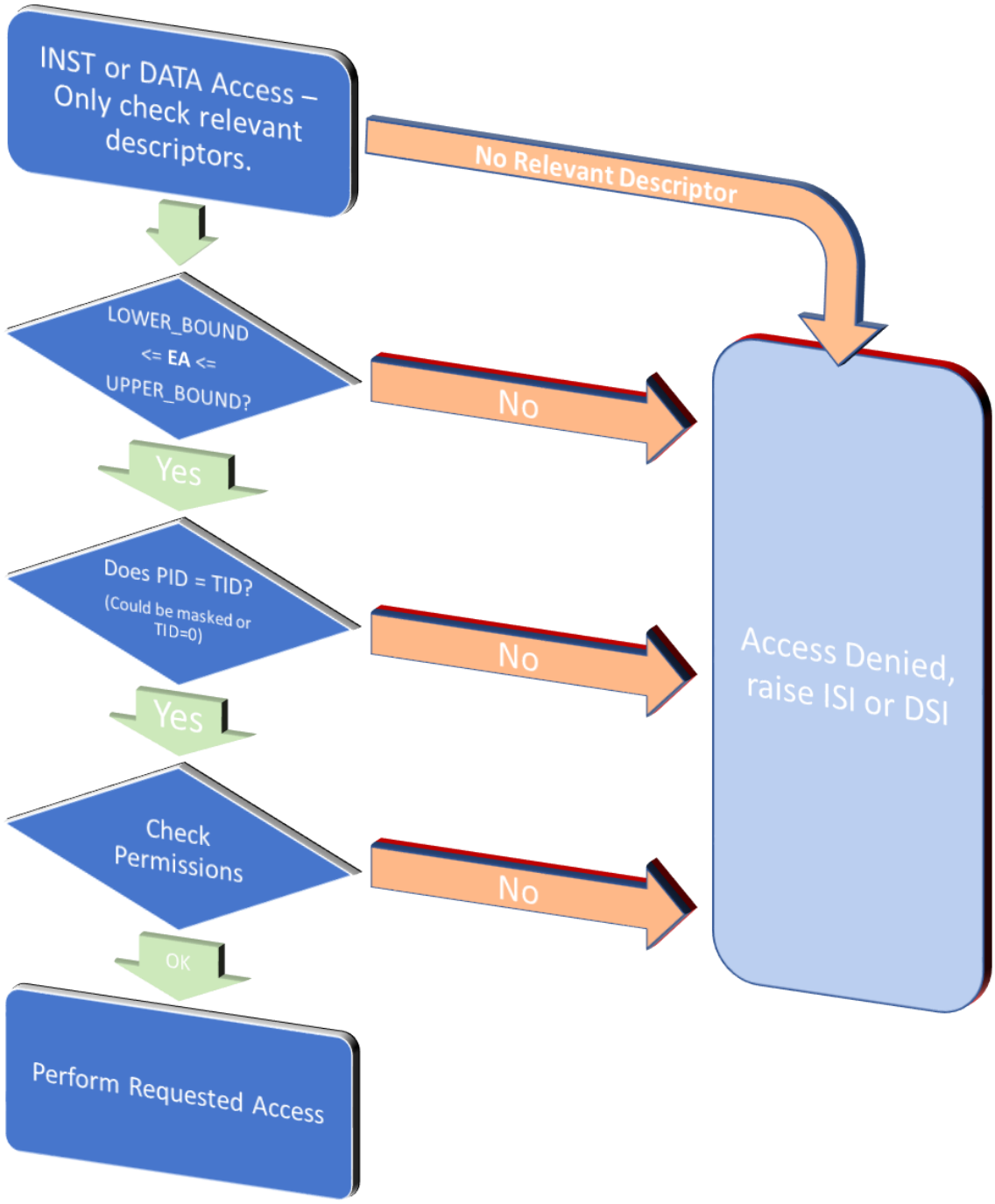

Figure 2. **High-level protection flow**

When an access occurs, the effective address is compared to the appropriate number of bits (depending on UAMSK and type of access) of the UPPER_BOUND and LOWER_BOUND address fields of the region descriptor of each MPU entry appropriate for the access type (instruction/data).

For the UPPER_BOUND comparison, the EA must be <= UPPER_BOUND for the check to pass. For the LOWER_BOUND comparison, the EA must be >=LOWER_BOUND. The comparisons are performed as unsigned compares.

If the effective address value falls within the EA range of the MPU entry, that MPU entry is a candidate for a possible match. In addition to a match in the EA range, a matching MPU entry must match the (possibly masked) stored entry TID value with the current Process ID of the access (current PID register values), or have a (possibly masked) TID value of '0', indicating the entry is globally shared among all processes.

Matching entries are then evaluated for permissions to see if any matching descriptor's permissions allow the access.

If no matching region descriptor is found, or if no region descriptor which matches has permissions which allow the access type, then the access is denied, and an ISI or DSI exception (as appropriate) will be generated.

6.1. Memory attributes (G, I)

Each descriptor contains memory attribute information for the corresponding memory address range covered by the descriptor. The G and I bits are provided to indicate whether a matching data access should be treated as guarded and/or cache-inhibited. It is not present in dedicated instruction entries, and is ignored in shared entries with INST=1. Instruction accesses will use the I bit to determine if a matching instruction fetch access should be treated as cache-inhibited, and to determine the AHB bus attributes for the instruction fetch on a cache miss.

Note that cacheability is checked only for the initial fetch of a cache linefill, thus if a cache line spans more than one region, the initial access address will determine cacheability of the entire line.

For data accesses, the initial address will also determine the guarded attribute for the access, regardless of whether it spans more than one region, unless the access spans a doubleword boundary. In this case both accesses of this misaligned access will perform MPU lookups and will utilize the corresponding attribute, unless the first portion of the access corresponds to a cacheable non-guarded region, in which case the second portion of the access may possibly be treated as non-guarded.

If more than one descriptor attempts to supply memory attributes, the most restrictive attribute settings are used for the Guarded attribute (ORing of G bits), and the least restrictive access settings are used for cacheability (ANDing of I bits), unless override control is enabled in one of the matching descriptors.

6.1.1. Memory attribute overrides (GOVR, IOVR)

Each descriptor also contains memory attribute override controls, which can be used to override the setting(s) of other matching descriptor(s). The GOVR (dedicated data and INST=0 shared entries only) and IOVR bits are provided to indicate whether a matching access uses the G and I settings of a descriptor regardless of the settings of any other matching descriptor or of the state of the guarded address or cache-inhibited address ports.

Only a single matching descriptor is allowed to have GOVR or IOVR set, although it is allowable for two different matching descriptors to each have one of these set which is not set in another matching descriptor. If multiple GOVR or IOVR bits are set, the result is boundedly-undefined. No detection mechanism is provided for this case.

6.2. Region descriptor invalidation and write-once protection

In order to provide an efficient mechanism for reloading the region descriptor table entries on a user task switch, and avoid the need for examining each entry to see if it belongs to the old task, the MPU supports hardware flash-invalidation of the table entries. This is controlled via the MPUFI bit in the MPU0CSR0 register. Supervisor entries can be protected as needed by setting the entry's IPROT (Invalidation protect) bit. This will prevent the entry from being cleared when a hardware flash-invalidation is performed.

In addition, each entry supports a write-once feature via the entry's RO control bit. Setting of the RO control bit prevents a subsequent write access from being performed while the entry remains valid. Only a flash-invalidation or a processor reset will clear the RO bit. If the entry needs to remain unmodified on a flash-invalidation, the entry's IPROT bit can be set concurrently with the RO bit. With IPROT and RO set, a region descriptor may not be changed until the next processor reset occurs which clears these bits.

7. MPU configuration

In order to configure the region descriptors as desired the MPU can be accessed and configured by system software by reading/writing the contents of a number of configuration SPR's (special purpose registers). The MPU is not memory mapped in the device peripheral map since it is part of the device cores.

The MPU entries are indirectly accessed through several MPU Assist (MAS) registers, which contain each of the configurations previously discussed in this document. There are four MAS registers (MAS0-MAS3), software can write and read the MAS registers with 'mtspr' and 'mfspr' instructions. Data is read from the MPU into the MAS registers with an 'mpure' (MPU read entry) instruction. Data is written to the MPU from the MAS registers with an 'mpuwe' (MPU write entry) instruction.

The device reference manual Core MPU chapters contain full bit descriptions of each register, and should be read in conjunction with this section which will give general advice and examples of the configuration steps.

7.1. Configuration steps

This section will describe the steps required to configure and enable the MPU. It should be known to the user which regions of memory require protection and therefore what configuration is required. When the MPU is enabled it operates on the principal that no match in the region descriptor table means the access is not allowed, therefore it is critical that all potential memory locations that could be accessed during the application are covered to ensure no unwanted access violations occur.

For illustration purposes this example will configure the region descriptors to cover all of the available memory map for the S32R274 device, this will ensure that any access outside of these areas results in an access violation and an ISI or DSI will occur depending on the type of access.

To avoid an unhandled exception being generated it is important to ensure that interrupts are enabled and that a relevant interrupt service routine (ISR) is configured to handle the DSI or ISI interrupt before the MPU is configured and enabled.

Data storage interrupts correspond to IVOR2 (Interrupt Vector 2), and Instruction Storage Interrupt corresponds to IVOR3 as defined by the Power Architecture standard.

7.1.1. Disable and invalidate MPU configuration

First the user should ensure the MPU is disabled and invalidate any previous MPU configuration by setting the MPUFI bit in the MPU0CSR0. This triggers the hardware flash-invalidation mechanism. The MPU0CSR0 register spr number is 0d1014, and can be written as follows:

```
# Disable and Invalidate core MPU
mtspr 1014, 0x00000002     # set MPUFI in MPU0CSR0 to trigger MPU invalidation
```

The invalidation process typically takes three core clock cycles, and once complete the MPUFI bit is automatically reset to 0b0. It is recommended to wait for this to complete before performing any subsequent steps.

7.1.2. Configure MAS0-3

Once the MPU has been disabled and invalidated, the next step is to program the MAS0-3 registers for each of the required descriptors, depending on the level of protection required in the application. In this example the following configuration will be maintained across each region:

Table 2. MAS Example Common Bit Settings

Register Bit	Setting	Comment
MAS0[VALID]	0b1	MPU entry is valid
MAS0[IPROT]	0b1	Entry is protected from invalidation
MAS0[SEL]	0b10	Select MPU[2]
MAS0[UAMSK]	0b000	No upper address bits are masked during range comparison
MAS0[UW]	0b1	User mode has write permission
MAS0[SW]	0b1	Supervisor mode has write permission
MAS0[UX/UR]	0b1	User mode has execute/read permission
MAS0[SX/SR]	0b1	Supervisor mode has execute/read permission
MAS0[IOVR]	0b0	No override of I attribute by entry
MAS0[GOVR]	0b0	No override of G attribute by entry
MAS0[I]	0b0	Region is considered cacheable
MAS0[G]	0b0	Accesses are not guarded
MAS1[TID]	0x00	Entry is global, matches any PID value
MAS1[TIDMSK]	0x00	No region ID masking (TID=0 so any PID value is a match)

[2] This is required even though only one MPU instance is present in each core

The MAS2 and MAS3 registers then are used for the UPPER_BOUND and LOWER_BOUND values for the effective address comparison respectively, as described in Section 6.

In this example since the above bits of the MAS0 and MAS1 registers will not change, it is only required to set the MAS0[INST], MAS0[SHD] and MAS0[ESEL] bits for the required range covered by that region descriptor.

To cover the full range of memories of the RRU device for both instruction and data accesses and ensure any access outside of these will generate an ISI or DSI, the following ranges need to be considered.

Table 3. S32R274 Memory Regions

Area	Address Range	Type	Region Descriptor
Flash	0x00F98000 -- 0x0117FFFF	Data AND Instruction	0 (Data AND Instruction)
SRAM	0x40000000 -- 0x4017FFFF	Data	Data 1
PBRIDGE	0xF8000000 -- 0xFFFFFFFF	Data	Data 2
UTEST	0x00400000 -- 0x00403FFF	Data	Data 3
DFLASH	0x00800000 -- 0x0080FFFF	Data	Data 4
DMEM	0x50800000 -- 0x5080FFFF	Data	Data 5

These can be configured by first programming each of the MAS registers with the data required:

```
# Set z4 Code Range (Entire Flash) - Instruction Region 0
mtspr 624, 0xE1000F30            #Write MAS0 – ESEL = 0, INST = 1, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x0117FFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x00F98000            #Write MAS3 – LOWER_BOUND
```

Once these have been programmed it is then required to write the data to the MPU, which is done as follows:

```
#Write Entry
mpuwe
mpusync
```

This process can then be repeated to program the rest of the region descriptors:

```
# Set z4 Data Range (Entire Flash) - Data Region 0
mtspr 624, 0xE0000F30            #Write MAS0 – ESEL = 0, INST = 0, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x0117FFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x00F98000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync

# Set z4 Data Range (SRAM) - Data Region 1
mtspr 624, 0xE0010F30            #Write MAS0 – ESEL = 1, INST = 0, SHD = 0
```

```
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x4017FFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x40000000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync

# Set z4 Data Range (PBRIDGE) - Data Region 2
mtspr 624, 0xE0020F30            #Write MAS0 – ESEL = 2, INST = 0, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0xFFFFFFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0xF8000000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync

# Set z4 Data Range (UTEST) - Data Region 3
mtspr 624, 0xE0030F30            #Write MAS0 – ESEL = 3, INST = 0, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x00403FFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x00400000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync

# Set z4 Data Range (DFLASH) - Data Region 4
mtspr 624, 0xE0040F30            #Write MAS0 – ESEL = 4, INST = 0, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x0080FFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x00800000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync

# Set z4 Data Range (DMEM) - Data Region 5
```

```
mtspr 624, 0xE0050F30            #Write MAS0 – ESEL = 5, INST = 0, SHD = 0
mtspr 625, 0x00000000            #Write MAS1 – TID = 0, TIDMSK = 0
mtspr 626, 0x5080FFFF            #Write MAS2 – UPPER_BOUND
mtspr 627, 0x50800000            #Write MAS3 – LOWER_BOUND

#Write Entry
mpuwe
mpusync
```

7.1.3. Enable the MPU

Now that all regions available on the device are covered the MPU is fully configured and ready to be enabled. To enable the MPU write 0b1 to MPU0CSR0[MPUEN]. From this point onwards all address accesses will be monitored by the MPU. As previously mentioned it is critical to ensure that the relevant interrupt routines are configured and enabled before this step is completed.

```
#Enable core MPU
mtspr 1014, 0x0001
```

8. Fault handling

Once the MPU is enabled the system is protected against illegal accesses based on the configuration applied. In the event of an illegal access the relevant interrupt (DSI or ISI) will be raised to the core. The handling of this interrupt is application dependant and can be tailored to provide the desired reaction to this specific type of fault. To aid this, information can be read from the CPU registers at the time of the illegal access to determine the cause of the fault. These registers are detailed in the following sections but consult the device reference manual 'Core Description' section for more details on the specific bit fields shown.

8.1. Data Storage Interrupt

Table 4 - DSI CPU register settings

Register	Setting description					
SRR0	Set to the effective address of the excepting load/store instruction.					
SRR1	Set to the contents of the MSR at the time of the interrupt					
MSR	SPV	0	FP	0	FE1	0
	WE	0	ME	—	IS	0
	CE	—	FE0	0	DS	0
	EE	0	DE	—	PMM	0
	PR	0			RI	—
ESR	Access:		[ST], [SPV], VLEMI. All other bits cleared.			
MCSR	Unchanged					
DEAR	For Access Control exceptions, set to the effective address of the access that caused the violation.					
Vector	$IVPR_{0:23}$ || 0x20					

8.2. Instruction Storage Interrupt

Table 5 - ISI CPU register settings

Register	Setting description					
SRR0	Set to the effective address of the excepting instruction.					
SRR1	Set to the contents of the MSR at the time of the interrupt					
MSR	SPV	0	FP	0	FE1	0
	WE	0	ME	—	IS	0
	CE	—	FE0	0	DS	0
	EE	0	DE	—	PMM	0
	PR	0			RI	—
ESR	VLEMI. All other bits cleared.					
MCSR	Unchanged					
DEAR	Unchanged					
Vector	$IVPR_{0:23}$ || 0x30					

9. Configuration lessons learnt

It is important that both the Core MPU and SMPU are configured correctly to avoid scenarios where accesses to illegal address locations can cause undesired or unforeseen system behaviour.

E.g if the Core MPU is not correctly configured and the core prefetch unit attempts a prefetch from an address outside of the device memory map, then no core exception would be raised unless this unknown instruction was executed. The System MPU would detect such a violation when the read occurs on the XBAR, this would be reported to the FCCU but any fault reaction routine would not see a core violation reported, since the unknown instruction was not executed. Hence the core cannot resolve this violation.

It is therefore recommended that both the SMPU and the Core MPU are configured to cover the same memory regions for instruction and data accesses, to ensure that both system and core level accesses are fully protected.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12177
Rev. 0
04/2018