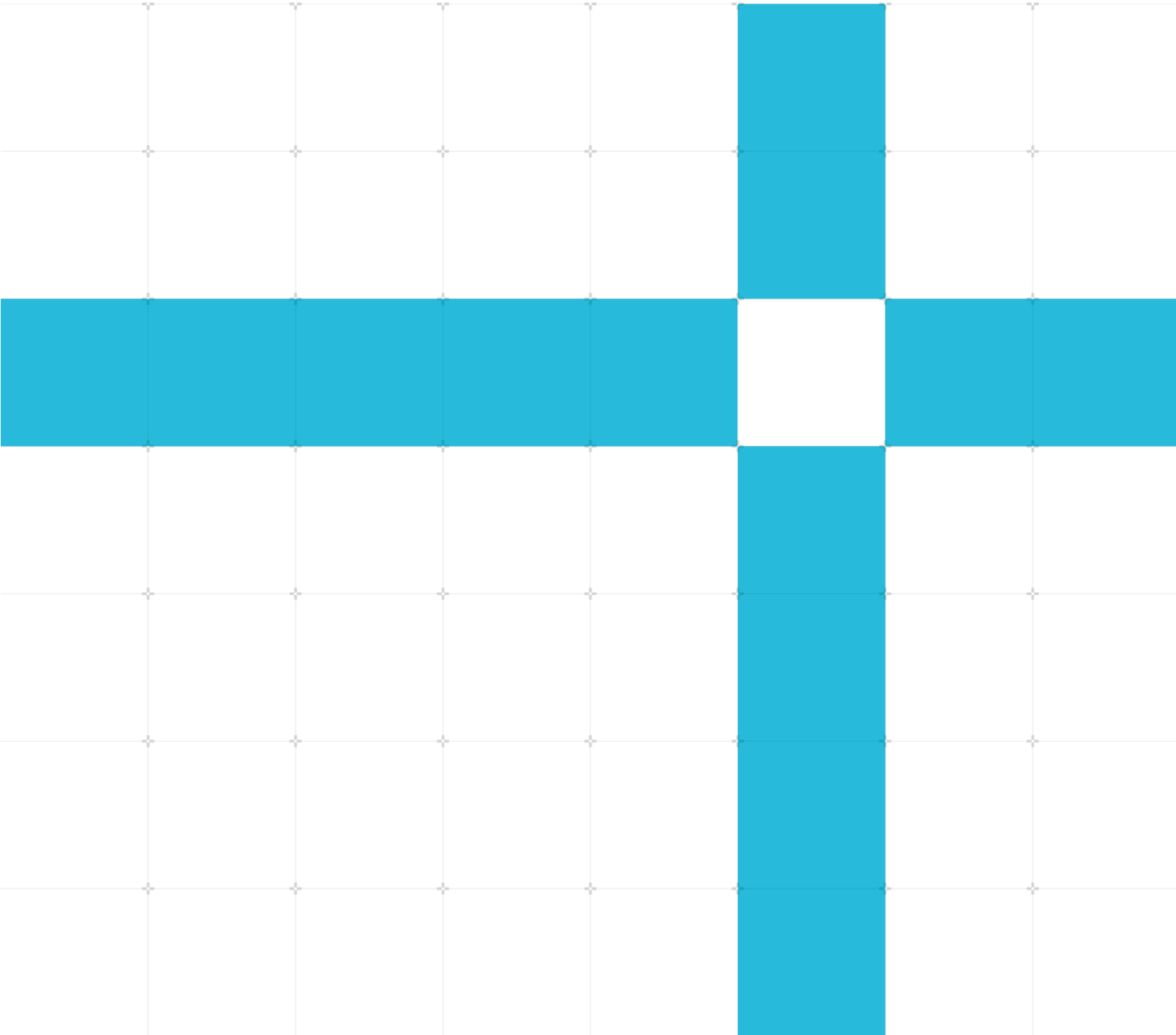
arm

Cortex®-A55 PMU Use-Cases

Application Note

Non-confidential

Document Issue: 1
Document ID: 107865



Release information

Document history

Issue	Date	Confidentiality	Change
1	05-10-2022	Non-Confidential	

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
(LES-PRE-20349)

Confidentiality Status

Feedback

Arm welcomes feedback on its products and documentation. To provide feedback on a product, create a ticket on **https://support.developer.arm.com**.

To provide feedback on this document, fill the following survey: **https://developer.arm.com/documentation-feedback-survey**.

Arm periodically provides updates and corrections to its technical content and Frequently Asked Questions (FAQs) on Arm Developer.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

To report offensive language in this document, email **terms@arm.com**.

Contents

1. Introduction

This document describes the typical use-cases to expand the knowledge further on the Cortex®-A55 PMU.

1.1. Intended audience

Arm customers using the performance monitoring unit.

1.2. Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

This document uses the following terms and abbreviations.

Terms and abbreviations

Term	Meaning
PMU	Performance Monitoring Unit
IFU	Instruction Fetch Unit
DPU	Data Processing Unit
DCU	Data Cache Unit
STB	Store Buffer
BIU	Bus Interface Unit
MMU	Memory Management Unit
IPC	Instructions per cycle
TLB	Translation Look-aside Buffer
CCI	Cache Coherent Interconnect
DSU	DynamIQ™ Shared Unit

Typographical conventions

Convention	Use	
italic	Citations.	
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.	
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.	
`monospace` **bold**	Language keywords when used outside example code.	
`monospace` underline	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.	
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`	
SMALL CAPITALS	Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.	
	Recommendations. Not following these recommendations might lead to system failure or damage.	
	Requirements for the system. Not following these requirements might result in system failure or damage.	
	Requirements for the system. Not following these requirements will result in system failure or damage.	
	An important piece of information that needs your attention.	
	A useful tip that might make it easier, better, or faster to perform a task.	

Convention	Use	
 Remember	A reminder of something important that relates to the information you are reading.	

1.3. Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on developer.arm.com/documentation. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

Arm products	Document ID	Confidentiality
Arm® Cortex®-A55 Core Technical Reference Manual	100442	Non-Confidential
Arm®Architecture Reference Manual for A-profile architecture	DDI 0487	Non-Confidential


Note

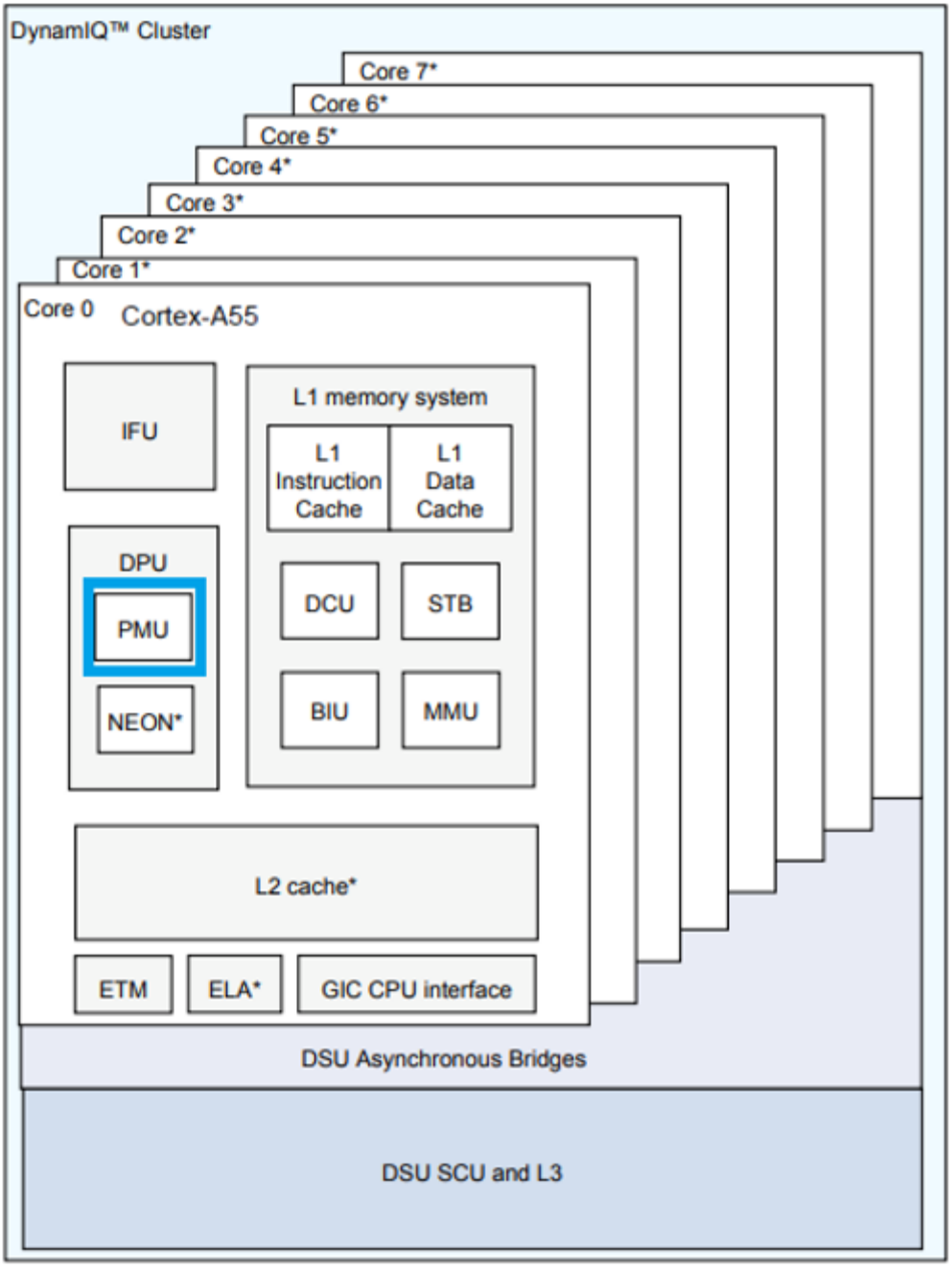
Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.
Adobe PDF reader products can be downloaded at **http://www.adobe.com**.

2. Problem Description

The Cortex®-A55 core includes performance monitors that enable you to gather various statistics on the operation of the core and its memory system during runtime. These provide useful information about the behavior of the core that you can use when debugging or profiling code.

The PMU provides six counters. Each counter can count any of the events available in the core. More information about PMU can be found in the Arm® Cortex®-A55 Core Technical Reference Manual, Debug Descriptions section, under PMU chapter.

There has been lot of questions from customers about setting up the PMU & using PMU for understanding bus traffic measurements from Cortex®-A55 through ACE/CHI interface. There needs to be a bridge to fill the gaps of using PMU to understand behavior of Arm core. This is done by extracting out detailed processor events that describe the processor behavior. This Application Note explains the typical use-cases to expand the knowledge further on the Cortex A55 PMU.


DynamIQ™ Cluster
Core 7*
Core 6*
Core 5*
Core 4*
Core 3*
Core 2*
Core 1*
Core 0
Cortex-A55
IFU
L1 memory system
L1 Instruction Cache
L1 Data Cache
DPU
PMU
NEON*
DCU
STB
BIU
MMU
L2 cache*
ETM
ELA*
GIC CPU interface
DSU Asynchronous Bridges
DSU SCU and L3

* Optional

3. Use-cases

Arm recommends the following use-cases to demonstrate the power of PMU to address this problem. They are described in the following topics:

- CPU Instruction Execution Rate.
- CPU Stall Analysis.
- Cache Performance Analysis.
- TLBs Performance Analysis.
- Memory Instructions Analysis.
- Bus bandwidth Analysis.
- External Memory Traffic Analysis.

These use-cases are exercised with a test-code in assembly. They were then simulated, which also produces a tarmac trace log that are used to observe the results.

The code examples referred in the following sections are in Aarch32 (32-bit assembly level coding).

3.1. PMU Setup

The PMU enable and disable sequences are common to all use-cases mentioned in this Application Note. Only step 2 is different and bespoke to each of these use-cases. Step 2 for each use-case is discussed under respective sections (section 3.3 onwards).

3.1.1. PMU Enable

This section explains the 4 steps involved in the process of enabling PMU.

4 step Enable of PMU

Step 1: Clear all counters (PMCR)

```
// Clear all counters
mrc     p15, 0, r0, c9, c12, 0          // Read PMCR
bic     r0, r0, #0xF                    // Clear PMCR.{D,C,P,E}
mcr     p15, 0, r0, c9, c12, 0          // Write PMCR
isb                                     // Instruction barrier
```

Step 2: Set-up event counters & event types

There is an example sequence below and it will differ with different use-cases.

Step 2 for each use-case is discussed under respective sections (section 3.3 onwards).

- Select register - PMSELR
- Select event – PMEVTYPER0

```
    .equ  PMU_CNTR_INST_RETIRED     ,0x0 // Counter 0 is selected from
available 6 counters
    .equ PMU_EVENT_INST_RETIRED   ,0x0008       // Event ID for INST_RETIRED
    // Set up counter for INST_RETIRED
    mov     r0, #PMU_CNTR_INST_RETIRED          // to be Written into PMSELR
    mov     r1, #PMU_EVENT_INST_RETIRED         // to be Written into
PMEVTYPER0
    bl      setup_evcntr                        // function does the writes

    .equ  PMU_CNTR_CPU_CYCLES ,0x1// Counter 1 is selected from available 6
counters

    .equ PMU_EVENT_CPU_CYCLES   ,0x0011   // Event ID for CPU_CYCLES
    // Set up counter for CPU_CYCLES
    mov     r0, #PMU_CNTR_ CPU_CYCLES           // to be Written into PMSELR
    mov     r1, #PMU_EVENT_ CPU_CYCLES          // to be Written into
PMEVTYPER0
    bl      setup_evcntr                        // function does the writes

    // Routine to set up event counter
    .type setup_evcntr, %function
    setup_evcntr:
    mcr     p15, 0, r0, c9, c12, 5              // Write PMSELR
    isb
    mcr     p15, 0, r1, c9, c13, 1              // Write PMEVTYPER0
    isb
    bx lr                                             // Return
```

Step 3: Clear overflow status and enable (PMOVSR & PMCNTENSET)

```
    // Clear overflow status and enable
    ldr     r1, =0xFFFFFFFF
    mcr     p15, 0, r1, c9, c12, 3              // Write PMOVSR
    mcr     p15, 0, r1, c9, c12, 1              // Write PMCNTENSET
```

Step 4: Reset and enable counters (PMCR)

```
    // Reset and enable counters
    mrc     p15, 0, r0, c9, c12, 0              // Write PMCR
```

```
orr     r0, r0, #0x7                            // Set PMCR.{C,P,E}
mcr     p15, 0, r0, c9, c12, 0              // Write PMCR
isb
```

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

3.1.2. PMU Disable

This section explains the 2 steps involved in the process of disabling PMU.

2 Step Disable of PMU

Step 1: Disable counters (PMCR)

```
// Disable counters
mrc     p15, 0, r0, c9, c12, 0              // Read PMCR
isb
bic     r0, r0, #0x1
mcr     p15, 0, r0, c9, c12, 0              // Write PMCR
isb
```

Step 2: Sample Event counters

- Select register - PMSELR
- Select event - PMEVTYPER0

```
    mov     r0, #PMU_CNTR_INST_RETIRED    // Write PMSELR & Read event counter
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

```
// Routine to read & move counter value to register r0
.type rd_print_evcntr, %function
rd_print_evcntr: mcr     p15, 0, r0, c9, c12, 5        // Write PMSELR
isb
mrc     p15, 0, r0, c9, c13, 2              // Read PMXEVCNTR into r0
isb // r0 contains the counter value for the event
```

3.2. CPU Instruction Execution rate

The formula to calculate the CPU performance with Instructions Per Cycle (IPC) is as follows:

$$IPC = \frac{INST\_RETIRED}{CPU\_CYCLES}$$

When this ratio is higher, it means that the processor has a high instruction execution rate.

The code snippet which can be used to measure IPC is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

- Select register - PMSELR
- Select event - PMEVTYPER0

```
    .equ  PMU_CNTR_INST_RETIRED     ,0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_INST_RETIRED    ,0x0008       // Event ID for INST_RETIRED
    // Set up counter for INST_RETIRED
    mov     r0, #PMU_CNTR_INST_RETIRED          // to be Written into PMSELR
    mov     r1, #PMU_EVENT_INST_RETIRED         // to be Written into
PMEVTYPER0
    bl      setup_evcntr                        // function does the writes


    .equ  PMU_CNTR_CPU_CYCLES ,0x1// Counter 1 is selected from available 6
counters


    .equ PMU_EVENT_CPU_CYCLES    ,0x0011   // Event ID for CPU_CYCLES
    // Set up counter for CPU_CYCLES
    mov     r0, #PMU_CNTR_CPU_CYCLES            // to be Written into PMSELR
    mov     r1, #PMU_EVENT_CPU_CYCLES           // to be Written into PMEVTYPE
    bl      setup_evcntr                        // function does the writes
```

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

- Select register - PMSELR
- Select event - PMEVTYPER0

```
    mov     r0, #PMU_CNTR_INST_RETIRED    // Write PMSELR & Read event counter
into r0
```

```
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value


    mov     r0, # PMU_CNTR_CPU_CYCLES
    bl      rd_print_evcntr   // Read event counter into r0 to form the
return value
```

3.3. CPU Stall Analysis

Pipeline Analysis can be done on the CPU Instruction Pipeline. This can be done by observing the number of times the CPU's Pipeline was stalled. Stalls might be related to Frontend or Backend of the pipeline.

Event ID	PMU Event
0x23	STALL_FRONTEND
0x24	STALL_BACKEND

Backend Pipeline stalls can be related to memory access (Load/Store).

Event ID	PMU Event
0xE7	STALL_BACKEND_LD
0xE8	STALL_BACKEND_ST

TLB related front & backend pipeline stalls (LD/ST) also can be informative to understand the CPU pipeline behavior during code execution.

Event ID	PMU Event
0xE2	STALL_FRONTEND_TLB
0xEA	STALL_BACKEND_LD_TLB
0xEC	STALL_BACKEND_ST_TLB

The code snippet which can be used to understand CPU Pipeline stalls is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    equ  PMU_CNTR_STALL_FRONTEND, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_STALL_FRONTEND, 0x0023// Event ID for STALL_FRONTEND
    // Set up counter for STALL_FRONTEND
    mov     r0, #PMU_CNTR_STALL_FRONTEND  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_STALL_FRONTEND// to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_STALL_FRONTEND            // Write PMSELR & Read
event counter into r0

    bl      rd_print_evcntr                          // Read PMXEVCNTR into
r0 as function return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly other events in this section can be PMU enabled, triggered, and sampled after disable to observe processor's behavior based on that event.

3.4. Cache Performance Analysis

The following ratios will become useful based on number of cache levels configured in the core and system design.

Each level of cache has cache access event & cache refill rate finding event counters. These can be used to calculate the cache performance measurement, like cache Miss/Refill rate.

3.4.1. L1 Data Cache

The formula to calculate L1 Data cache refill rate is as follows:

L1 Data cache REFILL RATE =	L1D_CACHE_REFILL
	L1D_CACHE

When this ratio is higher, it means that the L1 Data cache misses are higher.

The code snippet which can be used to measure L1 Data cache refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L1D_CACHE_REFILL, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L1D_CACHE_REFILL, 0x0003    // Event ID for
L1D_CACHE_REFILL

    // Set up counter for L1D_CACHE_REFILL

    mov     r0, #PMU_CNTR_L1D_CACHE_REFILL     // to be Written into PMSELR

    mov     r1, #PMU_EVENT_L1D_CACHE_REFILL    // to be Written into
PMEVTYPER0
```

```
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L1D_CACHE_REFILL     // Write PMSELR & Read event
counter into r0
    bl      rd_print_evcntr                     // Read PMXEVCNTR into r0 as
function return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L1D_CACHE (Event ID: 0x04) can be PMU enabled, triggered, and sampled after disable to calculate L1 Data cache REFILL RATE.

L1 refill rate will result in L2 access rate. Another thing to note is, L1 cache miss may result in some L1 Write-Backs if eviction should be done for new cache line refills.

3.4.2. L1 Data Cache Read

The formula to calculate L1 Data cache read refill rate is as follows:

L1 Data cache READ REFILL RATE =	L1D_CACHE_REFILL_RD
	L1D_CACHE_RD

When this ratio is higher, it means that the L1 Data cache read misses are higher.

The code snippet which can be used to measure L1 Data cache read refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L1D_CACHE_REFILL_RD, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L1D_CACHE_REFILL_RD, 0x0042 // Event ID for
L1D_CACHE_REFILL_RD
    // Set up counter for L1D_CACHE_REFILL_RD
    mov     r0, #PMU_CNTR_L1D_CACHE_REFILL_RD  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L1D_CACHE_REFILL_RD // to be Written into
PMEVTYPER0
    bl      setup_evcntr                            // function does the
writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L1D_CACHE_REFILL_RD // Write PMSELR & Read event
counter into r0

    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L1D_CACHE_RD (Event ID: 0x40) can be PMU enabled, triggered, and sampled after disable to calculate L1 Data cache READ REFILL RATE.

3.4.3. L1 Data Cache Write

The formula to calculate L1 Data cache write refill rate is as follows:

L1 Data cache WRITE REFILL RATE =	L1D_CACHE_REFILL_WR
	L1D_CACHE_WR

When this ratio is higher, it means that the L1 Data cache write misses are higher.

The code snippet which can be used to measure L1 Data cache write refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L1D_CACHE_REFILL_WR, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L1D_CACHE_REFILL_WR, 0x0043 // Event ID for
L1D_CACHE_REFILL_WR
    // Set up counter for L1D_CACHE_REFILL_WR
    mov     r0, #PMU_CNTR_L1D_CACHE_REFILL_WR  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L1D_CACHE_REFILL_WR // to be Written into
PMEVTYPER0
    bl      setup_evcntr                       // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L1D_CACHE_REFILL_WR // Write PMSELR & Read event
counter into r0

    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

function rd_print_evcntr is defined in section 3.1.

Similarly, L1D_CACHE_WR (Event ID: 0x041) can be PMU enabled, triggered, and sampled after disable to calculate L1 Data cache WRITE REFILL RATE.

3.4.4. L2 Data cache

The formula to calculate L2Data refill rate is as follows:

L2 Data cache REFILL RATE =	L2D_CACHE_REFILL
	L2D_CACHE

When this ratio is higher, it means that the L2 Data cache misses are higher.

The code snippet which can be used to measure L2 Data cache refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L2D_CACHE_REFILL, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L2D_CACHE_REFILL, 0x0017     // Event ID for STALL_FRONTEND
    // Set up counter for L2D_CACHE_REFILL
    mov     r0, #PMU_CNTR_L2D_CACHE_REFILL      // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L2D_CACHE_REFILL     // to be Written into
PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L2D_CACHE_REFILL     // Write PMSELR & Read event
counter into r0
```

```
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L2D_CACHE (Event ID: 0x16) can be PMU enabled, triggered, and sampled after disable to calculate L2 Data cache REFILL RATE.

L2 refill rate will result in L3 access rate. Another thing to note is, L2 cache miss may result in some L2 Write-Backs if eviction should be done for new cache line Refills.

3.4.5. L2 Data cache Read

The formula to calculate L2 Data cache read refill rate is as follows:

L2 Data cache READ REFILL RATE =	L2D_CACHE_REFILL_RD
	L2D_CACHE_RD

When this ratio is higher, it means that the L2 Data cache read misses are higher.

The code snippet which can be used to measure L2 Data cache read refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L2D_CACHE_REFILL_RD, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L2D_CACHE_REFILL_RD, 0x0052 // Event ID for
L2D_CACHE_REFILL_RD
    // Set up counter for L2D_CACHE_REFILL_RD
    mov     r0, #PMU_CNTR_L2D_CACHE_REFILL_RD  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L2D_CACHE_REFILL_RD // to be Written into
PMEVTYPER0
    bl      setup_evcntr                            // function does the
writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L2D_CACHE_REFILL_RD // Write PMSELR & Read event
counter into r0
    bl      rd_print_evcntr    // Read PMXEVCNTR into r0 as function return
value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L2D_CACHE_RD (Event ID: 0x50) can be PMU enabled, triggered, and sampled after disable to calculate L2 Data cache READ REFILL RATE.

3.4.6. L2 Data cache Write

The formula to calculate L2 Data cache write refill rate is as follows:

L2 Data cache WRITE REFILL RATE =	L2D_CACHE_REFILL_WR
	L2D_CACHE_WR

When this ratio is higher, it means that the L2 Data cache write misses are higher.

The code snippet which can be used to measure L2 Data cache write refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L2D_CACHE_REFILL_WR, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L2D_CACHE_REFILL_WR, 0x0053 // Event ID for
L2D_CACHE_REFILL_WR
    // Set up counter for L2D_CACHE_REFILL_WR
    mov     r0, #PMU_CNTR_L2D_CACHE_REFILL_WR  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L2D_CACHE_REFILL_WR // to be Written into
PMEVTYPER0
    bl      setup_evcntr                       // function does the
writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L2D_CACHE_REFILL_WR // Write PMSELR & Read event
counter into r0
    bl      rd_print_evcntr    // Read PMXEVCNTR into r0 as function return
value
```

function rd_print_evcntr is defined in section 3.1

Similarly, L2D_CACHE_WR (Event ID: 0x51) can be PMU enabled, triggered, and sampled after disable to calculate L2 Data cache WRITE REFILL RATE.

3.4.7. L3 Data cache

The formula to calculate L3Data refill rate is as follows:

L3 Data cache REFILL RATE =	L3D_CACHE_REFILL
	L3D_CACHE

When this ratio is higher, it means that the L3 Data cache misses are higher.

The code snippet which can be used to measure L3 Data cache refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L3D_CACHE_REFILL, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L3D_CACHE_REFILL, 0x002A    // Event ID for
L3D_CACHE_REFILL
    // Set up counter for L3D_CACHE_REFILL
    mov     r0, #PMU_CNTR_L3D_CACHE_REFILL     // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L3D_CACHE_REFILL    // to be Written into
PMEVTYPER0
    bl      setup_evcntr                       // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L3D_CACHE_REFILL    // Write PMSELR & Read event
counter into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L3D_CACHE (Event ID: 0x2B) can be PMU enabled, triggered, and sampled after disable to calculate L3 Data cache REFILL RATE.

L3 refill rate will result in external memory access (BUS_ACCESS event). Another thing to note is, L3 cache miss may result in some L3 Write-Backs if eviction should be done for new cache line Refills.

3.4.8. L3 Data cache Read

The formula to calculate L3 Data cache read refill rate is as follows:

L3 Data cache READ REFILL RATE =	L3D_CACHE_REFILL_RD
	L3D_CACHE_RD

When this ratio is higher, it means that the L3 Data cache read misses are higher.

The code snippet which can be used to measure L3 Data cache read refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L3D_CACHE_REFILL_RD, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_L3D_CACHE_REFILL_RD, 0x00A2 // Event ID for
L3D_CACHE_REFILL_RD
    // Set up counter for L3D_CACHE_REFILL_RD
    mov     r0, #PMU_CNTR_L3D_CACHE_REFILL_RD  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L3D_CACHE_REFILL_RD // to be Written into
PMEVTYPER0
    bl      setup_evcntr                              // function does the
writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L3D_CACHE_REFILL_RD //Write PMSELR & Read event
count into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L3D_CACHE_RD (Event ID: 0xA0) can be PMU enabled, triggered, and sampled after disable to calculate L3 Data cache READ REFILL RATE.

3.4.9. Last Level Data Cache Read

The formula to calculate last level, LL Data cache read refill rate is as follows:

LL Data cache REFILL READ RATE =	LL_CACHE_MISS_RD
	LL_CACHE_RD

When this ratio is higher, it means that the LL Data cache read misses are higher.

The code snippet which can be used to measure LL Data cache refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_LL_CACHE_MISS_RD, 0x0// Counter 0 is selected from
available 6 counters


    .equ PMU_EVENT_LL_CACHE_MISS_RD, 0x0036    // Event ID for
LL_CACHE_MISS_RD
    // Set up counter for LL_CACHE_MISS_RD
    mov     r0, #PMU_CNTR_LL_CACHE_MISS_RD     // to be Written into PMSELR
    mov     r1, #PMU_EVENT_LL_CACHE_MISS_RD    // to be Written into
PMEVTYPER0
    bl      setup_evcntr                       // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_LL_CACHE_MISS_RD//Write PMSELR & Read eventcount
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, LL_CACHE_RD (Event ID: 0x37) can be PMU enabled, triggered, and sampled after disable to calculate LL Data cache READ REFILL RATE.

This LL Data cache READ REFILL RATE will be same as L3 Data cache READ REFILL RATE if L3 is the last cache level before external memory.

3.5. TLBs Performance Analysis

The following ratios will become useful based on number of cache levels configured in the core and system design.

3.5.1. L1 Instruction TLB

The formula to calculate L1 Instruction TLB refill rate is as follows:

L1 Instruction TLB REFILL RATE =	L1I_TLB_REFILL
	L1I_TLB

When this ratio is higher, it means that the L1 Instruction TLB misses are higher.

The code snippet which can be used to measure L1 Instruction TLB refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L1I_TLB_REFILL, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_L1I_TLB_REFILL, 0x0002// Event ID for L1I_TLB_REFILL
    // Set up counter for L1I_TLB_REFILL
    mov     r0, #PMU_CNTR_L1I_TLB_REFILL // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L1I_TLB_REFILL// to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L1I_TLB_REFILL //Write PMSELR & Read event count
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L1I_TLB (Event ID: 0x26) can be PMU enabled, triggered, and sampled after disable to calculate L1 Instruction TLB REFILL RATE.

3.5.2. L1 Data TLB

The formula to calculate L1 Data TLB refill rate is as follows:

L1 Data TLB REFILL RATE =	L1D_TLB_REFILL
	L1D_TLB

When this ratio is higher, it means that the L1 Data TLB misses are higher.

The code snippet which can be used to measure L1 Data TLB refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L1D_TLB_REFILL, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_L1D_TLB_REFILL, 0x0005       // Event ID for L1D_TLB_REFILL
    // Set up counter for L1D_TLB_REFILL
```

```
    mov     r0, #PMU_CNTR_L1D_TLB_REFILL        // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L1D_TLB_REFILL       // to be Written into
PMEVTYPER0
    bl      setup_evcntr                                // function does the
writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L1D_TLB_REFILL //Write PMSELR & Read event count
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L1D_TLB (Event ID: 0x25) can be PMU enabled, triggered, and sampled after disable to calculate L1 Data cache TLB REFILL RATE.

3.5.3. L2 Instruction TLB

The formula to calculate L2 Instruction TLB refill rate is as follows:

L2 Instruction TLB REFILL RATE =	L2I_TLB_REFILL
	L2I_TLB

When this ratio is higher, it means that the L2 Instruction TLB misses are higher.

The code snippet which can be used to measure L2 Instruction TLB refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L2I_TLB_REFILL, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_L2I_TLB_REFILL, 0x002E// Event ID for L2I_TLB_REFILL
    // Set up counter for L2I_TLB_REFILL
    mov     r0, #PMU_CNTR_L2I_TLB_REFILL // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L2I_TLB_REFILL// to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L2I_TLB_REFILL //Write PMSELR & Read event count
into r0

    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L2I_TLB (Event ID: 0x30) can be PMU enabled, triggered, and sampled after disable to calculate L2 Instruction TLB REFILL RATE.

3.5.4. L2 Data TLB

The formula to calculate L2 Data TLB refill rate is as follows:

L2 Data TLB REFILL RATE =	L2D_TLB_REFILL
	L2D_TLB

When this ratio is higher, it means that the L2 Data TLB misses are higher.

The code snippet which can be used to measure L2 Data TLB refill rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_L2D_TLB_REFILL, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_L2D_TLB_REFILL, 0x002D// Event ID for L2D_TLB_REFILL
    // Set up counter for L2D_TLB_REFILL
    mov     r0, #PMU_CNTR_L2D_TLB_REFILL // to be Written into PMSELR
    mov     r1, #PMU_EVENT_L2D_TLB_REFILL// to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_L2D_TLB_REFILL // Write PMSELR & Read eventcount
into r0
    bl      rd_print_evcntr    // Read PMXEVCNTR into r0 as function return
value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, L2D_TLB (Event ID: 0x2F) can be PMU enabled, triggered, and sampled after disable to calculate L2 Data TLB REFILL RATE.

3.6. Memory Instructions Analysis

Memory accesses based on the Instructions can help one understand how many data read and data writes make up the total memory accesses. MEM_ACCESS is one such event which counts the memory access due to load and store instructions.

The formula to calculate main memory read access rate is as follows:

MEM READ ACCESS RATIO =	MEM_ACCESS_RD
	MEM_ACCESS

When this ratio is higher, it means that there is lot of memory read accesses issued from Cortex®-A55.

The code snippet which can be used to measure main memory read access rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_MEM_ACCESS_RD, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_MEM_ACCESS_RD, 0x0066 // Event ID for MEM_ACCESS_RD
    // Set up counter for MEM_ACCESS_RD
    mov     r0, #PMU_CNTR_MEM_ACCESS_RD  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_MEM_ACCESS_RD // to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_MEM_ACCESS_RD // Write PMSELR & Read event count
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, MEM_ACCESS (Event ID: 0x13) can be PMU enabled, triggered, and sampled after disable to calculate MEM READ ACCESS RATIO.

The formula to calculate main memory write access rate is as follows:

MEM WRITE ACCESS RATIO =	MEM_ACCESS_WR
	MEM_ACCESS

When this ratio is higher, it means that there is lot of memory write accesses issued from Cortex®-A55.

The code snippet which can be used to measure main memory write access rate is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_MEM_ACCESS_WR, 0x0// Counter 0 is selected from available
6 counters


    .equ PMU_EVENT_MEM_ACCESS_WR, 0x0066 // Event ID for MEM_ACCESS_WR
    // Set up counter for MEM_ACCESS_WR
    mov     r0, #PMU_CNTR_MEM_ACCESS_WR  // to be Written into PMSELR
    mov     r1, #PMU_EVENT_MEM_ACCESS_WR // to be Written into PMEVTYPER0
    bl      setup_evcntr                        // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

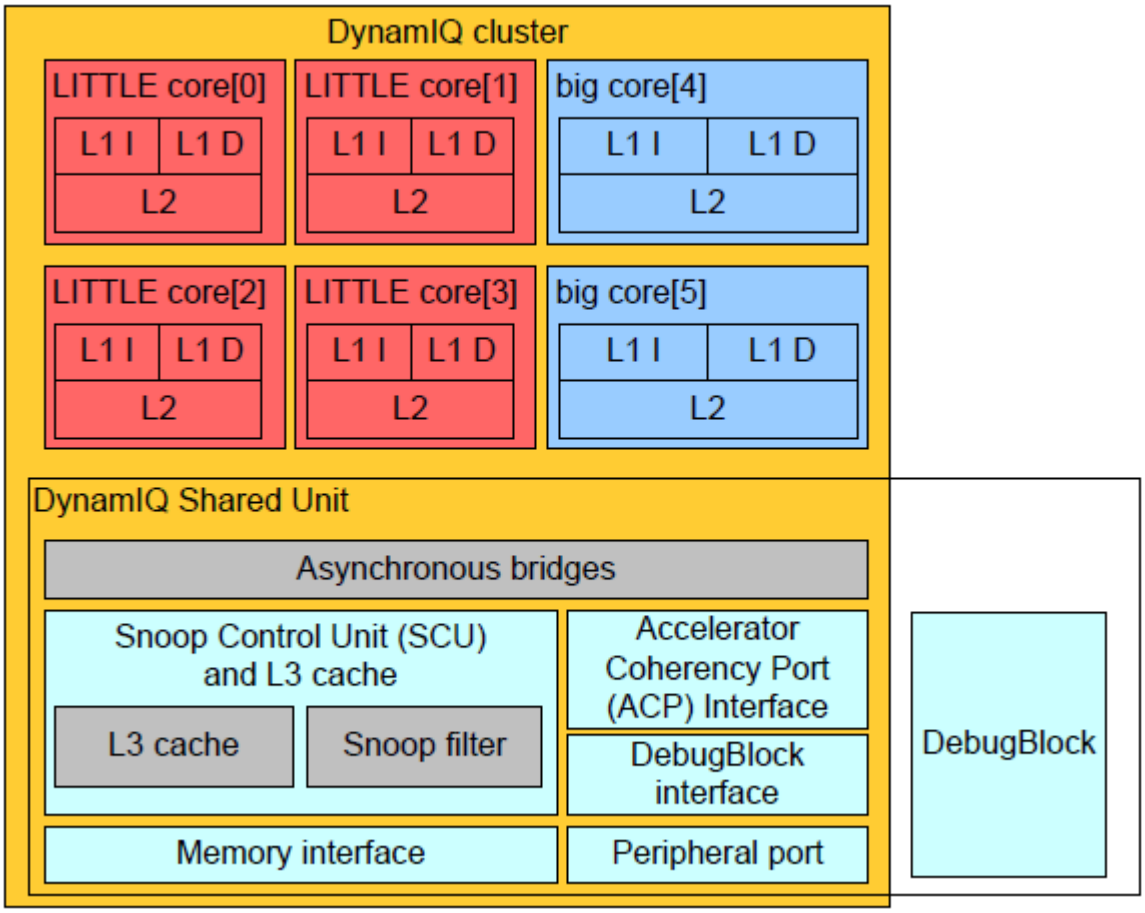
```
    mov     r0, # PMU_CNTR_MEM_ACCESS_WR // Write PMSELR & Read event count
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, MEM_ACCESS (Event ID: 0x13) can be PMU enabled, triggered, and sampled after disable to calculate MEM WRITE ACCESS RATIO.

3.7. Bus Bandwidth Analysis

As we have seen earlier, L3 cache miss results in BUS_ACCESS to memory external to DSU (DynamIQ Shared Unit). The Cortex®-A55 is a DSU-based core. In the below diagram, Cortex®-A55 could go into the red boxes (little cores).



It is very useful to do Bandwidth analysis based on the bus traffic from the DSU. This will particularly be handy to identify bottlenecks in the system.

The formula to calculate bus activity is as follows:

BUS ACTIVITY =	BUS_ACCESS
	BUS_CYCLES

When this ratio is higher, it means that there is lot of bus accesses issued from Cortex®-A55.

The code snippet which can be used to measure Bus Access per Cycle is as follows:

Please refer to section 3.1.1 to enable PMU and replace ONLY step 2 with the following:

Set-up event counters & event types

```
    .equ  PMU_CNTR_BUS_ACCESS, 0x0// Counter 0 is selected from available 6
counters


    .equ PMU_EVENT_BUS_ACCESS, 0x0019          // Event ID for BUS_ACCESS
    // Set up counter for BUS_ACCESS
    mov     r0, #PMU_CNTR_BUS_ACCESS           // to be Written into PMSELR
    mov     r1, #PMU_EVENT_BUS_ACCESS          // to be Written into
PMEVTYPER0
    bl      setup_evcntr                       // function does the writes
```

Function setup_evcntr is defined in section 3.1.

The events are triggered by the test for the event counters to run. These count values are then sampled after the disable of Performance Monitoring Unit.

Please refer to section 3.1.2 to disable PMU and replace ONLY step 2 with the following:

Sample Event counters

```
    mov     r0, # PMU_CNTR_BUS_ACCESS     // Write PMSELR & Read event counter
into r0
    bl      rd_print_evcntr         // Read PMXEVCNTR into r0 as function
return value
```

Function rd_print_evcntr is defined in section 3.1.

Similarly, BUS_CYCLES (Event ID: 0x1D) can be PMU enabled, triggered, and sampled after disable to calculate BUS ACTIVITY.

3.8. External Memory Traffic Analysis

It is important to understand about the traffic to external memory access. Here external means any memory outside DSU point of view.

Traffic analysis is done as,

BUS_ACCESS = Cacheable Traffic + Non-cacheable Traffic

Cacheable Traffic to external memory access

Cacheable Traffic = (L2D_CACHE_REFILL event +L2D_CACHE_WRITEBACK event) * cache line size, L2 is chosen as an example in the equation. Last level should be used in the equation.

The absolute counts recorded might vary because of pipeline effects, speculative executed LD/ST & data prefetches, cache coherency operations (flush & invalidate) in the background

Based on the last level before external memory we have 3 cases:

Case 1: L2 is the last level cache and there is no L3 or System Level Cache (SLC)

Cacheable Traffic = (L2D_CACHE_REFILL event +L2D_CACHE_WRITEBACK event) * cache line size

Case 2: L3 is the last level cache and there is no System Level Cache(SLC)

Cacheable Traffic = (L3D_CACHE_REFILL event +L3D_CACHE_WRITEBACK event) * cache line size

In Case 2, there could be snoop HIT/ MISS:

Case2a: If Snoop HIT in other cores L2, then no bus traffic.

Case2b: If Snoop MISS in other cores L2, then new bus traffic that gets reflected on BUS_ACCESS PMU event.

Case 3: SLC is the last level cache

There is System Level Cache between last level cache and DDR (external memory).

If Cache Coherent Inter-connect (CCI) exists in the design, then, PMU events of CCI like booker-ci can help in mapping the external memory accesses further.

Non-cacheable Traffic to external memory access

Non-cacheable Traffic = BUS_ACCESS-(L2D_CACHE_REFILL+L2D_CACHE_WRITEBACK) * cache line size

Non-Cacheable Traffic includes bus traffic due to write-streaming. In write streaming, cache is by-passed for long sequence of memory RD/WR access when threshold is met for L1 & L2 levels (Please refer to Arm®Architecture Reference Manual for A-profile architecture for more details on write streaming)

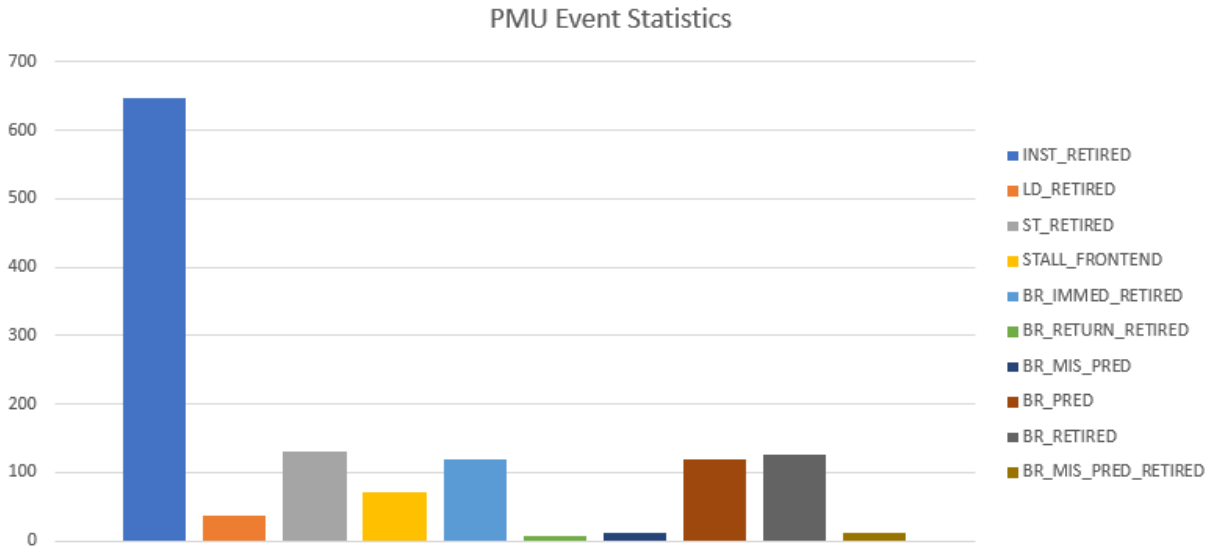
Similarly, write to NC (NC = non-cacheable normal memory) or Device memory, appears as a bus write.

The yellow highlights are the equation which will help in understanding the Bandwidth analysis going out from Arm core through the memory hierarchy all the way to external memory.

3.9. Summary

This Application Note explains about setting up the PMU & using PMU to understand bus traffic measurements from Cortex®-A55. Going further, this referred to some examples of how to use the PMU to understand the behaviour of Arm core. It was done by setting up necessary processor events that describes the processor behaviour. Further Information can be found from the Cortex®-A55 Technical Reference Manual and the Arm® Cortex®-A55 Core Integration Manual documents.

An example of Statistical view of PMU event counters is depicted in the following plot chart.



4. Appendix 1: Origin of source data

Application Note section	Source of data	Reference	Details
Section 3	Tests Developed & Run	https://git.research.arm.com/karram02/ananke_perf_monitoring	Code snippets extracted from git
Problem Description & Summary	441615	Case ID	How to use Cortex®-A55 PMU event counter to record a specific event
Problem Description & Summary	TAC674105	Case ID	Questions about L1/L2/L3 PMU Counters
Problem Description & Summary	TAC672983	Case ID	L1D refill event
Problem Description & Summary	00298096	Case ID	PMU events of A53 and A55
Section 4	Design Team Review	Peng Wang, Debug Expert of Cortex®-A55	Review comments shared