Arm® CoreSight™ Architecture

Performance Monitoring Unit Architecture

ARM IHI 0091 A.b

CoreSight Performance Monitoring Unit Architecture

Release information

Date	Version	Changes
2022/Oct/05	A.b	• Second non-confidential EAC release.
2020/Nov/04	A.a	• First non-confidential beta release.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Contents

CoreSight Performance Monitoring Unit Architecture

Preface

Document status

EAC release.

EAC quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release will be sufficiently stable and committed for product development.

About this book

I$_{BBMHH}$ This manual describes a standard *Performance Monitoring Unit* (PMU). A PMU primarily consists of *monitors* that measure a characteristic of a *component*. Monitors are often *event counters* that count *events* generated by the component. However, in some cases a PMU provides *monitors* that measure the state or an operational characteristic of the component.

R$_{GNQRW}$ In keeping with the rest of the architecture reference manuals, features which are optional are explicitly declared in this manual as being optional, and the presence of independent ID codes for features does not implicitly mean that the features are optional.

Using this book

IWZYHY This manual has the following sections:

Chapter 1 *Description*
An *informative* section describing the architecture and motivation.

Chapter 2 *Specification*
A *normative* section that specifies the mandatory aspects of the architecture. The *Specification* section uses Rules-based writing.

Chapter 3 *Extensions*
A *normative* section that specifies optional standard extensions to the architecture. The *Extensions* section uses Rules-based writing.

Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented* **and** **Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented***
Normative sections that provide the definitions of the registers added by this manual.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used in body text for terms, such as IMPLEMENTATION DEFINED, that have specific technical meanings described in the Arm Architecture Reference Manual.

Blue text

Indicates a link. This can be a cross-reference to another location within the document, or a URL such as http://developer.arm.com.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Rule.
- Information.
- Rationale.
- Implementation note.
- Software usage.

Rules are normative statements. An implementation that is compliant with this specification must conform to all Rules in this specification that apply to that implementation.

Rules must not be read in isolation. Where a particular feature is specified by multiple Rules, these are grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiiii}

- L is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

Content item classes

Rule

A Rule is a statement that does one or more of the following:

- Describes the behavior of a compliant implementation.
- Defines concepts or terminology.

A Rule is rendered with the label *R*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *The Every Computer Performance Book*. (ISBN 9781482657753) Bob Wescott.

[2] *Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture*. (ARM DDI 0598) Arm Limited.

[3] *Arm® Architecture Reference Manual, for A-profile architecture*. (ARM DDI 0487) Arm Limited.

[4] *ARM CoreSight Architecture Specification*. (ARM IHI 0029) Arm Limited.

[5] *Armv8-M Architecture Reference Manual*. (ARM DDI 0553) Arm Limited.

[6] *Arm Realm Management Extension (RME) System Architecture*. (ARM DEN 0129) Arm Limited.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (CoreSight Performance Monitoring Unit Architecture).
- The number (ARM IHI 0091 A.b).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Progressive terminology statement

Arm values inclusive communities.

Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive terms.

If you find offensive terms in this document, please contact terms@arm.com.

Chapter 1
Description

This section is *informative*.

1.1 About Performance Monitors

This manual describes a standard *Performance Monitoring Unit* (PMU). A PMU primarily consists of monitors that measure a characteristic of a *component*.

Monitors are often *event counters* that count *events* generated by the component. (For the purposes of the PMU architecture, a *cycle counter* is an *event counter* that counts the *cycle* event.) The architecture includes a mechanism to generate an interrupt when a counter reaches a threshold value.

In the CoreSight Performance Monitoring Unit Architecture, event counters are monotonically increasing.

However, in some cases a PMU provides *monitors* that measure the state or an operational characteristic of the component. For instance, a monitor might increment when a resource is allocated and decrement when the resource is deallocated, meaning it provides the current allocation level for the resource and is not monotonic.

A PMU might consist of a mix of such monitors and event counters. Where this manual uses the term monitor it can mean either an event counter or some other monitor, unless explictly stated.

An implementation of the CoreSight Performance Monitoring Unit Architecture can have up-to 128 monitors of up-to 64-bits in size, or up-to 256 monitors of up-to 32-bits in size.

A PMU has two main use models:

- 1.1.1 *Profiling and software optimization*.
- 1.1.2 *Monitoring*.

Both approaches typically use 1.1.3 *Sampling* to read the PMU.

PMUs are sometimes also described as *Hardware Performance Monitors* (HPM).

1.1.1 Profiling and software optimization

Profiling is a tool used in software optimization. Performance monitors are a hardware feature that can be used by profiling.

The main goal of optimization is usually reducing the elapsed time required to perform a task. Reducing power consumption might be a goal, but is usually a side-effect of reducing the elapsed time.

The focus for performance optimization, and the effort put into it, differs according to the application and market.

For instance, on server and *High-performance Computing* (HPC) systems software optimization is a mainstream activity. Improving performance to increase efficiency translates directly to the bottom line. As such, these systems are often marketed on their performance at key benchmarks. For example, SPEC, TPC, Linpack, and so on.

In other markets, performance merely has to be *good enough* to satisfy the user, and there might not be much benefit in making improvements beyond this.

For instance, in personal computing and mobile applications, performance might be measured against performance targets. For example, the time to respond to a keypress should be imperceptible, or the frame-rate in a game should match the hardware capabilities.

An application programmer might focus on profiling and optimizing use of frameworks and middleware. For example, reducing the number of calls to a graphics API.

There are also a wide-range of techniques used.

For instance, a developer might use a performance analysis tool to gain insights into the software. This might be augmented by expert systems that suggest improvements. Using this information, the developer makes changes to the source code, and uses the tools to confirm improvements.

Not all software is amenable to being modified. There is much *dusty deck* code that is either deemed *good enough* or too hard to optimize. Profiling can also be used as a mechanism to expose this code, and can also provide data for *Profile-guided Optimization* (PGO) that might improve its performance without modifying the source.

1.1.2 Monitoring

Monitoring is a technique used in system operations. Performance monitors are a hardware feature used in monitoring.

The goal of performance monitoring is to provide a system operator with answers that can be used to improve the efficiency of the system. Examples of the questions that answers are needed for are:

- "What is the system doing right now?"
- "Why is the system running slow?"
- "Can the system handle the upcoming peak load?"

These questions are taken from *The Every Computer Performance Book* [1].

The answers to these questions might cause the operator to, for example, increase or decrease capacity or to rearrange load on the available machines. Operators are not necessarily human. Systems might automate collecting and processing data from performance monitors, and making these changes.

An AMU is a particular kind of PMU used for monitoring. See 2.4 *Activity Monitor Units (AMUs)*.

1.1.3 Sampling

Two commonly used techniques for using performance monitors are time-based sampling and event-based sampling.

Figure 1.1 and Figure 1.2 show examples of time-based sampling tool user interfaces.

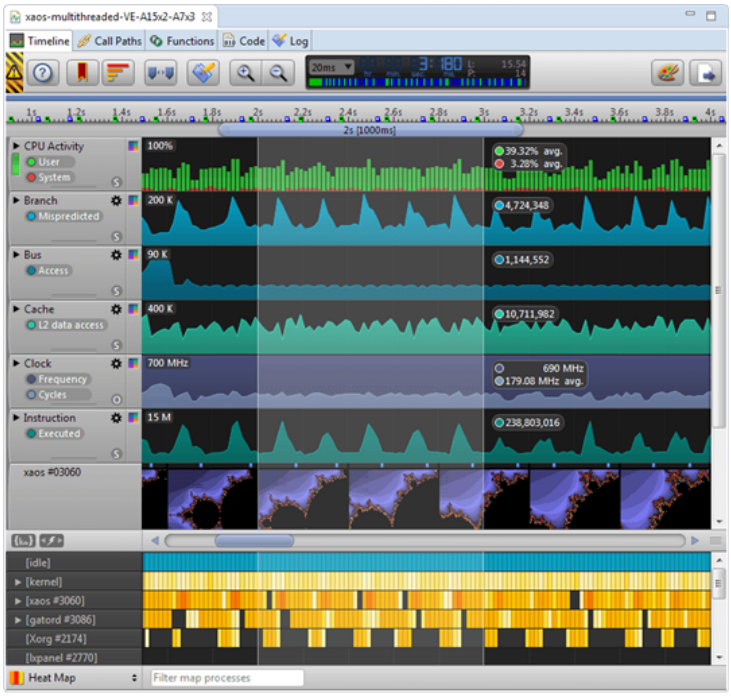


Figure 1.1: Time-based sampling with Arm DS-5 Streamline Performance Analyzer

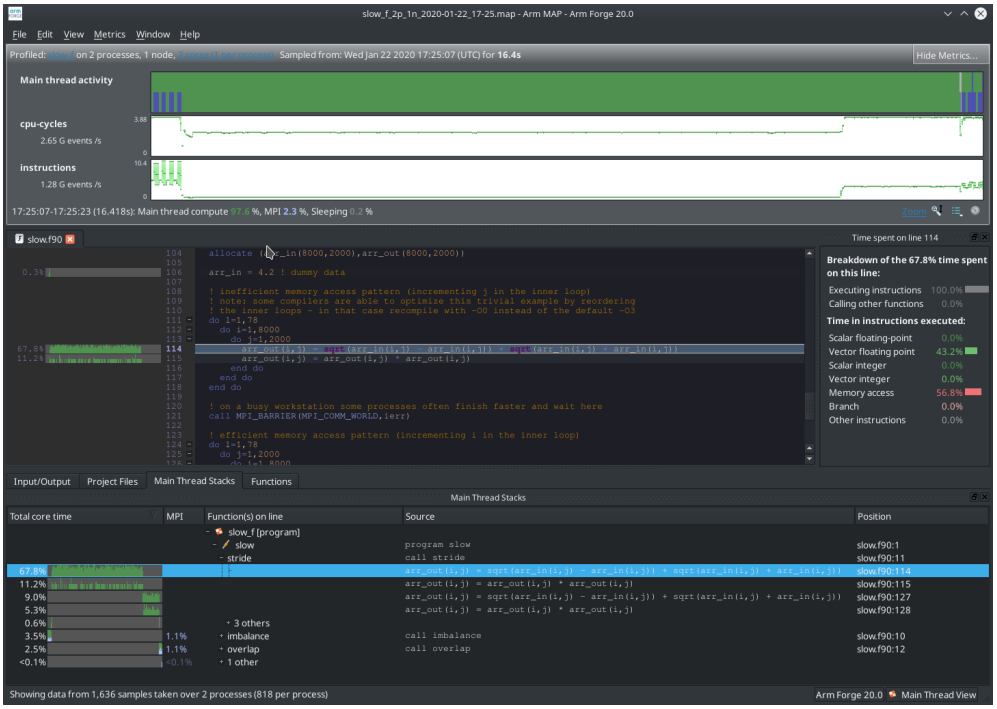
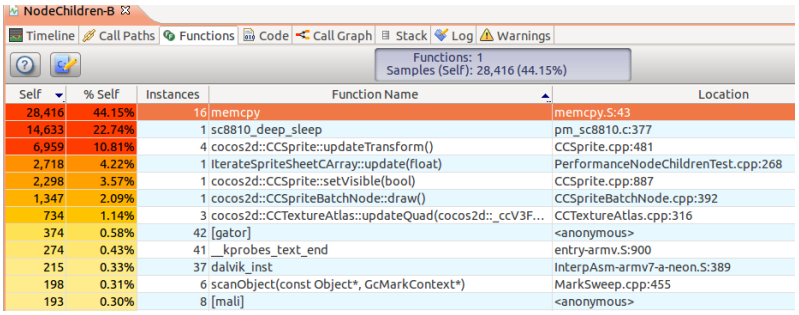


Figure 1.2: Source-level profiling with Arm MAP source-level profiler

Figure 1.3 shows an example of an event-based sampling tool user interface.



Self	% Self	Instances	Function Name	Location
28,416	44.15%	16	memcpy	memcpy.S:43
14,633	22.74%	1	sc8810_deep_sleep	pm_sc8810.c:377
6,959	10.81%	4	cocos2d::CCSprite::updateTransform()	CCSprite.cpp:481
2,718	4.22%	1	IterateSpriteSheetCArray::update(float)	PerformanceNodeChildrenTest.cpp:268
2,298	3.57%	1	cocos2d::CCSprite::setVisible(bool)	CCSprite.cpp:887
1,347	2.09%	1	cocos2d::CCSpriteBatchNode::draw()	CCSpriteBatchNode.cpp:392
734	1.14%	3	cocos2d::CCTextureAtlas::updateQuad(cocos2d::_ccV3F...	CCTextureAtlas.cpp:316
374	0.58%	42	[gator]	<anonymous>
274	0.43%	41	__kprobes_text_end	entry-armv.S:900
215	0.33%	37	dalvik_inst	InterpAsm-armv7-a-neon.S:389
198	0.31%	6	scanObject(const Object*, GcMarkContext*)	MarkSweep.cpp:455
193	0.30%	8	[mali]	<anonymous>

Figure 1.3: Hot-spot analysis from event-based sampling with Arm DS-5 Streamline Performance Analyzer

1.2 Rationale for a standard PMU architecture

Layered software design allows software to be configurable and reusable across many systems.

For example, a standard architecture for hardware PMU monitors allows a shared *PMU driver* software layer. The driver provides a common software abstraction of a PMU to profiling tools, requiring minimal target-specific information.

For instance, in a typical Arm processor PMU there are an IMPLEMENTATION DEFINED number of event counter monitors and a single fixed-function cycle counter. The PMU driver adapts to the number of event counters using the identification registers provided by the PMU. Each of the event counters can be configured to count any one event from an IMPLEMENTATION DEFINED set of events. The PMU driver does not need to be aware of what these events are, only how to program the event selection logic with an event number supplied by the profiling tool.

This architecture provides a hardware abstraction that allows such a software abstraction to be implemented. The Arm Architecture *Performance Monitors Extensions*, for both A-profile and M-profile, are based on this architecture.

1.3 What to measure

The following are suggestions of metrics that a performance monitor might be used to measure.

Utilization

Utilization (ρ) is a key measurement for any performance monitor, for two main reasons:

- If a resource is under-utilized, there is scope for improving performance.
- Performance drops dramatically as a resource reaches saturation, because of the additional waiting time in a system.

Figure 1.4 shows the ratio of Queuing time to Service time (S) varying over Utilization (ρ), by considering the total Response time (W) for both an M/M/1 queue ($W = S/(1-\rho)$) and an M/M/2 queue ($W = S/(1-\rho^2)$) using Little's Law.

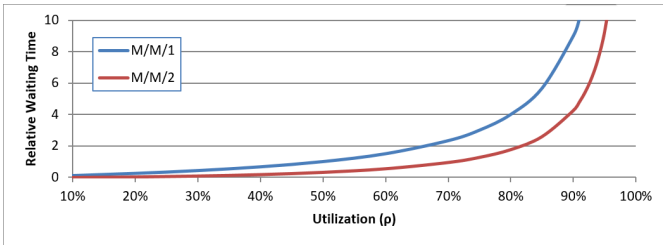


Figure 1.4: Queuing time as a function of utilization for M/M/1 and M/M/2 queues

Utilization is defined as the total amount of time the resource is busy (B) (not idle) during a period, divided by the length of the period (T). ($\rho = B \div T$)

For some interfaces, an access on the interface is defined as a sequence of units, each of which comprises a sequence of smaller units that are transmitted one-per-cycle on the interface link. For example, a packet interface might define a packet as a sequence of flow-control units (*flits*) which are sent one physical unit (*phit*) per cycle:

- A flit is a fixed integer (N) multiple of the size of a phit.
- A phit is the fixed number of data bits that can be transmitted in a single cycle on the link.
- The link is busy (B) when it is transferring a phit, that is, part of a flit or packet.

To calculate Utilization for such an interface, the number of phits ($B = \#\{\text{phit}\}$) or flits ($B = N \times \#\{\text{flit}\}$) must be counted, along with the total number of cycles (T).

Latency

Many processes are very latency sensitive. For example, a PE cannot make forward progress once all instructions are stalled waiting on long latency operations.

Latency is also referred to as *Response time*, W.

Statistics such as the maximum latency or distribution of latencies for a population of operations can be used in evaulating the effect of long latency operations on performance. For example, a PMU might provide a mechanism to count every operation that took longer than a programmed minimum latency. By varying this minimum in steps, software can create a histogram for the distribution of latencies.

However, these can be hard to measure if there can be many simultaneously outstanding operations. Statistical sampling of operations can be used to reduce the number of simultaneously monitored operations.

Alternatively, Little's Law can be used to derive average Latency compartively cheaply. Average latency is less useful, as performance tends to be most affected by outliers. To calculate average Latency, the

total number of outstanding operations on each cycle can be divided by the number of operations. See Occupancy.

Bandwidth

Cumulative number of data bytes (or some other fixed unit) transferred per unit of time.

When calculating *useful* bandwidth, only *data* bytes should be counted. Overheads that might form part of a transfer, such as the address or access type information, should not be counted. Example of *other fixed units* that might be counted are full and partial cache-lines.

Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture [2] defines a standard bandwidth monitor. See 1.4 Memory System Resource Partitioning and Monitoring *(MPAM)*.

Throughput

Cumulative number of packets (or some other variable unit) transferred per unit of time. In a packetized system, packet throughput can be an important measure of system performance. Packets might vary in size, meaning a comparison of Throughput and Bandwidth can determine average packet size.

Effectiveness

For example, a cache is *effective* if accesses to the cache have low latency because they do not miss in the cache.

Counting events that indicate effectiveness or ineffectiveness (for example, a cache miss) might be more cost efficient than measuring actual Latency.

Errors

An *error* means any unexpected circumstance; or, at least, any circumstance that the designer and user should not *expect*. That is, any significant performance or power impacting should-be-rare events.

Completions

Average *Completion rate* (μ) can be calculated by counting the total number of completed *tasks* and dividing by the elapsed time. The definition of a *task* is specific to each measurement. Completion rate is sometimes referred to as the *output rate* or *departure rate*.

For example, for a stored-program PE, a *task* might be completing a program instruction to retirement (giving a completion rate of instructions-per-cycle or IPC), or a particular class of operation.

Bandwidth and Throughput are other forms of Completion rate, where the *task* is transferring a byte of data or a packet across an interface.

Arrivals

Arrival rate (λ) can be calculated by counting the total number of *arrivals* and dividing by the elapsed time.

If tasks always complete then the average Arrival rate is the same as the average Completion rate.

Efficiency

If tasks do not always complete, then the ratio of the Completion rate to the Arrival rate is the Efficiency.

For example, for a stored-program PE, the ratio of the number of instructions architecturally executed to the number speculatively issued.

Occupancy

Average occupancy (L) can be calculated by counting the total *occupancy* of tasks on each cycle, and dividing by the total number of cycles. The definition of a *task* is specific to each measurement.

If Arrival rate (λ) is also measured then Little's Law can then be used to calculate average Response time ($W = L \div \lambda$). That is, the average time each task spends in the system.

For example, for a load-store unit of a processor, a *task* might be a memory access, and the occupancy is the total number of outstanding accesses in the load-store queue on each cycle.

Cumulative duration (occupancy) in particular states
For example, in a configuration state, refresh state, processing with interrupts masked, and so on. That is, the monitor measures total occupancy in the state of the resource being monitored. Typically, this is either zero (not in the state) or one (in the state) on each cycle.

If entries into the state are also measured then average Occupancy in the state can be calculated.

Cumulative duration of stalls (occupancy in the stalled state)
Time (or resources) spent doing nothing. Stalls might be further split into stall because there are no tasks to perform (frontend stall, $L = 0$) and stall due to inability to complete a task (backend stall, $L \neq 0$). These might be further refined into significant reasons for stalls.

Utilization can be calculated from the frontend stall time ($t_{L=0}$). ($\rho = 1 - t_{L=0}/T$)

Statistically sampled events
For some events or characteristic it might be expensive to accurately monitor the event or characteristic. For example calculating average duration for a task can be achieved using only a pair of event counters. However, to determine the *maximum* (or minimum) duration for the task means measuring the duration of all tasks, which might be impractical if there can be many tasks operating in parallel.

In such situations, the monitor might be designed to *sample* a subset of the tasks and instead measure the maximum or minimum duration of only the sampled tasks. That way, the monitor unit only has to be able to measure the number of sampled tasks that can operate in parallel.

Some sampling techniques, such as systematic sampling, can either guarantee or at least make it unlikely that more than one sampled task operates in parallel.

When such a technique is used, it is important that the monitor describes the parameters used for sampling.

Software increment (SW_INCR)
Increment on writes to a software increment register.

Cycles (CYCLES)
Increment once every cycle. An implementation might also include a fixed cycle counter.

Counter overflows (CHAIN)
Allowing one counter to overflow into another counter can be an effective way to flexibly allocate counters if resources are constrained. For example if 16-bit or 32-bit counters are implemented.

Note

This architecture does not require an implementation to have standard event types. The events that an event counter counts might be fixed by the implementation.

Certain events might only be applicable to certain performance monitors. For example, software increment is only useful on a stored-program PE, but generally not useful on a bus monitor.

1.4 *Memory System Resource Partitioning and Monitoring* (MPAM)

MPAM is an Arm architecture extension that provides mechanisms for partitioning shared resources between software agents such as Virtual Machines (VMs). MPAM also includes standard interfaces for memory-mapped resource monitoring components.

1.4.1 *Overview* is taken from the introduction to the *Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture* [2].

1.4.1 Overview

Some shared-memory computer systems run multiple applications or multiple virtual machines (VMs) concurrently. Such systems might have one or more of the following needs:

- Control the performance effects of misbehaving software on the performance of other software.
- Bound the performance impact on some software by any other software.
- Minimize the performance impact of some software on other software.

These scenarios are common in enterprise networking and server systems. The MPAM extension addresses these scenarios with two approaches that work together, under software control, to apportion the performance-giving resources of the memory system. The apportionment can be used to align the division of memory-system performance between software, to match higher-level goals for dividing the performance of the system between software environments.

These approaches are:

- Memory-system resource partitioning.
- Memory-system resource usage monitoring.

The main motivation of the extension is to make data centers less expensive. The extension can increase server utilization, so that fewer servers are needed for a given level of service. Utilization can be increased by controlling how much impact the best-effort jobs have on the tail latency of responses by web-facing jobs.

The MPAM extension describes:

- A mechanism for attaching partition identifiers and a monitoring property, for executing software on an Arm processing element (PE).
- Propagation of a *Partition ID* (PARTID) and *Performance Monitoring Group* (PMG) through the memory system.
- A framework for *Memory System Component* (MSC) controls that partition one or more of the performance resources of the component.
- *Resource Instance Selectors* (RIS) to access control settings within an MSC for multiple resources of the same type.
- Extension of the framework for MSCs to have performance monitoring that is sensitive to a combination of PARTID and PMG.
- Some implementation-independent, memory-mapped interfaces to memory-system component controls for performance resource controls most likely to be deployed in systems.
- Some implementation-independent memory-mapped interfaces to memory-system component resource monitoring that would likely be needed to monitor the partitioning of memory-system resources.

Chapter 2
Specification

This section is *normative*.

2.1 Organization

R_{KZZTB} A *Performance Monitoring Unit* (PMU) contains one or monitors that count events or otherwise monitor characteristics of one or more components in a system.

R_{DRPFZ} Each monitor <n> comprises:

- A *monitor value* register, PMEVCNTR<n>.
- Optional *monitor configuration* registers, to control event selection and filtering:
  - PMEVTYPER<n>.
  - PMEVFILTR<n>.
  - PMEVFILT2R<n>.
- A monitor enable bit, PMCNTEN[n].

R_{LFSVL} Each monitor might define an optional *overflow condition*. If the monitor is an event counter, the optional overflow condition is unsigned overflow of the event counter.

I_{MHCBH} An activity monitor is an example of an event counter that does not implement an overflow condition. See 2.4 *Activity Monitor Units (AMUs)*.

R_{WRKBB} If a monitor <n> defines an overflow condition, the monitor also comprises:

- A monitor overflow flag, PMOVS[n].
- An interrupt enable bit, PMINTEN[n].

R_{YCLQB} If a monitor <n> does not define an overflow condition, PMOVS[n] and PMINTEN[n] are RES0.

I_{ZJTZQ} The names of registers in a functionally-compliant implementation might differ from those in this document. In particular:

- The mnemonic and name might include some reference to the component being monitored.
- The mnemonics and names in this document are appropriate only for monitors that implement counters.

R_{KRLSW} The PMU might include a Software Increment register, PMSWINC. This feature is deprecated and not recommended for new designs. The Software Increment register is not described in this document. See the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

R_{JFFVG} If the 64-bit programmers' model extension is not implemented, the PMU might include Counter Event Identifications registers, PMCEID<n>. This feature is deprecated and not recommended for new designs. The Counter Event Identification registers are not described in this document. See the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

R_{KBRSS} When any monitor defines an overflow condition, the PMU includes an overflow interrupt request signal.

R_{MRBCQ} Monitors might be split into monitor groups.

R_{VPSKV} The size of each monitor value is IMPLEMENTATION DEFINED, up-to 64 bits.

S_{JRSCZ} The size of the largest monitor value is discoverable through PMCFGR.SIZE.

U_{NNHQG} When event counters are implemented, Arm recommends that all event counters are the same size.

Exceptions can be made for *fixed-function* counters, such as cycle counters, as is the case in the Armv8-A Performance Monitors Extensions when FEAT_PMUv3p5 is not implemented, and in the Armv8-M Performance Monitoring Extension.

U_{DPSDJ} When event counters are implemented, the choice of counter size is usually determined by the periodicity and impact of an event counter unsigned overflow:

- The expected period between overflows is $\frac{2^{SZ}}{f \times E(N)}$, where SZ is the size of the counter, $E(N)$ is the expected average number of events per counting cycle, and f is the counting frequency.

  For example, if SZ is 32, $E(N)$ is 0.1 and f is 2GHz, then the expected period between overflows is 21.4 seconds. The same example with a 48-bit counter has an expected period between overflows of just over 16 days, whereas a 64-bit counter has an expected period between overflows of almost 3,000 years.

- The impact of a counter overflow depends on the intended usage models for the counters.

  For example, if an overflow generates a processor interrupt and handled by software incrementing a soft overflow counter then resetting the overflow flag, then the impact is manageable if the overflow is infrequent, likely to be serviced before the next overflow, and the software complexity and maintenance cost is acceptable.

  However, if the counter is free-running and overflows are never serviced by software, the impact of overflow can be very large. Such a system might require that the counter never overflows.

R_{BNQPR} The alignment of monitor value registers in the programmers' model depends on the implemented features:

- If the 64-bit programmers' model extension is not implemented and the largest monitor value register is 32 bits or smaller, all monitor value registers are at word-aligned addresses.
- Otherwise, all monitor value registers are at doubleword-aligned addresses.

R_{HKCFV} The number of monitors is IMPLEMENTATION DEFINED and is discoverable through PMCFGR.N. If monitor groups, the 64-bit programmers' model extension, and the snapshot extension are not implemented:

- If the largest monitor value register is 32 bits or smaller, up-to 256 monitors can be implemented.
- Otherwise, up-to 128 monitors can be implemented.

I_{XDDYT} If any of monitor groups, the 64-bit programmers' model extension, or the snapshot extension are implemented, the maximum number of counters that can be implemented might be fewer than described by R_{HKCFV}. See the descriptions of the extensions for more information.

R_{GZMZD} When the 64-bit programmers' model extension is not implemented, each of the registers PMOVS, PMCNTEN, and PMINTEN are programmed through pairs of set/clear registers:

- PMOVSSET<m> and PMOVSCLR<m>.
- PMCNTENSET<m> and PMCNTENCLR<m>.
- PMINTENSET<m> and PMINTENCLR<m>.

When the register is accessed as a 32-bit register, each bit [q] in these registers corresponds to a status or control bit for monitor <n>, where $n = q + (32 \times m)$. For each register pair PM{*fn*}SET and PM{*fn*}CLR, where {*fn*} is one of OVS, CNTEN, or INTEN:

- A write of `0b1` to PM{*fn*}SET<m>[q] sets the PM{*fn*}[$q + (32 \times m)$] bit to `0b1`.
- A write of `0b1` to PM{*fn*}CLR<m>[q] clears the PM{*fn*}<m>[$q + (32 \times m)$] bit to `0b0`.
- A write of `0b0` to PM{*fn*}SET<m>[q] or PM{*fn*}CLR<m>[q] has no effect.
- A read of PM{*fn*}SET<m>[q] or PM{*fn*}CLR<m>[q] returns the current value of PM{*fn*}[$q + (32 \times m)$].

Note

Due to technical limitations of this manual, the set/clear registers are described in Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented* as:

- PMOVSSET<n> and PMOVSCLR<n>.
- PMCNTENSET<n> and PMCNTENCLR<n>.
- PMINTENSET<n> and PMINTENCLR<n>.

Do not confuse the <n> index used for these registers with the <n> index used for monitor value and configuration registers.

RCVRCT When the 64-bit programmers' model extension is implemented, the direct access to enables and overflows extension is also implemented. Each of the registers PMOVS, PMCNTEN, and PMINTEN can be programmed directly or through pairs of set/clear registers:

- PMOVS, PMOVSSET, and PMOVSCLR.
- PMCNTEN, PMCNTENSET, and PMCNTENCLR.
- PMINTEN, PMINTENSET, and PMINTENCLR.

When the 64-bit programmers' model extension is implemented, these register names do not have a number suffix.

Each bit [n] in these registers corresponds to a status or control bit for monitor <n>. For each group of registers PM{*fn*}, PM{*fn*}SET, and PM{*fn*}CLR, where {*fn*} is one of OVS, CNTEN, or INTEN:

- A write to PM{*fn*}[n] updates the PM{*fn*}[n] bit.
- A write of `0b1` to PM{*fn*}SET[n] sets the PM{*fn*}[n] bit to `0b1`.
- A write of `0b1` to PM{*fn*}CLR[n] clears the PM{*fn*}[n] bit to `0b0`.
- A write of `0b0` to PM{*fn*}SET[n] or PM{*fn*}CLR[n] has no effect.
- A read of PM{*fn*}[n], PM{*fn*}SET[n], or PM{*fn*}CLR[n] returns the current value of PM{*fn*}[n].

SYDRVT The PMU is described to software by the following registers:

- The PMCFGR register describes the capabilities of the PMU to software.
- The following registers provide a unique combination of a part number identifier, revision, and designer of the PMU:
  - The PMIIDR register. This register is optional and recommended if the 64-bit programmers' model extension is not implemented. This register is required if the 64-bit programmers' model extension is implemented.
  - CoreSight PMCIDR<n> and PMPIDR<n> registers. These registers are optional.

  Arm recommends that at least one of these identification mechanisms is implemented.
- The optional PMDEVARCH register describes when a PMU follows an architectural programmers' model.

  This might be the generic CoreSight PMU architecture defined by this manual, or another architecture such as an Armv8-A PE PMU. This manual defines architecture identifiers for the generic CoreSight PMU architecture. Other architecture values are defined by other manuals.
- The optional PMDEVAFF register describes when a PMU has an *affinity* with a single PE, or a group of PEs in the system.

  Each PE has a unique value that identifies it in the system. MPIDR_EL1 in the PE and PMDEVAFF in the PMU contain this value. PMDEVAFF might contain a value that matches a group of PEs.
- The optional PMDEVID register describes additional implementation-specific features of the PMU. This register is IMPLEMENTATION DEFINED and is interpreted in the context of the part number or architecture of the PMU.

Other characteristics of the PMU are discovered by software by other means, such as by using the part number as index to a configuration database.

IPFGYY The PMU might include a CoreSight Lock Access Register, although this is deprecated and not recommended for new designs. The CoreSight Lock Access Register mechanism is not described in this document. See the *ARM CoreSight Architecture Specification* [4].

I_{MFDCF} The PMU might have other interfaces to the registers which are not considered by this architecture. For example, an Armv8-A PE has a System register interface to the PMU registers and a recommended memory-mapped interface that is compatiable with this architecture.

These interfaces access the same performance monitor counters, but might have some differences in access permissions or other behaviors. For example, the Armv8-A Performance Monitors Extension can be configured to restrict the number of event counters that are accessible to a Guest operating system.

These interfaces are outside the scope of this architecture.

I_{GFGGZ} The PMU might have other deviations from this structure. For example, an Armv8-A PE has:

- Controls to partition the counters between those used by an Operating System and those used by a Hypervisor.
- Controls to configure overflow from either the bottom 32 bits or full 64 bits of the counter.

See also:

- *Arm® Architecture Reference Manual, for A-profile architecture* [3].
- *Armv8-M Architecture Reference Manual* [5].
- 2.5 *Accessing registers*.
- Chapter 3 *Extensions*.
- Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented*.
- Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented*.

2.2 Operation

R_{WXSQT} The PMU has a *RUN* state and a *STOP* state. Figure 2.1 shows this.

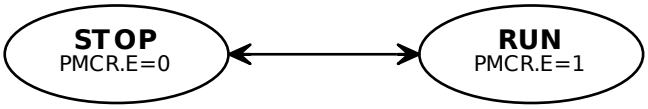


Figure 2.1: PMU state-machine

R_{HLTDF} The PMU operating state is determined by the PMCR.E control bit.

PMCR.E	State
0b0	STOP
0b1	RUN

I_{KNMNQ} The freeze-on-overflow extension describes a third state, WAIT.

I_{CZDHC} An AMU does not implement the STOP state.

S_{FJDNY} Software configures the monitors using the monitor configuration registers, PMEVTYPER<n>, PMEVFILTR<n>, and PMEVFILT2R<n>. If these registers are programmable, software must program the monitor configuration registers for each monitor before enabling that monitor. See R_{QLMFG} and R_{HSHSJ}.

I_{SDBSN} The behavior of accesses to PMU registers when the PMU is powered off or in a low-power retention state is IMPLEMENTATION DEFINED.

See also:

- 2.3 *Accuracy*.
- 2.6 *Security*.
- Chapter 3 *Extensions*.

2.2.1 Event counting

This section applies for monitors that are event counters.

R_{LXYWS} For a given event counter, *n*, the event counter increments by an IMPLEMENTATION DEFINED amount each time all of the following occur:

- The PMU is in the RUN state.
- The event counter is enabled by PMCNTEN[n].
- Counting is not prohibited. See 2.6 *Security*.
- The event conditions specified by the monitor configuration registers occur.

R_{TGVMJ} Increments of event counters by the PMU are atomic operations on the event counter.

R_{XKQWT} Event counters are unsigned integer counters.

R_{QLMFG} The event conditions specified by the monitor configuration registers configure what the event counter counts, including:

- The event counted by the event counter.

- Any filtering of the event, for example, on the context of the component being monitored. See also 3.7 *Reusable event filter definitions*.

Note

Depending on the implementation of the PMU, one or more of the monitor configuration registers might be RES0, meaning software does not need to program the register before enabling the event counter.

R_{VPCMF} For a given event counter, *n*, if the event counter increment generates unsigned overflow of the event counter:

- The overflow status flag PMOVS[n] is set to `0b1`, if the optional status flag is implemented.
- The event counter wraps through zero.

I_{SCFBG} If the PMU does not implement the freeze-on-overflow extension or *freeze-on-overflow* is not enabled, the event counter continues counting events. Counting continues as long as the event counter is enabled, regardless of any overflows.

I_{GKWYH} Unsigned overflow of an event counter might generate a countable event. See 3.5 *Counter chaining extension*.

S_{RBDBW} Software can reset all event counters to zero in a single operation, by writing `0b1` to PMCR.P.

2.2.2 State or event monitoring

This section applies for monitors that are not event counters.

R_{XWGCN} For a given monitor, *n*, the monitor monitors the IMPLEMENTATION DEFINED state or characteristic of the monitored component while all of the following are true:

- The PMU is in the RUN state.
- The monitor is enabled by PMCNTEN[n].
- Monitoring is not prohibited. See 2.6 *Security*.

R_{MBSSC} The conditions under which the monitored state or characteristic are updated, including the frequency of any updates, are IMPLEMENTATION DEFINED.

R_{NBWKD} Updates of the monitor by the PMU are atomic operations on the monitor.

R_{HSHSJ} The monitoring conditions specified by the monitor configuration registers configure what the monitor monitors, including:

- The state or events monitored by the monitor.
- Any filtering of the state or events, for example, on the context of the component being monitored. See also 3.7 *Reusable event filter definitions*.

I_{WNNQM} Depending on the implementation of the PMU, one or more of the monitor configuration registers for a monitor might be RES0, meaning software does not need to program the register before enabling the monitor.

R_{ZFGGW} The monitor value register for a monitor is either read-only or read/write. If the monitor value register is read/write, the effect on the monitor of writing a value to the register is IMPLEMENTATION DEFINED.

R_{VHWBS} For a given monitor, *n*, the monitor might define an overflow condition. When the monitor enters the overflow condition, the overflow status flag PMOVS[n] is set to `0b1`.

I_{LPGWN} If the PMU does not implement the freeze-on-overflow extension or *freeze-on-overflow* is not enabled, the monitor continues operating after recording the overflow condition.

R_{GHXCP} When software writes `0b1` to PMCR.P:

- If a monitor has a defined reset value or state, then the monitor is reset to its reset value or state.

- Otherwise, the monitor ignores the write to PMCR.P.

2.2.3 Mapping controls and fixed-function monitors

R_{YSQVN} The mapping of the monitor configuration register contents to controls is IMPLEMENTATION DEFINED, and might differ between monitors.

I_{XNQJD} R_{YSQVN} means that monitors might be dynamically programmable, but also that PMUs can have fixed monitors with IMPLEMENTATION DEFINED configurations.

U_{KDGTQ} It is preferred, but not required, that programmable monitors are homogeneous. That is, all monitors can be configured in the same way. However, for systems with very large numbers of events this might not be plausible, so this restriction is not enforced in the architecture.

As a result it is not required that there is a single definition for all event select registers. However, this is also strongly recommended. A part or all of the event select register might read-as a fixed, non-zero value.

2.2.4 Interrupt signaling

R_{MDCYL} If the PMU implements an overflow interrupt then it is asserted when all of the following are true for any given monitor, *n*:

- The overflow status flag PMOVS[n] is `0b1`.
- The interrupt enable bit PMINTEN[n] is `0b1`.
- The PMU is in the RUN or WAIT state.

S_{VFGSS} If software programs an event counter with a negative number then the PMU starts asserting the overflow interrupt request when the event counter wraps through zero. That is, after minus that number of events have been counted.

Software is responsible for deasserting the overflow interrupt request. For example, by clearing the overflow status flags to `0b0`.

I_{QQFDG} One overflow interrupt request is implemented for each PMU.

R_{RSZKZ} The mechanism for signaling overflow interrupt requests is IMPLEMENTATION DEFINED.

R_{FDLKQ} A PMU might include the PMIRQCR0, PMIRQCR1, and PMIRQCR2 registers to configure a message-signaled interrupt request.

When a message-signaled interrupt is implemented, when the PMU asserts the overflow interrupt request, the PMU writes the value specified in the PMIRQCR1 register to the address location specified by the PMIRQCR0 and PMIRQCR2 registers.

The status of the message is indicated in PMIRQSR. If this standard form of message-signaled interrupts are implemented then PMCFGR.MSI reads as `0b1`.

I_{VBWMC} When an interrupt request signal is implemented, Arm strongly recommends that:

- If the PMU is accessible to an application processor in the system, then the interrupt request signal is connected to an interrupt controller that can route the interrupt to the application processor.
- If the PMU has close affinity to a single PE, then the interrupt request signal is configured as a *Private Peripheral Interrupt* (PPI) for that PE.

2.3 Accuracy

I_{SDBRN} The PMU provides approximately accurate performance information.

R_{YTRMH} To keep the implementation and validation cost low, a reasonable degree of inaccuracy in the monitored values is often acceptable. The permitted degree of inaccuracy is IMPLEMENTATION DEFINED.

I_{DYQXP} There is no exact definition of *reasonable degree of inaccuracy*, but the following guidelines are recommended:

- Under normal operating conditions, the monitors must present an accurate value of the monitored characteristic.
- In exceptional circumstances, such as changes in Security state or other boundary conditions, it is acceptable for the value to be inaccurate.
- Under very unusual non-repeating pathological cases values can be inaccurate. These cases are likely to occur as a result of asynchronous exceptions, such as interrupts, where the chance of a systematic error in the value is vanishingly unlikely.

An implementation must not introduce inaccuracies that can be triggered systematically by normal pieces of code that are running. For example, dropping a branch count in a loop due to the structure of the loop gives a systematic error that makes the count of branch behavior very inaccurate, and this is not reasonable. However, the dropping of a single branch count as the result of a rare interaction with an interrupt is acceptable.

The permitted inaccuracy limits the possible uses of the PMU. In particular, the point in an operation pipeline where the monitor is updated is not defined relative to the point where a read of the monitor is made. This means that pipelining effects can cause some imprecision. An implementation must document any particular scenarios where significant inaccuracies are expected.

2.4 Activity Monitor Units (AMUs)

D_{RTBQV} An Activity Monitor Unit *(AMU)* is a PMU that includes only activity monitors.

D_{DYHKN} An *activity monitor* is a monitor with the following properties:

- Once initialized, the monitor is not reprogrammed again, unless reset by a hardware reset. If the monitor is an event counter, this means the counter is *free-running*.
- The monitor does not implement an overflow condition. This means the PMOVS and PMINTEN registers, and the overflow interrupt request signal are not implemented.
- The monitor is often *fixed-function*, although this is not required.

R_{HXNVJ} When the activity monitor is enabled, writes to AMU monitor value or monitor configuration registers are not allowed and might be ignored or UNPREDICTABLE.

S_{LVLGJ} System firmware programs an AMU at power-on of the system component to a single configuration. The configuration then does not change and the activity monitors are free-running while the component is powered.

This model guarantees a consistent view, supporting multiple accessing agents, for system management and monitoring.

The system firmware is responsible for providing this guarantee. For instance, if an activity monitor configuration register allows writes before the AMU is enabled, then part of this guarantee is that the event is configured once by system firmware at power-on, and not changed during system runtime.

Allowing system firmware to configure an activity monitor configuration register in this way provides system firmware the flexibility to make final event choices, where committing to a full counter per-event is considered too high a cost.

U_{ZSFGG} The behavior of accesses to AMU registers when the AMU is powered off or in a low-power retention state is IMPLEMENTATION DEFINED, as defined by I_{SDBSN}.

Arm recommends that AMU registers are RAZ/WI when the AMU is powered off or in a low-power retention state.

R_{QMNHC} An AMU optionally supports the following PMU extensions:

- The halt-on-debug extension.
- The snapshot extension.
- Monitor groups.
- The direct access to enables and overflows extension and 64-bit programmers' model extension.
- The observability and access control extension.

An AMU might also include the trace generation extension and export extension, although these are not expected in most implementations.

R_{BGHGF} An AMU does not support the following PMU features and extensions:

- The overflow interrupt request signal.
- The freeze-on-overflow extension.
- The fixed-function cycle counter extension. Although an AMU does typically include a fixed-function cycle counter, this is implemented as one of the general event counters.
- Counter chaining.
- The threshold extension and edge-detect extension.
- The MPAM filtering extension.
- The dual-page extension.

I_{PWNVC} AMU registers typically have names starting *AM*. The following table maps PMU register names to AMU registers. Other PMU registers may be implemented.

PMU name	AMU name	Comments
PMCFGR	AMCFGR	
PMCGCR<n>	AMCGCR<n>	If AMCFGR.NCG > 0.
PMCNTENCLR<n>	AMCNTENCLR<n>	
PMCNTENSET<n>	AMCNTENSET<n>	
PMCR	AMCR	
PMEVCNTR<n>	AMEVCNTR<n>	
PMEVFILTR<n>	AMEVFILTR<n>	Optional.
PMEVFILT2R<n>	AMEVFILT2R<n>	Optional.
PMEVTYPER<n>	AMEVTYPER<n>	May be read-only.
PMIIDR	AMIIDR	Required for AMU.

R_{QZBSN} Figure 2.2 shows the bit assignments for the AMCFGR register. All other fields from PMCFGR are RAZ.

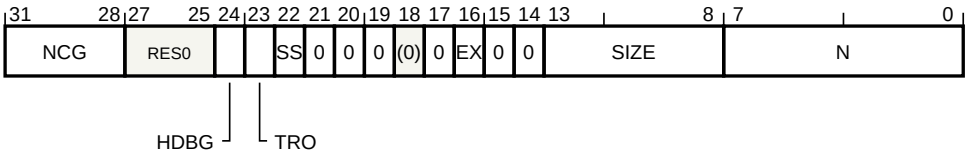


Figure 2.2: AMCFGR bit assignments

When the AMU implements the 64-bit programmers' model extension, AMCFGR is a 64-bit register.

R_{DPGNP} The AMU control register, AMCR, does not implement the following control bits present in PMCR:

- AMCR.E is RAZ/WI. The AMU is always in the RUN state. AMU activity monitors are controlled by the AMCNTEN<n> registers, which reset to zero
- AMCR.P ignores writes.

Figure 2.3 shows the bit assignments for the AMCR register, showing the optional control fields for the trace generation extension, halt-on-debug extension, and export extension.

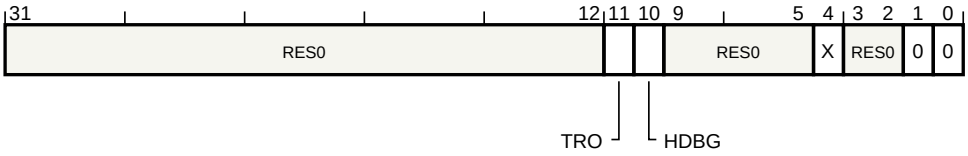


Figure 2.3: AMCR bit assignments

When the AMU implements the 64-bit programmers' model extension, AMCR is a 64-bit register.

2.5 Accessing registers

R_{WVQDZ} It is IMPLEMENTATION DEFINED whether the monitor value registers, PMEVCNTR<n>, are writable through the register interface.

R_{RNPJY} It is IMPLEMENTATION DEFINED whether the monitor configuration registers, PMEVTYPER<n>, PMEVFILTR<n>, and PMEVFILT2R<n>, are writable through the register interface.

R_{CGWBM} If monitors are writable, then it is IMPLEMENTATION DEFINED whether all monitor value and monitor configuration registers can be written to in states other than the STOP state.

R_{YNJLW} If monitors are writable and the PMU does not allow any monitor value or monitor configuration registers be written to in states other than the STOP state then PMCFGR.NA reads as `0b1`. Otherwise, PMCFGR.NA reads as `0b0`.

R_{GBTNB} If monitors are writable and PMCFGR.NA is `0b0`, then it is further IMPLEMENTATION DEFINED whether the PMU allows writes to individual monitor value and monitor configuration registers in states other than the STOP state for a monitor that is enabled.

R_{CLXGJ} A write to a monitor value or monitor configuration register that is not allowed by R_{CGWBM} or R_{GBTNB} might do any of:

- Be ignored by the PMU.
- Cause the monitor value to become incorrect.

This rule does not apply to PMCR and the snapshot extension registers. are ignored.

R_{YHTDZ} A write to a monitor value and monitor configuration register that is allowed by R_{CGWBM} and R_{GBTNB} in a state other than the STOP state takes effect in finite time. That is:

- Before the write takes effect, indirect reads of the register by the PMU made as part of the operation of the PMU yield the old value.
- After the write takes effect, indirect reads of the register by the PMU made as part of the operation of the PMU yield the new value.

R_{JQPHY} A write to a monitor value and monitor configuration register in the STOP state takes effect before leaving the STOP state.

R_{QBXTC} Reads of monitor value and monitor configuration registers through the register interface return the last value written to the registers. This is either the last value written through the register interface, or a value written by the PMU after the write through the register interface took effect.

I_{PYSHB} A write to the register through the register interface will not be overwritten by a concurrent hardware update of the monitor value register, for example, when incrementing an event counter. See R_{TGVMJ} and R_{NBWKD}.

A following read of the monitor value register through the register interface will return either the last value written through the register interface, or the last value written by the PMU. If the value returned is the value written by the PMU, then this write by the PMU will have occurred after the write through the register interface.

That is, if the monitor is an event counter, a read of the event counter through the register interface will return either the value written to the event counter, or the value written to the event counter and subsequently incremented by the PMU. The read of the event counter does not return either the old value or a value incremented from the old value.

R_{PVPYD} An implementation might define additional constraints on accessing PMU registers.

I_{DCPZN} For example, the Armv8-A Performance Monitors Extension defines that external accesses to the PMU are not permitted when the OS Lock is locked. See the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

R_{ZWQLG} A memory-mapped PMU must implement the supported access sizes defined by the section *Supported access sizes* in the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

R_{ZQQKP} When the 3.13 *64-bit programmers' model extension* is not implemented, permitted word-aligned 32-bit accesses to either half of a 64-bit register that is mapped to a doubleword-aligned pair of adjacent 32-bit locations are supported.

R_{ZNTBP} When the 3.13 *64-bit programmers' model extension* is not implemented, it is IMPLEMENTATION DEFINED and recommended that doubleword-aligned 64-bit accesses are supported to the following pairs of 32-bit registers, where *n* is even and both registers are implemented:

- PMCNTENSET<n> and PMCNTENSET<n+1>.
- PMCNTENCLR<n> and PMCNTENCLR<n+1>.
- PMINTENSET<n> and PMINTENSET<n+1>.
- PMINTENCLR<n> and PMINTENCLR<n+1>.
- PMOVSSET<n> and PMOVSSET<n+1>.
- PMOVSCLR<n> and PMOVSCLR<n+1>.

I_{MSFBY} R_{ZNTBP} means that it is IMPLEMENTATION DEFINED and recommended that the PMU treats each PM{*fn*}{SET|CLR}<n> and PM{*fn*}{SET|CLR}<n+1> register pair as a 64-bit register. This recommendation includes the case where the <n+1> register is reserved because fewer than $32 \times (n + 1)$ monitors are implemented.

R_{DDZMD} When the 3.13 *64-bit programmers' model extension* is not implemented, it is IMPLEMENTATION DEFINED and recommended that permitted doubleword-aligned 64-bit accesses to 64-bit registers and supported pairs of 32-bit registers are single-copy atomic at doubleword granularity.

I_{WHSHV} If doubleword-aligned 64-bit accesses are only single-copy atomic at word granularity, then the system might generate a pair of 32-bit accesses from a 64-bit access. The order in which the two halves are accessed is not specified.

R_{ZWZGC} A memory-mapped PMU must implement the memory-mapped component synchronization rules defined by the section *Synchronization of memory-mapped registers* in the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

R_{KRVRQ} A memory-mapped PMU must implement the access requirements for reserved and unallocated registers defined by the section *Access requirements for reserved and unallocated registers* in the *Arm® Architecture Reference Manual, for A-profile architecture* [3].

See also:

- *Arm® Architecture Reference Manual, for A-profile architecture* [3].
- 2.6 *Security*.
- 3.13 *64-bit programmers' model extension*.
- Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented*.
- Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented*.

2.6 Security

Note

A PMU produces performance data about the behavior of the system being monitored. It might be possible for users to deduce information about the system from this performance data. This might be done directly or indirectly using the PMU as a side-channel.

The PMU architecture does not, by itself, protect against exposing such information. Arm recommends that, when implementing performance monitoring features, designers assess the types of information that might be exposed by a PMU and implement any necessary safeguards to avoid exposure of any information that the designer deems to be secure or confidential. This assessment should be based on the security requirements of the system being monitored and to whom the performance data is being exposed.

This section provides recommendations for a PMU that might monitor systems processing secure or confidential information. It does not define which information is *secure* or *confidential*, nor does it define mechanisms to control access to the performance data. These details are implementation-specific. Implementations of this architecture should extend and adapt these rules as required.

R_{JGBVD} A PMU must not count events or expose characteristics when counting that event or monitoring that characteristic is prohibited.

R_{VHTGC} When an event or characteristic of an agent being monitored is attributable to an *operating state* of the agent:

- Counting the event or monitoring the characteristic is *allowed* when non-invasive debug of the operating state is allowed.
- Counting the event or monitoring the characteristic is *prohibited* when non-invasive debug of the operating state is prohibited.

I_{PPGYJ} The *Arm® Architecture Reference Manual, for A-profile architecture* [3] defines the following Security operating states for a PE, corresponding to physical address spaces (PAS):

- Non-secure operating state and the Non-secure physical address space.
- Secure operating state and the Secure physical address space.
- Realm operating state and the Realm physical address space, when FEAT_RME is implemented.
- Root operating state and the Root physical address space, when FEAT_RME is implemented.

R_{WRKQV} A PMU might include IMPLEMENTATION DEFINED *authentication controls* for whether non-invasive debug is allowed or prohibited. These controls might further determine whether non-invasive debug is allowed or prohibited for a subset of the operating states or operations of the agent being monitored.

I_{JVJPQ} A PMU also includes controls that *enable* or *disable* the monitors. Disabling a monitor takes precedence over the authentication controls. Non-invasive debug authentication only controls whether monitoring is allowed, it does not control access to the performance monitor registers, although this might be an additional feature of an implementation.

I_{GRNVM} Example implementations of authentication controls are:

1. An authentication interface, where asserting a signal to the component means a class of operation is allowed and de-asserting the signal means the class of operation is prohibited.
2. A control register or registers that are not accessible to untrusted software. For example, a Secure register within the component that cannot be accessed by Non-secure agents and must be programmed by a Secure agent to control whether monitoring of Secure operations by the PMU is allowed or prohibited.
3. A fixed configuration. For example, always treating Secure operations as prohibited operations.
4. A combination of 1, 2, or 3.

R_{BLDJB} The observability and access control extension defines standard control registers that a PMU can implement to provide authentication controls.

R_{NHTQL} When a component containing a PMU implements a Root operating state or processes Root operations, a PMU that is accessible to Secure, Realm, or Non-secure software treats events or characteristics attributable to Root operation of the agent being monitored as prohibited unless enabled by the IMPLEMENTATION DEFINED authentication controls.

R_{VSQRY} When a component containing a PMU implements a separate Realm operating state or processes Realm operations, a PMU that is accessible to Secure or Non-secure software treats events or characteristics attributable to Realm operation of the agent being monitored as prohibited unless enabled by the IMPLEMENTATION DEFINED authentication controls.

R_{DBXDJ} When a component containing a PMU implements a Secure operating state or processes Secure operations, a PMU that is accessible to Non-secure or Realm software treats events or characteristics attributable to Secure operation of the agent being monitored as prohibited unless enabled by the IMPLEMENTATION DEFINED authentication controls.

I_{GMSMS} Access to a PMU might depend on assignment of a PMU to a Physical Address Space (PAS).

For example, a PMU assigned to the Non-secure PAS can be accessed from any state, but a PMU assigned to the Secure PAS can only be accessed from Secure or Root state.

U_{CJBXN} R_{NHTQL}, R_{VSQRY}, and R_{DBXDJ} might be a property of the implementation, or might be a requirement on device firmware to manage the IMPLEMENTATION DEFINED authentication controls appropriately. In particular, when assignment is controlled by device firmware, the firmware must also manage the authentication controls for the PMU to ensure these rules are met.

I_{WZSYV} When the component containing the PMU implements a Secure operating state or processes Secure operations, the system might include additional IMPLEMENTATION DEFINED authentication controls that further restrict access to the PMU to only Secure state. This is in addition to any access control provided by an MMU or System MMU restricting access to the memory-mapped registers of the component.

I_{THBHR} When the component containing the PMU implements a Root operating state or processes Root operations, the system might include additional IMPLEMENTATION DEFINED authentication controls that further restrict access to the PMU to only Root state. This is in addition to any access control provided by an MMU or System MMU restricting access to the memory-mapped registers of the component.

R_{RYNLT} When a component containing a PMU implements a Secure operating state or processes Secure operations, any IMPLEMENTATION DEFINED authentication controls for Secure operation are not configurable by Non-secure or Realm software.

R_{BZLYW} When a component containing a PMU implements a Realm operating state or processes Realm operations, any IMPLEMENTATION DEFINED authentication controls for Realm operation are not configurable by Secure or Non-secure software.

R_{RRLNM} When a component containing a PMU implements a Root operating state or processes Root operations, any IMPLEMENTATION DEFINED authentication controls for Root operation are not configurable by Secure, Non-secure, or Realm software.

I_{WTVDK} The CoreSight architecture describes the **NIDEN**, **DBGEN**, **SPIDEN**, and **SPNIDEN** authentication interface signals. These signals control:

- Invasive and non-invasive debug authentication.
- Secure and Non-secure debug authentication.

I_{DWKKS} The Realm Management Extension describes the **RLPIDEN** and **RTPIDEN** authentication interface signals. These signals control Root and Realm debug authentication.

I_{DNMWG} In the Armv8-A Performance Monitors Extension:

- The Non-secure debug authentication signals (**NIDEN** and **DBGEN**) are ignored by the PMU when determining whether to count events attributable to Non-secure state.
- The MDCR_EL3.SPME control and, if FEAT_PMUv3p7 is implemented, the MDCR_EL3.MPMX control prohibit counting of events attributable to EL3 and/or Secure state.
- If FEAT_PMUv3p1 is implemented, the MDCR_EL2.HPMD prohibits counting of events attributable to EL2 by some event counters.
- If FEAT_Debugv8p2 is not implemented, when asserted, the Secure debug authentication signals (**SPNIDEN** and **SPIDEN**) override the MDCR_EL3.SPME and MDCR_EL2.HPMD controls.
- If FEAT_Debugv8p2 is implemented, the Secure debug authentication signals are ignored by the PMU when determining whether to count events attributable to Secure state or EL2. If the Realm Management Extension, FEAT_RME, is also implemented, the Root and Realm debug authentication signals (**RLPIDEN** and **RTPIDEN**) are ignored by the PMU when determining whether to count events attributable to Root and Realm states.
- If FEAT_Debugv8p4 is implemented, the Non-invasive debug authentication signals (**NIDEN** and **SPNIDEN**) are not implemented.

This means that the authentication interface is ignored by the PMU when FEAT_Debugv8p2 is implemented, and the only authentication controls are those managed by privileged software.

ITZLYG Arm recommends that:

- A PMU that is used by external or other independent agents in the system, and requires authentication without the intervention of software, implements an authentication interface. For example, a PMU that can be used by an external debug agent and can monitor behavior from system reset.
- A PMU that is primarily used by software, particularly software running on application processors and executing in a rich operating system environment, implements software authentication controls and does not implement an authentication interface. For example, the Armv8-A Performance Monitors Extension.

See also:

- *Arm® Architecture Reference Manual, for A-profile architecture* [3].
- *ARM CoreSight Architecture Specification* [4].
- *Arm Realm Management Extension (RME) System Architecture* [6].
- 3.14 *Observability and access control extension*.

Chapter 3
Extensions

3.1 Freeze on overflow extension

R_{YTDDD} It is IMPLEMENTATION DEFINED whether a *Performance Monitoring Unit* (PMU) implements a control, *PMCR.FZO*, to disable (freeze) monitors when any monitor overflows.

R_{GYWBJ} If PMCR.FZO is implemented then PMCFGR.FZO reads as `0b1`.

I_{FSDCY} If PMCR.FZO is implemented then the PMU has RUN, STOP, and *WAIT* states. Figure 3.1 shows this.

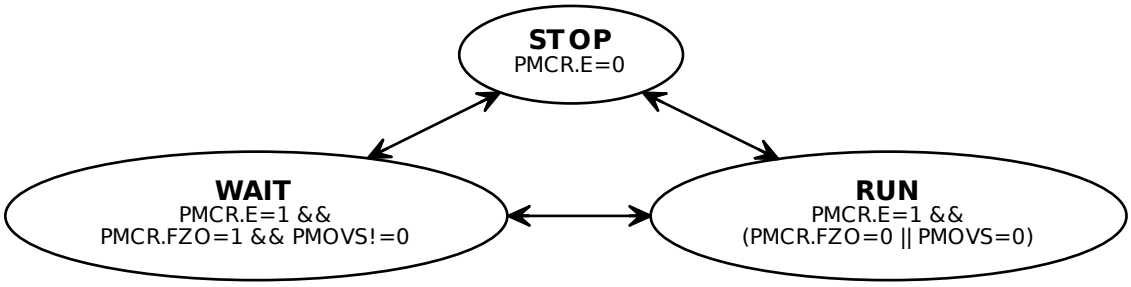


Figure 3.1: Freeze-on-overflow state machine

R_{MLXVL} The PMU operating state is determined by the PMCR.E and PMCR.FZO bits and applicable monitor overflow flags, PMOVS.

PMCR.E	PMCR.FZO	PMOVS	State
0b0	X	X	STOP
0b1	0b0	X	RUN
0b1	0b1	zero	RUN
0b1	0b1	nonzero	WAIT

The applicable monitor overflow flags are all monitor overflow flags other than as described by R_{JKHCM} and R_{VPDGQ}. This includes overflow flags for disabled monitors.

I_{DHXMS} State changes are not precise, meaning that some events might be counted after overflow, or might not be counted after software clears the overflow flags. However, state changes must occur in finite time. See 2.3 *Accuracy*.

R_{JKHCM} If a PMU implements the freeze-on-overflow extension and counter chaining, it is IMPLEMENTATION DEFINED which of the following applies:

- The overflow flag for an even-numbered counter PMEVCNTR<n> is ignored by the freeze-on-overflow feature when the corresponding odd-numbered counter PMEVCNTR<n+1> is configured to count the CHAIN event.
- The overflow status for all counters are considered for the freeze-on-overflow feature.

S_{NXGXL} If the freeze-on-overflow feature of a PMU that implements counter chaining does not ignore the overflow status for an even-numbered counter when the corresponding odd-numbered counter is configured to count the CHAIN event, then on overflow of the even-numbered counter:

- The CHAIN event is generated for the odd-numbered counter.
- The PMU enters the WAIT state.

It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether the CHAIN event is counted before the PMU enters the WAIT state or after the PMU enters the WAIT state. This can occur only once, as the even-numbered counter now stops counting events, meaning there is no benefit from setting the CHAIN event.

R_{DPGBH} If a PMU implements the freeze-on-overflow extension and fixed-function cycle counter extension, the operation of the cycle counter in the WAIT state is IMPLEMENTATION DEFINED. That is, if the cycle counter is enabled and allowed to count and the PMU is in the WAIT state, the PMU does one of:

- The cycle counter will count cycles.
- The cycle counter will count cycles if PMCR.DP is 0b0 and not count cycles if PMCR.DP is 0b1.
- The cycle counter will not count cycles.

R_{VPDGQ} If a PMU implements the freeze-on-overflow extension and fixed-function cycle counter extension and the cycle counter can count cycles in the WAIT state, then the freeze-on-overflow feature ignores the overflow flag for the cycle counter.

See also:

- 3.3 *Fixed-function cycle counter extension*.
- 3.5 *Counter chaining extension*.

3.2 Halt-on-debug extension

I_{XXHPW} A PMU might have an affinity with a PE or other agent that has both a Non-debug operating state and a Debug operating state.

R_{SQJTW} If the PMU has affinity with an agent that includes a Debug operating state, the PMU behaves as one of the following and it is IMPLEMENTATION DEFINED which one:

- Events are counted in the Debug operating state.
- The PMU includes a control, *PMCR.HDBG*, that controls whether events are counted in the Debug operating state.
- Events are not counted in the Debug operating state.

This might differ for groups of events.

R_{XPPSF} If the PMU implements PMCR.HDBG then PMCFGR.HDBG reads as `0b1`.

3.3 Fixed-function cycle counter extension

I_{SZJRL} This feature is described for compatibility with the Arm architecture PMU. It is not recommended for other PMUs to implement a fixed-function cycle counter in this way.

An implementation can include other fixed-function monitors. If the cycle counter extension is not implemented, a PMU might still include a fixed-function cycle counter. The only difference is that the PMU does not implement the fixed-function cycle counter as monitor 31.

R_{KFQDK} An implementation might include a dedicated *cycle counter*, PMCCNTR. It counts unhalted clock cycles in the clock domain of the PMU.

R_{KFHHP} If the fixed-function cycle counter extension is implemented then PMCFGR.CC reads as `0b1`.

R_{HJZYK} For compatibility with the Arm architecture PMU, the cycle counter is always an alias for PMEVCNTR31.

R_{CMLXS} If the fixed-function cycle counter extension is implemented and the cycle counter is 32 bits or fewer, the implementation might include a prescaler for the cycle counter, such that it counts by one for every 64 cycles. If the prescaler is implemented then PMCFGR.CCD reads as `0b1`.

I_{JNMYM} If the fixed-function cycle counter extension is not implemented then monitor 31 (if implemented) is a regular monitor.

R_{VZKXS} If the fixed-function cycle counter extension is implemented and the agent being monitored has two operating states then the cycle counter:

- Counts when PMCR.DP is `0b0` or the agent being monitored is in the first state.
- Does not count when PMCR.DP is `0b1` and the agent being monitored is in the second state.

The definitions of the first and second states are IMPLEMENTATION DEFINED.

I_{BNDHS} In the Armv8-A Performance Monitors Extension, the first state is a state where counting events is allowed and the second state is a state where counting events is prohibited.

- Depending on the features of the PE and the values of the MDCR_EL3.SPME and MDCR_EL3.HPMD controls, the second state might include Secure state or EL2.
- If FEAT_PMUv3p5 is implemented, MDCR_EL3.SCCD disables the cycle counter in Secure state and MDCR_EL2.HCCD disables the cycle counter at EL2, regardless of the value of PMCR.DP.

I_{SVGVM} In the Armv8-M Performance Monitoring Extension, when the Security Extensions are implemented, the first state is Non-secure state and the second state is Secure state. This is irrespective of whether Secure non-invasive debug is allowed or prohibited.

See also:

- *Arm® Architecture Reference Manual, for A-profile architecture* [3].
- *Armv8-M Architecture Reference Manual* [5].
- 2.6 *Security*.
- 3.1 *Freeze on overflow extension*.

3.4 Monitor group extension

I_{GPQWD} An implementation might include up-to 16 *monitor groups*. Monitor groups are a mechanism for a PMU to partition monitors.

R_{KZNGG} If monitor groups are implemented, the number of monitor groups is IMPLEMENTATION DEFINED, and discoverable through PMCFGR.NCG.

R_{ZSFHF} If monitor groups are implemented then PMCFGR.NCG is nonzero.

R_{TFNPJ} The maximum number of monitors per group depends on the number of groups, the size of the largest implemented monitor, and whether 64-bit programmers' model extension is implemented.

The following table shows the maximum number of monitors per group for the following configurations:

- Configuration **A**, when 64-bit programmers' model extension is not implemented, and monitors are 32-bits or smaller.
- Configuration **B**, when 64-bit programmers' model extension is not implemented, and at least one monitor is larger-than 32 bits.
- Configuration **C**, when 64-bit programmers' model extension is implemented.

Number of monitor groups	A	B	C
2	32	32	32
3 or 4	32	32	16
Between 5 and 8	32	16	8
9 or more	16	8	4

I_{GNPZG} The first counter for monitor group m is PMEVCNTR<$m \times max$>, where max is defined by R_{TFNPJ}.

I_{HGRPD} For example, an implementation might define:

- PMCFGR.NCG reads as `0b0001`, indicating 2 monitor groups.
- PMCGCR0 reads as `0x00000604`, indicating:
  - Group 0 has four monitors: PMEVCNTR0 to PMEVCNTR3.
  - Group 1 has six monitors: PMEVCNTR32 to PMEVCNTR37.
- PMCFGR.N reads as `0x09`, indicating 10 monitors are implemented overall.

I_{JJQRM} The System MMU architecture implements a different model of monitor groups.

3.5 Counter chaining extension

I_{TCBXV} Support for larger event counters can be provided by *counter chaining*.

R_{YBGZK} If counter chaining is implemented, then an odd-numbered counter PMEVCNTR<n+1> configured to count the `CHAIN` event increments when the even-numbered counter PMEVCNTR<n> has an unsigned overflow on counting an event.

S_{PGJDB} For example where two 16-bit counters, PMEVCNTR0 and PMEVCNTR1 are provided, they can be chained to form a 32-bit counter by programming PMEVTYPER1 to count the `CHAIN` event.

R_{DJHLY} It is IMPLEMENTATION DEFINED whether the increment of the odd-numbered counter PMEVCNTR<n+1> occurs atomically with update of the even-numbered counter PMEVCNTR<n>.

If the increment is not atomic it must occur in limited finite time such that a read of PMEVCNTR<n+1> that is ordered-after a read of PMEVCNTR<n> must not observe the updated PMEVCNTR<n> without also observing the updated PMEVCNTR<n+1>. However, it is possible that the reads might observe the updated PMEVCNTR<n+1> without observing the updated PMEVCNTR<n>.

S_{QZDDJ} If the updates are not atomic, software must take care when reading the pair of counters.

See also:

- 3.1 *Freeze on overflow extension*.

3.6 Event counter threshold and edge detection extensions

I_{FVTLP} A *threshold condition* allows software to program an event counter such that the counter counts only when the amount that the counter would otherwise count by meets the threshold condition.

The *threshold extension* is a reusable programmers' model for providing a threshold condition. The extension defines the control fields used to program a threshold condition, but it is IMPLEMENTATION DEFINED where these fields appear in the monitor configuration registers.

The supported threshold conditions are *less-than*, *greater-than-or-equal-to*, *not-equals*, and *equal-to*. The amount the counter counts by is either one or the amount the event would have counted by.

I_{YCRKM} An *edge condition* allows software to program an event counter such that the counter counts only when the amount the counter would otherwise count by transistions between zero and nonzero.

The *edge-detect extension* is a reusable programmers' model for providing an edge condition. The extension defines the control fields used to program an edge condition, but it is IMPLEMENTATION DEFINED where these fields appear in the monitor configuration registers.

The supported edge conditions are *leading-edge*, *falling-edge*, or *either edge*.

R_{XJNGF} The event counter <n> increments by:

- V_E, if edge counting for event counter <n> is implemented and enabled.
- V_T, if threshold counting for event counter <n> is implemented and enabled.
- V_B, otherwise.

D_{PDZFT} V_B is the value the counter increments by when threshold counting is disabled and edge counting is disabled for the event counter <n>.

I_{GSNZJ} V_T is defined by R_{NHDQS}.

V_E is defined by R_{BKVJC} if the threshold extension is implemented, and R_{LWKFP} otherwise.

3.6.1 Threshold extension

When the threshold extension is implemented by an event counter <n>:

R_{WCJQQ}
- The event counter implements two control fields TC and TH. TC is a 3-bit field.

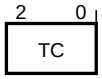


Figure 3.2: Threshold control

The size of the TH field is IMPLEMENTATION DEFINED.

R_{SWGMM}
- If TC is not `0b000` or TH is nonzero, *threshold counting* for event counter <n> is *enabled*.
- If TC is `0b000` and TH is zero, threshold counting for event counter <n> is *disabled*.

R_{DTCTP}
- The *threshold condition* is controlled by TC and TH. TC also controls the amount the counter increments by when the threshold condition is met.

R_{NHDQS}
- The value V_T is defined as follows:

$$V_T = \begin{cases} V_B, & \text{if TC[0] = 0b0 and the threshold condition matches.} \\ 1, & \text{if TC[0] = 0b1 and the threshold condition matches.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{threshold condition} = \begin{cases} V_B \neq \text{TH}, & \text{if TC[2:1] = 0b00 (not equal).} \\ V_B = \text{TH}, & \text{if TC[2:1] = 0b01 (equals).} \\ V_B \geq \text{TH}, & \text{if TC[2:1] = 0b10 (greater-than-or-equal).} \\ V_B < \text{TH}, & \text{if TC[2:1] = 0b11 (less-than).} \end{cases}$$

Note that V_T equals V_B when threshold counting for event counter <n> is disabled (TC = 0b000).

Table 3.3: Threshold condition

TC[2:0]	Description	Condition
0b000	Not-equal.	V_T is V_B if V_B is not equal to TH, and zero otherwise. If TH is zero, the threshold function is disabled.
0b001	Not-equal, count.	V_T is 1 if V_B is not equal to TH, and zero otherwise.
0b010	Equals.	V_T is V_B if V_B is equal to TH, and zero otherwise.
0b011	Equals, count.	V_T is 1 if V_B is equal to TH, and zero otherwise.
0b100	Greater-than-or-equal.	V_T is V_B if V_B is TH or greater, and zero otherwise.
0b101	Greater-than-or-equal, count.	V_T is 1 if V_B is TH or greater, and zero otherwise.
0b110	Less-than.	V_T is V_B if V_B is less than TH, and zero otherwise.
0b111	Less-than, count.	V_T is 1 if V_B is less than TH, and zero otherwise.

3.6.2 Edge-detect extension

3.6.2.1 Threshold extension implemented

When the edge-detect extension is implemented and the threshold extension is implemented by an event counter <n>:

R_YWVCK
- If the threshold extension is not implemented, the event counter implements a 1-bit control field, TE.

Figure 3.3: Edge control (thresold extension implemented)

R_WSQDY
- The *edge condition* is controlled by TE and TC.

D_LQGJX
- If TE is 0b1, edge counting for event counter <n> is enabled.
- If TE is 0b0, edge counting for event counter <n> is disabled.

R_LRPMJ
- The value V_C is defined as follows:

$$V_C = \begin{cases} 1, & \text{if the threshold condition matches.} \\ 0, & \text{otherwise.} \end{cases}$$

R_CTSXC

- V_P is the value V_C took on the previous cycle if counting the event was allowed on the previous cycle, and zero otherwise.

R_{BKVJC}

- The value V_E is defined as follows:

$$V_E = \begin{cases} 1, & \text{if } V_P \text{ and } V_C \text{ satisfy the edge condition.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{edge condition} = \begin{cases} V_P = 0 \neq V_C, & \text{if TC[0] = 0b0 (both edges).} \\ V_P = 0 \wedge V_C \neq 0, & \text{if TC[0] = 0b1 (single edge).} \end{cases}$$

Note that V_P and V_C are always one or zero.

R_{KMZMV} V_P is UNKNOWN if the configuration of the PMU changes such that the event counter <n> might have counted a different event, or not counted any event. For example, when any of the following occur:

- A write to any of the monitor configuration registers that changes the register.
- Any action that either enables or disables event counter <n>.

I_{CMSKR} The edge condition is applied *after* evaluating the threshold condition.

X_{FHSJK} The edge condition is checking when the threshold condition changes, rather than for $0 \leftrightarrow$ nonzero edges on V_T. This is equivalent for all cases other than when the threshold value is set to zero and the threshold condition set to equals or greater-than-or-equal, neither of which is a useful condition.

This means that TC[0], which controls whether V_T is V_B or one when the threshold condition is satisfied, can be reused for the edge condition condition when edge counting is enabled.

X_{HKNCP} TC[1] inverts the threshold condition, meaning it also controls which edge is detected by the edge condition when TC[0] selects a single edge.

Table 3.4: Edge condition (threshold extension implemented)

TC[2:0]	TE	Description
X	0b0	Disabled. Described by threshold extension.
0b000	0b1	Reserved.
0b001	0b1	Equal to not-equal (EQ $\rightarrow$ NE).
0b010	0b1	Equal to/from not-equal (EQ $\leftrightarrow$ NE).
0b011	0b1	Not-equal to equal (NE $\rightarrow$ EQ).
0b100	0b1	Reserved.
0b101	0b1	Less-than to greater-than-or-equal (LT $\rightarrow$ GE).
0b110	0b1	Less-than to/from greater-than-or-equal (LT $\leftrightarrow$ GE).
0b111	0b1	Greater-than-or-equal to less-than (GE $\rightarrow$ LT).

3.6.2.2 Threshold extension not implemented

When the edge-detect extension is implemented and the threshold extension is not implemented by an event counter <n>:

R_{JLHWP}

- If the threshold extension is not implemented, the event counter implements a 2-bit control field, TE.

Figure 3.4: Edge control (thresold extension not implemented)

R_{MZGNH} • The *edge condition* is controlled by TE.

D_{YLPKX} • If TE is not `0b00`, *edge counting* for event counter <n> is *enabled*.

• If TE is `0b00`, edge counting for event counter <n> is *disabled*.

R_{WSYHL} • V_P is the value V_B took on the previous cycle if counting the event was allowed on the previous cycle, and zero otherwise.

R_{LWKFP} • The value V_E is defined as follows:

$$V_E = \begin{cases} 1, & \text{if } V_P \text{ and } V_B \text{ satisfy the edge condition.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{edge condition} = \begin{cases} V_P = 0 \wedge V_B \neq 0, & \text{if TE} = \texttt{0b01} \text{ (rising edge).} \\ (V_P = 0 \wedge V_B \neq 0) \vee (V_P \neq 0 \wedge V_B = 0), & \text{if TE} = \texttt{0b10} \text{ (edge.)} \\ V_P \neq 0 \wedge V_B = 0, & \text{if TE} = \texttt{0b11} \text{ (falling edge).} \end{cases}$$

Table 3.5: Edge condition (threshold extension not implemented)

TE[1:0]	Description
`0b00`	Edge condition is disabled.
`0b01`	Zero to nonzero ($0 \rightarrow \neq 0$).
`0b10`	Zero to/from nonzero ($0 \leftrightarrow \neq 0$).
`0b11`	Nonzero to zero ($\neq 0 \rightarrow 0$).

3.6.3 Pseudocode

I_{SZSCQ} The operation of the threshold condition and edge condition is described by the pseudocode function `CountValue()`.

```
// CountValue()
// ============
// Implements the PMU threshold and edge functions.
// Returns the value to increment the event counter by.
// 'Vb' is the base value of the event that the event counter is configured to count.
// 'TH', 'TC', and 'TE' are control fields for the event counter


integer CountValue(integer Vb, bits(N) TH, bits(3) TC, bits(2) TE)

    integer T = UInt(TH);
    integer V;
    integer Vt;
    boolean Vc;
```

```
// Threshold condition
case TC[2:1] of
    when '00'  Vc = (Vb != T);   // Disabled or not-equal
    when '01'  Vc = (Vb == T);   // Equals
    when '10'  Vc = (Vb >= T);   // Greater-than-or-equal
    when '11'  Vc = (Vb <  T);   // Less-than

if TC[0] == '0' then
    Vt = (if Vc then Vb else 0); // Count values
else
    Vt = (if Vc then 1 else 0);  // Count matches

// Edge condition
if TE == '1' then
    // 'pmu_last_th' records 'Vc' on the previous cycle.
    Vp = pmu_last_th[n];

    tc = TC[1:0];
    // Check for reserved case
    if tc == '00' then
        (c, tc) = ConstrainUnpredictableBits();
        if c == Contraint_DISABLED then tc = '00';

    case tc of
        when '00'  V = Vt;       // Reserved - treat as disabled
        when '10'  V = (if Vp != Vc then 1 else 0);   // Both edges
        when 'x1'  V = (if !Vp && Vc then 1 else 0);  // Single edge
else
    V = Vt;

pmu_last_th[n] = Vc;

return V;
```

3.7 Reusable event filter definitions

I_{FQPMH} The *reusable event filter* definitions are reusable programmers' models for conditioning the behavior of a monitor, based on an operating state of the component generating the event or an attribute of the event.

Each filter defines the control fields used to program filtering, but it is IMPLEMENTATION DEFINED where these fields appear in the monitor configuration registers.

Some of these fields are optional and might be implemented as RES0 or not implemented. *Not implemented* allows the bit position to have other uses in the PMU.

R_{JMSQC} For each reusable event filter It is IMPLEMENTATION DEFINED whether each reusable event filter is supported by all, some, or no monitors in the PMU, and, when supported, which events or monitored characteristics support filtering.

I_{CBWPY} If non-invasive debug of an operating state is prohibited, then R_{VHTGC} requires counting the event or monitoring the characteristic is prohibited, even if a reusable event filter is programmed to match that state.

See also:

- 2.6 *Security*.

3.7.1 Security operating state filtering

I_{GHXKJ} For a PMU monitoring an agent that implements multiple Security states or processes operations attributable to mulitple Physical Address spaces, the *Security state filtering extension* is a reusable event filter for conditioning the behavior of a monitor on the Security state or PAS.

For example, if an event counter counts events related to transactions on a bus, where each transaction has an associated PAS, then these controls can be used to configure to count:

- Only transactions related to one or more PAS.
- All transactions.

R_{JYHZX} When the Security state filtering extension is implemented by a monitor, the monitor implements a group of control fields, the State match controls. This group is a contiguous set of bits from one of the monitor configuration registers.

I_{CSLXT} Each Security state or PAS corresponds to an operating state.

The State match controls bit assignments are:

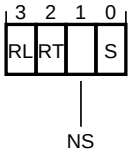


Figure 3.5: State match controls

RL, bit [3]
Realm state.

When Realm state is implemented
Configures matching Realm state. The possible values of this bit are:

0b0	Events or characteristics attributable to Realm state are unaffected by this bit.

0b1	Do not count events or monitor characteristics attributable to Realm state.

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Realm operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Realm operation of the agent being monitored.
- Otherwise, this bit is read/write.

Otherwise
Reserved. This bit is RES0.

RT, bit [2]
Root state.

When Root state is implemented
Configures matching Root state. The possible values of this bit are:

0b0	Events or characteristics attributable to Root state are unaffected by this bit.
0b1	Do not count events or monitor characteristics attributable to Root state.

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Root operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Root operation of the agent being monitored.
- Otherwise, this bit is read/write.

Otherwise
Reserved. This bit is RES0.

NS, bit [1]
Non-secure state.

When Non-secure state is implemented
Configures matching Non-secure state. The possible values of this bit are:

0b0	Events or characteristics attributable to Non-secure state are unaffected by this bit.
0b1	Do not count events or monitor characteristics attributable to Non-secure state.

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Non-secure operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Non-secure operation of the agent being monitored.
- Otherwise, this bit is read/write.

Otherwise
Reserved. This bit is RES0.

S, bit [0]

Secure state.

When Secure state is implemented
Configures matching Secure state. The possible values of this bit are:

0b0	Events or characteristics attributable to Secure state are unaffected by this bit.
0b1	Do not count events or monitor characteristics attributable to Secure state.

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Secure operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Secure operation of the agent being monitored.
- Otherwise, this bit is read/write.

Otherwise
Reserved. This bit is RES0.

3.7.2 Exception level filtering

I_{PDWGG} For a PMU monitoring an agent that implements multiple Exception levels and Security states or processes operations attributable to Exception levels and Security states, the *Exception level state filtering extension* is a reusable event filter for conditioning the behavior of a monitor on the Security state and Exception level.

For example, if an event counter counts events related to operations executed by a PE that implements the Exception level model described by the *Arm® Architecture Reference Manual, for A-profile architecture* [3], then these controls can be used to configure to count:

- Only operations related to one Exception level.
- Only operations related to one or more Exception levels in a Security state.
- All operations.

R_{KWDKJ} When the Exception level state filtering extension is implemented by a monitor, the monitor implements a group of control fields, the Exception level match controls. This group is a contiguous set of bits from one of the monitor configuration registers.

I_{TTVKD} Each Exception level and Security state corresponds to an operating state.

I_{MTVQQ} In the Armv8-A Performance Monitors Extension, bits [31:26,24,22:20] of the PMEVTYPER<n>_EL0 and PMCCFILTR_EL0 registers implement the Exception level match controls. Bits [25,23] are assigned for other purposes.

The Exception level match controls bit assignments are:

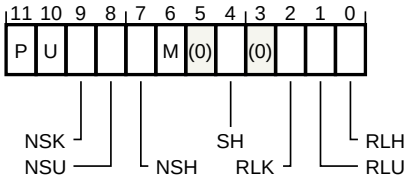


Figure 3.6: Exception level match controls

P, bit [11]
EL1 filtering. Controls counting events attributable to EL1. The possible values of this bit are:

0b0	This bit has no effect on filtering of events.
0b1	Events attributable to EL1 are not counted.

If Secure and Non-secure states are implemented, then counting events attributable to Non-secure EL1 is further controlled by NSK.

If FEAT_RME is implemented, then counting events attributable to Realm EL1 is further controlled by RLK.

If EL3 is implemented, then counting events attributable to EL3 is further controlled by M.

U, bit [10]
EL0 filtering. Controls counting events attributable to EL0. The possible values of this bit are:

0b0	This bit has no effect on filtering of events.
0b1	Events attributable to EL0 are not counted.

If Secure and Non-secure states are implemented, then counting events attributable to Non-secure EL0 is further controlled by NSU.

If FEAT_RME is implemented, then counting events attributable to Realm EL0 is further controlled by RLU.

NSK, bit [9]
Non-secure EL1 filtering.

When Secure and Non-secure states are implemented
Controls counting events attributable to Non-secure EL1. If NSK is not equal to P, then events attributable to Non-secure EL1 are not counted. Otherwise, NSK has no effect on filtering of events attributable to Non-secure EL1. The possible values of this bit are:

0b0	If P == 0b0, then this bit has no effect on filtering of events. If P == 0b1, then events attributable to Non-secure EL1 are not counted.
0b1	If P == 0b0, then events attributable to Non-secure EL1 are not counted. If P == 0b1, then this bit has no effect on filtering of events.

Otherwise
Reserved. This bit is RES0.

NSU, bit [8]
Non-secure EL0 filtering.

When Secure and Non-secure states are implemented
Controls counting events attributable to Non-secure EL0. If NSU is not equal to U, then events attributable to Non-secure EL0 are not counted. Otherwise, NSU has no effect on filtering of events attributable to Non-secure EL0. The possible values of this bit are:

0b0	If U == 0b0, then this bit has no effect on filtering of events. If U == 0b1, then events attributable to Non-secure EL0 are not counted.
0b1	If U == 0b0, then events attributable to Non-secure EL0 are not counted. If U == 0b1, then this bit has no effect on filtering of events.

Otherwise
Reserved. This bit is RES0.

NSH, bit [7]
EL2 filtering.

When EL2 is implemented
Controls counting events attributable to EL2. The possible values of this bit are:

0b0	Events attributable to EL2 are not counted.
0b1	This bit has no effect on filtering of events.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting events attributable to Secure EL2 is further controlled by SH.

If FEAT_RME is implemented, then counting events attributable to Realm EL2 is further controlled by RLH.

Otherwise
Reserved. This bit is RES0.

M, bit [6]
EL3 filtering.

When EL3 is implemented
Controls counting events attributable to EL3. If M is not equal to P, then events attributable to EL3 are not counted. Otherwise, M has no effect on filtering of events attributable to EL3. The possible values of this bit are:

0b0	If P == 0b0, then this bit has no effect on filtering of events. If P == 0b1, then events attributable to EL3 are not counted.
0b1	If P == 0b0, then events attributable to EL3 are not counted. If P == 0b1, then this bit has no effect on filtering of events.

Otherwise
Reserved. This bit is RES0.

Bits [5,3]
Reserved. This field is RES0.

SH, bit [4]
Secure EL2 filtering.

When Secure and Non-secure states are implemented, and FEAT_SEL2 is implemented
Controls counting events attributable to Secure EL2. If SH is equal to NSH, then events attributable to Secure EL2 are not counted. Otherwise, SH has no effect on filtering of events attributable to Secure EL2. The possible values of this bit are:

0b0	If NSH == 0b0, then events attributable to Secure EL2 are not counted. If NSH == 0b1, then this bit has no effect on filtering of events.
0b1	If NSH == 0b0, then this bit has no effect on filtering of events. If NSH == 0b1, then events attributable to Secure EL2 are not counted.

Otherwise
Reserved. This bit is RES0.

RLK, bit [2]
Realm EL1 filtering.

When Realm state is implemented
Controls counting events attributable to Realm EL1. If RLK is not equal to P, then events attributable to Realm EL1 are not counted. Otherwise, RLK has no effect on filtering of events attributable to Realm EL1. The possible values of this bit are:

0b0	If P == 0b0, then this bit has no effect on filtering of events. If P == 0b1, then events attributable to Realm EL1 are not counted.
0b1	If P == 0b0, then events attributable to Realm EL1 are not counted. If P == 0b1, then this bit has no effect on filtering of events.

Otherwise
Reserved. This bit is RES0.

RLU, bit [1]
Realm EL0 filtering.

When Realm state is implemented
Controls counting events attributable to Realm EL0. If RLU is not equal to U, then events attributable to Realm EL0 are not counted. Otherwise, RLU has no effect on filtering of events attributable to Realm EL0. The possible values of this bit are:

0b0	If U == 0b0, then this bit has no effect on filtering of events. If U == 0b1, then events attributable to Realm EL0 are not counted.
0b1	If U == 0b0, then events attributable to Realm EL0 are not counted. If U == 0b1, then this bit has no effect on filtering of events.

Otherwise
Reserved. This bit is RES0.

RLH, bit [0]
Realm EL2 filtering.

When Realm state is implemented
Controls counting events attributable to Realm EL2. If RLH is equal to NSH, then events attributable to Realm EL2 are not counted. Otherwise, RLH has no effect on filtering of events attributable to Realm EL2. The possible values of this bit are:

0b0	If NSH == 0b0, then events attributable to Realm EL2 are not counted. If NSH == 0b1, then this bit has no effect on filtering of events.
0b1	If NSH == 0b0, then this bit has no effect on filtering of events. If NSH == 0b1, then events attributable to Realm EL2 are not counted.

Otherwise
Reserved. This bit is RES0.

3.7.3 VMID filtering

I_{BJXRK} For a PMU monitoring an agent that supports a *Virtual Machine Identifier* (VMID), the *VMID filtering extension* is a reusable event filter for conditioning the behavior of a monitor on VMID.

For example, if an event counter counts events related to transactions on a bus, where each transaction has an associated VMID, then these controls can be used to configure to count only transactions related to a specific VMID.

Each VMID is defined in a *VMID space*, and the VMID filtering extension also allows the user to specify the VMID space as well the VMID to match against.

I_{GKDMV} Each VMID space corresponds to an operating state.

R_{SBCLF} When the VMID filtering extension is implemented by a monitor, the monitor implements two groups of control fields. Each group is a contiguous set of bits from one of the monitor configuration registers. The two groups are:

- The match controls, VMID_SP, MATCH_VMID, and MATCH_VMID_SP.
- The match value, VMID.

The VMID match controls bit assignments are:

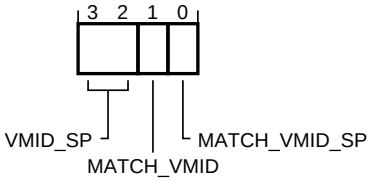


Figure 3.7: VMID match controls

VMID_SP, bits [3:2]
Virtual Machine Identifier space. Selects the VMID space to match. The possible values of this field are:

0b00	Only count events or monitor characteristics attributable to a VMID in the Secure VMID space.
0b01	Only count events or monitor characteristics attributable to a VMID in the Non-secure VMID space.
0b11	Only count events or monitor characteristics attributable to a VMID in the Realm VMID space.

All other values are reserved.

If this PMU is not allowed to observe events or characteristics attributable to Secure or Realm operation of the agent being monitored, then the values for the corresponding VMID spaces behave as 0b01.

Values corresponding to unimplemented VMID spaces, or VMID spaces that this PMU never monitors, are Reserved and behave as 0b01.

If Realm VMID space is not implemented, then VMID_SP[1] is RES0, and might not be implemented, and bit [0] of this field might be named VMID_NS.

If only a single VMID space is implemented, then VMID_SP is RES0, and might not be implemented.

If any of the following apply, VMID_SP is ignored and the PMU will match events attributable to any VMID space:

- MATCH_VMID_SP is implemented and MATCH_VMID_SP is 0b0.
- MATCH_VMID_SP is not implemented and MATCH_VMID is 0b0.

MATCH_VMID, bit [1]
Match on Virtual Machine Identifier. The possible values of this bit are:

0	VMID is ignored.
1	Only count events or monitor characteristics attributable to the Virtual Machine ID specified by VMID and, if applicable, VMID_SP.

MATCH_VMID_SP, bit [0]
Match on VMID space. The possible values of this bit are:

0	VMID_SP is ignored. Monitor events with any VMID space.
1	Only count events or monitor characteristics attributable to a Virtual Machine ID in the VMID space specified by VMID_SP.

This bit is RES0 and might be not implemented if VMID_SP is RES0 or not implemented.

Otherwise, this bit is optional. When MATCH_VMID_SP is not implemented, the PMU behaves as if MATCH_VMID_SP has the same value as MATCH_VMID.

If Realm VMID space is not implemented, then this bit might be named MATCH_VMID_NS.

The VMID match values bit assignments are:

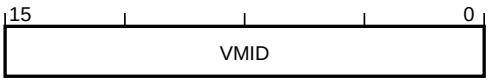


Figure 3.8: VMID match values

VMID, bits [15:0]
Virtual Machine Identifier. Selects the VMID to match.

Ignored if MATCH_VMID is 0b0.

3.7.4 MPAM filtering

I_{XRXLC} For a PMU monitoring an agent that supports *Memory System Resource Partitioning and Monitoring* (MPAM), the *MPAM filtering extension* is a reusable event filter for conditioning the behavior of a monitor on MPAM partition ID (PARTID) and MPAM performance monitoring group ID (PMG).

For example, if an event counter counts events related to transactions on a bus, where each transaction has an associated PARTID and PMG, then these controls can be used to configure to count:

- Only transactions related to a specific PARTID.
- Only transactions related to a specific PMG.
- Only transactions related to a specific PARTID and a specific PMG.
- All transactions.

Each PARTID and PMG is defined in a *PARTID space*, and the MPAM filtering extension also allows the user to specify the PARTID space as well the PARTID and PMG to match against.

I_{LFRHX} Each PARTID space defined by MPAM corresponds to an operating state.

R_{JTGBW} When the MPAM filtering extension is implemented by a monitor, the monitor implements two groups of control fields. Each group is a contiguous set of bits from one of the monitor configuration registers. The two groups are:

- The match controls, MPAM_SP, MATCH_PMG, MATCH_PARTID, and MATCH_MPAM_SP.
- The match values, PMG and PARTID.

The MPAM match controls bit assignments are:

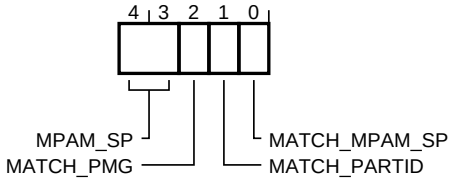


Figure 3.9: MPAM match controls

MPAM_SP, bits [4:3]
Partition Identifier space. Selects the PARTID space to match. The possible values of this field are:

0b00	Only count events or monitor characteristics attributable to a PARTID in the Secure PARTID space.
0b01	Only count events or monitor characteristics attributable to a PARTID in the Non-secure PARTID space.
0b10	Only count events or monitor characteristics attributable to a PARTID in the Root PARTID space.
0b11	Only count events or monitor characteristics attributable to a PARTID in the Realm PARTID space.

If this PMU is not allowed to observe events or characteristics attributable to any of Secure, Root, or Realm operation of the agent being monitored, then the values for the corresponding PARTID spaces behave as 0b01.

Values corresponding to unimplemented PARTID spaces, or PARTID spaces that this PMU never monitors, are Reserved and behave as 0b01.

If Root and Realm PARTID spaces are not implemented, then MPAM_SP[1] is RES0, and might not be implemented, and bit [0] of this field might be named MPAM_NS.

If only a single PARTID space is implemented, then MPAM_SP is RES0, and might not be implemented.

If any of the following apply, MPAM_SP is ignored and the PMU will match events attributable to any PARTID space:

- MATCH_MPAM_SP is implemented and MATCH_MPAM_SP is 0b0.
- MATCH_MPAM_SP is not implemented, MATCH_PMG is 0b0, and MATCH_PARTID is 0b0.

MATCH_PMG, bit [2]
Match on Performance Monitoring Group. The possible values of this bit are:

0	PMG is ignored.
1	Only count events or monitor characteristics attributable to the Performance Monitoring Group specified by PMG and, if applicable, MPAM_SP.

MATCH_PARTID, bit [1]
Match on Partition Identifier. The possible values of this bit are:

0	PARTID is ignored.
1	Only count events or monitor characteristics attributable to the Partition ID specified by PARTID and, if applicable, MPAM_SP.

MATCH_MPAM_SP, bit [0]
Match on PARTID space. The possible values of this bit are:

0	MPAM_SP is ignored. Monitor events with any PARTID space.
1	Only count events or monitor characteristics attributable to a Partition ID in the PARTID space specified by MPAM_SP.

This bit is RES0 and might be not implemented if MPAM_SP is RES0 or not implemented.

Otherwise, this bit is optional. When MATCH_MPAM_SP is not implemented, the PMU behaves as if MATCH_MPAM_SP is:

- 0b0, if MATCH_PMG is 0b0 and MATCH_PARTID is 0b0.
- 0b1, otherwise.

If Root and Realm PARTID spaces are not implemented, then this bit might be named MATCH_NS.

The MPAM match values bit assignments are:

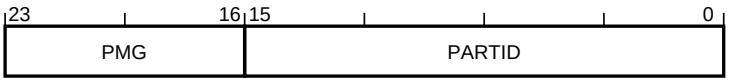


Figure 3.10: MPAM match values

PMG, bits [23:16]
Performance Monitoring Group. Selects the PMG to match.

Ignored if MATCH_PMG is 0b0.

Only sufficient low-order bits are required to represent the PMG_MAX. Higher-order bits are RES0.

PARTID, bits [15:0]
Partition Identifier. Selects the PARTID to match.

Ignored if MATCH_PARTID is 0b0.

Only sufficient low-order bits are required to represent the PARTID_MAX. Higher-order bits are RES0.

See also:

- *Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture* [2].

3.8 Snapshot extension

R_{WGXRN} It is IMPLEMENTATION DEFINED whether the PMU includes an optional *snapshot extension*.

R_{PDXMS} If the snapshot extension is implemented then PMCFGR.SS reads as `0b1`.

R_{DSDDX} If the snapshot extension is implemented then the PMU includes the following registers:

- An IMPLEMENTATION DEFINED number of *snapshot value registers*, PMSVR<n>.

  If the 64-bit programmers' model extension is implemented, each snapshot value register is a 64-bit register. Otherwise, each snapshot value register is a 32-bit register.
- The *snapshot control register*, PMSSCR. PMSSCR is a 64-bit register.
- An IMPLEMENTATION DEFINED number of optional *overflow status saved value registers*, PMOVSSR<n>.

  If the 64-bit programmers' model extension is implemented, each overflow status saved value register is a 64-bit register. Otherwise, each overflow status saved value register is a 32-bit register, but pairs of overflow status saved value registers might be accessible as a single 64-bit register. See also R_{ZNTBP}.
- An optional snapshot reset register, PMSSRR. PMSSRR is a 64-bit register.

R_{SCWKJ} In some implementations, when the 64-bit programmers' model extension is not implemented, part of the snapshot control register is replaced by a separate Snapshot Status Register, PMSSSR. This includes the *No capture* bit, PMSSSR.NC.

When PMSSSR is implemented:

- PMSSSR is a 32-bit register, implemented as an alias for PMSVR<n>, where *n* is IMPLEMENTATION DEFINED.
- PMSSCR is the 32-bit *snapshot capture register*.

R_{DNJSY} On a Capture event, the PMU writes snapshot values to the snapshot value registers, and then sets PMSSCR.NC to zero to indicate a successful capture.

I_{NNVZK} The snapshot values typically include:

- The event monitors, PMEVCNTR<m>.
- PMCCNTR, if implemented.
- PMSSSR, if implemented.
- The overflow status flags, captured into PMOVSSR<n>, if implemented.
- Any additional IMPLEMENTATION DEFINED syndrome information captured by the PMU.

Up-to 128 32-bit values or sixty-four 64-bit values can be captured.

I_{SSNNY} The snapshot extension might not capture all the implemented monitors, due to the limited number of snapshot value registers and the need to capture PMSSSR (if implemented), PMOVSSR<n> (if implemented), and/or any other IMPLEMENTATION DEFINED capture information.

R_{CFXNM} The mapping of snapshot values to the snapshot value registers, is IMPLEMENTATION DEFINED.

R_{BXVMT} When the *3.13 64-bit programmers' model extension* is not implemented, it is IMPLEMENTATION DEFINED and recommended that all of the following apply:

- Any 64-bit snapshot value is mapped to a pair of 32-bit snapshot value registers, PMSVR<n> and PMSVR<n+1>, where *n* is even.
- Doubleword-aligned 64-bit accesses to pair of registers are supported.

I_{TRJQW} When implemented, each PMOVSSR<m> is an alias for PMSVR<n>, where *n* is IMPLEMENTATION DEFINED.

R_{CVTPF} When PMSSRR is implemented, for each monitor *n*, if *n* is less than 64 and PMSSRR[n] is `0b1` then PMEVCNTR<n> and PMOVS[n] are reset to zero after a Capture event.

R_{NBXGD} When PMSSRR is implemented, the overflow status saved value registers are implemented for at least the first 64 monitors.

R_{ZXZYK} A *Capture event* is generated by one of:

- Assertion of an IMPLEMENTATION DEFINED snapshot request signal, **PMUSNAPSHOTREQ**.
- Writing `0b1` to PMSSCR.SS or PMSSCR.SS.

3.9 Trace generation extension

R_{LBLQK} It is IMPLEMENTATION DEFINED whether the PMU can generate a *trace* of events or event monitor data. If the PMU can generate a trace of events or event monitor data then it is IMPLEMENTATION DEFINED whether the PMU includes an additional control to enable trace, PMCR.TRO.

R_{SRKVT} If the PMU implements PMCR.TRO then PMCFGR.TRO reads as `0b1`.

R_{ZNVML} Trace is not generated for prohibited events. See 2.6 *Security*.

R_{NVRCD} The details of any trace generated by the PMU are IMPLEMENTATION DEFINED.

3.10 Export extension

R_{TYMMH} It is IMPLEMENTATION DEFINED whether events are *exported* to an external monitoring agents to provide triggering information. If events are exported then it is IMPLEMENTATION DEFINED whether the PMU includes an additional control to gate export of events, PMCR.X.

R_{JSMMJ} If the PMU implements PMCR.X then PMCFGR.EX reads as `0b1`.

R_{YJZSC} Prohibited events are not exported. See 2.6 *Security*.

3.11 Dual-page extension

R_{FJVQZ} It is IMPLEMENTATION DEFINED whether the PMU is programmed through a single page of registers, or *two pages* of registers.

R_{XZWHB} The following are Page 1 registers:

- PMEVCNTR<n>.
- PMOVSCLR<n>.
- PMOVSSET<n>.

R_{MBLCQ} If the direct access to enables and overflows extension is implemented, the following is a Page 1 register:

- PMOVS.

R_{PVRYN} If the fixed-function cycle counter extension is implemented, the following is a Page 1 register:

- PMCCNTR.

R_{MYXPD} If the snapshot extension is implemented, the following are Page 1 registers:

- PMSVR<n>.
- PMOVSSR<n>.
- PMSSSR.

R_{HXZHB} The following registers are both Page 0 and Page 1 registers:

- PMIIDR.
- PMDEVAFF.
- PMDEVID.
- PMPIDR0.
- PMPIDR1.
- PMPIDR2.
- PMPIDR3.
- PMPIDR4.
- PMPIDR5.
- PMPIDR6.
- PMPIDR7.
- PMCIDR0.
- PMCIDR1.
- PMCIDR2.
- PMCIDR3.

The following registers are both Page 0 and Page 1 registers that, in a dual-page implementation, do not have the same value in both Page 0 and Page 1 views:

- PMCFGR.
- PMDEVARCH.
- PMDEVTYPE.

R_{FQFQP} For each IMPLEMENTATION DEFINED register, it is IMPLEMENTATION DEFINED whether the register is a Page 0 register, a Page 1 register, or both.

R_{MLQHD} All PMU registers that are not Page 1 registers are Page 0 registers.

R_{RSCXF} In a dual-page implementation:

- *Page 0* registers are memory-mapped at architecturally-defined offsets from the IMPLEMENTATION DEFINED Page 0 base addresses.
- *Page 1* registers are memory-mapped at architecturally-defined offsets from the IMPLEMENTATION DEFINED Page 1 base addresses.

R_{BVDMH} In a single-page implementation, all Page 0 and Page 1 registers are memory-mapped at architecturally-defined offsets from an IMPLEMENTATION DEFINED base address.

R_{JYWDL} Base addresses must be aligned to a multiple of 4KB.

I_{BKZLQ} Arm recommends that base addresses are aligned to a multiple of 64KB.

S_{TCPZJ} A dual-page implementation allows a Hypervisor to provide different accessibility to the Page 0 and Page 1 registers to a Guest operating system.

For example, the Guest can read performance monitors directly from the Page 1 registers, but writes to Page 1 and any accesses to Page 0 registers generate a stage 2 translation fault and can be emulated by the Hypervisor.

R_{QCGTS} If the PMU implements dual-pages then this is identified through the unique PMDEVARCH values for the two pages.

I_{CQFCY} The Arm Performance Monitors Extensions for A-profile architecture do not implement the dual-page view on their memory-mapped PMU interfaces. The PE implements a second view of the PMU through System registers.

I_{MZWTV} This extension does not provide any mechanism for the Page 1 view to be further restricted by the Hypervisor, beyond that provided for by an MMU. A future extension might provide controls to restrict access to monitors or monitor groups visible in the Page 1 view.

3.12 Direct access to enables and overflows extension

I_{JVKHW} The *direct access to enables and overflows extension* provides read/write registers for PMOVS, PMCNTEN, and PMINTEN through PMOVS, PMCNTEN, and PMINTEN registers.

S_{YZWCG} The direct access to enables and overflows extension allows software to:

Efficiently restore the registers
Software can efficiently save the registers without this extension. However, to restore the registers, software must write to both the set and clear registers. With this extension, restoring the registers is performed in a single write.

Atomically set and clear flags in the registers
Set and clear pairs allow software to atomically set or clear one or more flags in one of the registers. This is advantageous where monitors might be shared between different agents. However, in other use cases it is important to be able to atomically set and clear different flags in the same register. For example, to atomically flip between two set of monitors, providing a facility similar to the snapshot extension.

R_{JYPYC} When the direct access to enables and overflows extension is implemented, the 64-bit programmers' model extension is implemented.

3.13 64-bit programmers' model extension

I_{TBMTK} The *64-bit programmers' model extension* is an alternative programmers' model for devices with a native 64-bit interface.

R_{LXDDK} When the 64-bit programmers' model extension is implemented, all PMU registers are 64 bits, other than the CoreSight management registers (when implemented) excluding PMDEVAFF:

- All 64-bit registers are accessed at doubleword-aligned addresses.
- All monitor value registers, PMEVCNTR<n>, are 64-bit registers at doubleword-aligned addresses. However, it is not required that monitor values are 64 bits.
- Up to 64 monitors can be implemented.
- The monitor configuration registers, PMEVTYPER<n>, PMEVFILTR<n>, and PMEVFILT2R<n>, are 64-bit registers.
- Each of the PMCNTENSET, PMCNTENCLR, PMINTENSET, PMINTENCLR, PMOVSSET, and PMOVSCLR registers are 64-bit registers.
- The direct access to enables and overflows extension is implemented. That is, PMCNTEN, PMINTEN, and PMOVS registers are implemented. These provide direct read/write access to the monitor control and status bits.
- If monitor groups are implemented, PMCGCR<n> are 64-bit registers. Each register describes up-to 8 monitor groups.
- The PMCFGR and PMCR registers are extended to 64 bits by defining bits [63:32] as reserved, RES0.
- If the message-signaled interrupt registers are implemented, then PMIRQCR1 and PMIRQCR2 can be accessed as a single 64-bit register, PMIRQCR12.

R_{SDWBN} When the 64-bit programmers' model extension is implemented, permitted doubleword-aligned 64-bit accesses are single-copy atomic at doubleword granularity.

I_{TMQTK} Because each of the PMCNTEN, PMCNTENSET, PMCNTENCLR, PMINTEN, PMINTENSET, PMINTENCLR, PMOVS, PMOVSSET, and PMOVSCLR registers are 64-bit registers, this includes 64-bit accesses to these registers.

See also:

- Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented*.

3.14 Observability and access control extension

I_{LSRMY} The optional *observability and access control extension* provides programmable authentication controls for privileged software.

R_{QCSPZ} When the observability and access control extension is implemented and the agent being monitored by the PMU implements a Secure operating state or processes Secure operations, the PMSCR register provides the following authentication controls:

- Observability controls for which events can be counted or which characteristics can be monitored by the PMU, for each of:
  - Events or characteristics attributable to the Secure operating state or Secure operations.
  - Non-attributable events or characteristics.
- An access control for controls Non-secure access to the PMU.

  When Non-secure register access is disabled, Non-secure access to all PMU registers is RAZ/WI.
- If *message-signaled interrupts* are implemented, a control the messsage-signaled interrupt physical address space.

  When Non-secure register access is disabled, *message-signaled interrupts* are always Non-secure.

PMSCR is not accessible to Non-secure and Realm states.

I_{XJBRQ} When the agent being monitored by the PMU implements a Root operating state, the system might include additional IMPLEMENTATION DEFINED authentication controls that further restrict access to PMSCR to only Root state.

These controls might be located outside of the PMU, and are outside the scope of this architecture.

R_{BGSHZ} When the observability and access control extension is implemented and the agent being monitored by the PMU implements Root and Realm operating states or processes Root and Realm operations, the PMROOTCR register provides observability authentication controls for which events can be counted or which characteristics can be monitored by the PMU, for each of:

- Events or characteristics attributable to the Root operating state or Root operations.
- Events or characteristics attributable to the Realm operating state or Root operations.
- Non-attributable events or characteristics.

PMROOTCR is only accessible to Root state.

S_{XRWTC} Software must ensure when modifying the authentication controls that it does not inadvertently expose confidential information to an untrusted agent. As well as potentially exposing the information so that it might be observed through the PMU, the PMU might be able to generate interrupts, including message-signaled interrupts.

Message-signaled interrupts are signaled as writes, and the message-signaled interrupt registers configure the address, data, and attributes for the write.

R_{HYCBH} It is IMPLEMENTATION DEFINED which of the following apply:

- PMSCR and PMROOTCR are registers in Page 0 of the PMU. Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented* and Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented* define standard offsets for PMSCR and PMROOTCR when this is the case.
- PMSCR and PMROOTCR are registers in a separate memory page or pages, at an IMPLEMENTATION DEFINED offset in the page. This approach can be simpler for the implementation in some cases, as it means that access to the PMU pages is uniform for all registers in the PMU.

SKKWKG The observability and access control extension is expected to be configured by firmware and not used by an operating system PMU driver. There are no identification mechanisms defined in hardware for any of the following:

- Whether the observability and access control extension is implemented.
- When implemented, whether the observability and access control extension registers are implemented as part of the PMU at the standard offsets or in a separate page or pages.
- When implemented in a separate page or pages, the offsets of the PMSCR and PMROOTCR registers in those pages.

If necessary, this information has to be provided to software through, for example, a firmware description table.

See also:

- 2.2.4 *Interrupt signaling*.
- 2.6 *Security*.

Chapter 4
Programmers' Model, 64-bit programmers' model extension not implemented

This section is *normative*.

I_{BWWWJ} The programmers' model described in this manual is derived from the Armv8-A Performance Monitors Extension, which in turn is based on the similar extension to Armv7-A. In particular, the structure and layout of registers is suited to implementations that support a 32-bit Execution state.

The register descriptions in this section describe the location. layout, and content of the register when the 3.13 *64-bit programmers' model extension* is not implemented.

I_{HVTKD} The register names in this programmers' model are generalized forms. An implementation of a *Performance Monitoring Unit* (PMU) might use different names, suited to the implementation, for registers with the same function.

See also:

- 2.5 *Accessing registers*.
- Chapter 5 *Programmers' Model, 64-bit programmers' model extension implemented*.

4.1 Memory-mapped registers

R_{LLXMB} The values in the *Version* columns of the register indices refer to the following extensions:

32b Monitors up-to 32 bits in size.
64b Monitors up-to 64 bits in size.
CC 3.3 *Fixed-function cycle counter extension*.
CS CoreSight registers.
MG
3.4 *Monitor group extension*.
MSI
Message-signaled interrupts. See R_{FDLKQ}.

RME
Realm Management Extensions. The component incorporating the PMU implements Root and Realm operating states, or processes Root and Realm operations. See 2.6 *Security* and 3.14 *Observability and access control extension*.

SS 3.8 *Snapshot extension*.
TZ Security extensions. The component incorporating the PMU implements a Secure operating state, or processes Secure operations. See 2.6 *Security* and 3.14 *Observability and access control extension*.

4.1.1 When the dual-page extension is not implemented

Table 4.1: Memory-mapped register map

Offset	Access	Version	Register	Description
0x000+4×*n*	R/W	32b	PMEVCNTR<n>	Event Count Register <*n*> (up-to 32 bits)
0x000+8×*n*	R/W	64b	PMEVCNTR<n>[31:0]	Event Count Register <*n*> (up-to 64 bits), bits[31:0]
0x004+8×*n*	R/W	64b	PMEVCNTR<n>[63:32]	Event Count Register <*n*> (up-to 64 bits), bits[63:32]
0x07C	R/W	32b,CC	PMCCNTR	Cycle Count Register (up-to 32 bits)
0x0F8	R/W	64b,CC	PMCCNTR[31:0]	Cycle Count Register (up-to 64 bits), bits[31:0]
0x0FC	R/W	64b,CC	PMCCNTR[63:32]	Cycle Count Register (up-to 64 bits), bits[63:32]
0x400+4×*n*	RO or R/W		PMEVTYPER<n>	Event Type Select Register <*n*>
0x47C	RO or R/W	CC	PMCCFILTR	Cycle Counter Filter Register
0x600+4×*n*	RO	SS	PMSVR<n>	Saved Value Register <*n*>
0x600+IMPDEF	RO	SS	PMSSSR	Snapshot Status Register
0x600+IMPDEF+4×*n*	RO	SS	PMOVSSR<n>	Overflow Status Snapshot Register <*n*>
0x800+4×*n*	RO or R/W		PMEVFILT2R<n>	Event Filter 2 Register <*n*>
0xA00+4×*n*	RO or R/W		PMEVFILTR<n>	Event Filter Register <*n*>
0xC00+4×*n*	R/W		PMCNTENSET<n>	Count Enable Set Register <*n*>
0xC20+4×*n*	R/W		PMCNTENCLR<n>	Count Enable Clear Register <*n*>
0xC40+4×*n*	R/W		PMINTENSET<n>	Interrupt Enable Set Register <*n*>
0xC60+4×*n*	R/W		PMINTENCLR<n>	Interrupt Enable Clear Register <*n*>
0xC80+4×*n*	R/W		PMOVSCLR<n>	Overflow Flag Status Clear Register <*n*>
0xCC0+4×*n*	R/W		PMOVSSET<n>	Overflow Flag Status Set Register <*n*>

Offset	Access	Version	Register	Description
0xCE0+4×*n*	RO	MG	PMCGCR<n>	Counter Group Configuration Register <*n*>
0xD80+4×*n*	R/W		PMIMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE04	R/W		PMCR	Control Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xE30	WO	SS	PMSSCR	Snapshot Capture Register (PMSSSR implemented)
0xE30	R/W	SS	PMSSCR[31:0]	Snapshot Control Register (PMSSSR not implemented), bits[31:0]
0xE34	R/W	SS	PMSSCR[63:32]	Snapshot Control Register (PMSSSR not implemented), bits[63:32]
0xE38	R/W	SS	PMSSRR[31:0]	Snapshot Reset Register, bits[31:0]
0xE3C	R/W	SS	PMSSRR[63:32]	Snapshot Reset Register, bits[63:32]
0xE40	R/W	TZ	PMSCR[31:0]	Secure Control Register, bits[31:0]
0xE44	R/W	TZ	PMSCR[63:32]	Secure Control Register, bits[63:32]
0xE48	R/W	RME	PMROOTCR[31:0]	Root and Realm Control Register, bits[31:0]
0xE4C	R/W	RME	PMROOTCR[63:32]	Root and Realm Control Register, bits[63:32]
0xE80	R/W	MSI	PMIRQCR0[31:0]	Interrupt Configuration Register 0, bits[31:0]
0xE84	R/W	MSI	PMIRQCR0[63:32]	Interrupt Configuration Register 0, bits[63:32]
0xE88	R/W	MSI	PMIRQCR1	Interrupt Configuration Register 1
0xE8C	R/W	MSI	PMIRQCR2	Interrupt Configuration Register 2
0xEF8	R/W	MSI	PMIRQSR[31:0]	Interrupt Status Register, bits[31:0]
0xEFC	R/W	MSI	PMIRQSR[63:32]	Interrupt Status Register, bits[63:32]
0xFA8	RO	CS	PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO	CS	PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFB8	RO	CS	PMAUTHSTATUS	Authentication Status Register
0xFBC	RO	CS	PMDEVARCH	Device Architecture Register
0xFC8	RO	CS	PMDEVID	Device Configuration Register
0xFCC	RO	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	CS	PMPIDR4	Peripheral Identification Register 4
0xFD4	RO	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	CS	PMPIDR0	Peripheral Identification Register 0
0xFE4	RO	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	CS	PMCIDR3	Component Identification Register 3

4.1.2 Page 0, when the dual-page extension is implemented

Table 4.2: Memory-mapped register map

Offset	Access	Version	Register	Description
0x400+4×*n*	RO or R/W		PMEVTYPER<n>	Event Type Select Register <*n*>
0x47C	RO or R/W	CC	PMCCFILTR	Cycle Counter Filter Register
0x800+4×*n*	RO or R/W		PMEVFILT2R<n>	Event Filter 2 Register <*n*>
0xA00+4×*n*	RO or R/W		PMEVFILTR<n>	Event Filter Register <*n*>
0xC00+4×*n*	R/W		PMCNTENSET<n>	Count Enable Set Register <*n*>
0xC20+4×*n*	R/W		PMCNTENCLR<n>	Count Enable Clear Register <*n*>
0xC40+4×*n*	R/W		PMINTENSET<n>	Interrupt Enable Set Register <*n*>
0xC60+4×*n*	R/W		PMINTENCLR<n>	Interrupt Enable Clear Register <*n*>
0xCE0+4×*n*	RO	MG	PMCGCR<n>	Counter Group Configuration Register <*n*>
0xD80+4×*n*	R/W		PMIMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE04	R/W		PMCR	Control Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xE30	WO	SS	PMSSCR	Snapshot Capture Register (PMSSSR implemented)
0xE30	R/W	SS	PMSSCR[31:0]	Snapshot Control Register (PMSSSR not implemented), bits[31:0]
0xE34	R/W	SS	PMSSCR[63:32]	Snapshot Control Register (PMSSSR not implemented), bits[63:32]
0xE38	R/W	SS	PMSSRR[31:0]	Snapshot Reset Register, bits[31:0]
0xE3C	R/W	SS	PMSSRR[63:32]	Snapshot Reset Register, bits[63:32]
0xE40	R/W	TZ	PMSCR[31:0]	Secure Control Register, bits[31:0]
0xE44	R/W	TZ	PMSCR[63:32]	Secure Control Register, bits[63:32]
0xE48	R/W	RME	PMROOTCR[31:0]	Root and Realm Control Register, bits[31:0]
0xE4C	R/W	RME	PMROOTCR[63:32]	Root and Realm Control Register, bits[63:32]
0xE80	R/W	MSI	PMIRQCR0[31:0]	Interrupt Configuration Register 0, bits[31:0]
0xE84	R/W	MSI	PMIRQCR0[63:32]	Interrupt Configuration Register 0, bits[63:32]
0xE88	R/W	MSI	PMIRQCR1	Interrupt Configuration Register 1
0xE8C	R/W	MSI	PMIRQCR2	Interrupt Configuration Register 2
0xEF8	R/W	MSI	PMIRQSR[31:0]	Interrupt Status Register, bits[31:0]
0xEFC	R/W	MSI	PMIRQSR[63:32]	Interrupt Status Register, bits[63:32]
0xFA8	RO	CS	PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO	CS	PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFB8	RO	CS	PMAUTHSTATUS	Authentication Status Register
0xFBC	RO	CS	PMDEVARCH	Device Architecture Register
0xFC8	RO	CS	PMDEVID	Device Configuration Register
0xFCC	RO	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	CS	PMPIDR4	Peripheral Identification Register 4
0xFD4	RO	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	CS	PMPIDR0	Peripheral Identification Register 0

Offset	Access	Version	Register	Description
0xFE4	RO	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	CS	PMCIDR3	Component Identification Register 3

4.1.3 Page 1, when the dual-page extension is implemented

Table 4.3: Memory-mapped register map

Offset	Access	Version	Register	Description
0x000+4×*n*	R/W	32b	PMEVCNTR<n>	Event Count Register <*n*> (up-to 32 bits)
0x000+8×*n*	R/W	64b	PMEVCNTR<n>[31:0]	Event Count Register <*n*> (up-to 64 bits), bits[31:0]
0x004+8×*n*	R/W	64b	PMEVCNTR<n>[63:32]	Event Count Register <*n*> (up-to 64 bits), bits[63:32]
0x07C	R/W	32b,CC	PMCCNTR	Cycle Count Register (up-to 32 bits)
0x0F8	R/W	64b,CC	PMCCNTR[31:0]	Cycle Count Register (up-to 64 bits), bits[31:0]
0x0FC	R/W	64b,CC	PMCCNTR[63:32]	Cycle Count Register (up-to 64 bits), bits[63:32]
0x600+4×*n*	RO	SS	PMSVR<n>	Saved Value Register <*n*>
0x600+IMPDEF	RO	SS	PMSSSR	Snapshot Status Register
0x600+IMPDEF+4×*n*	RO	SS	PMOVSSR<n>	Overflow Status Snapshot Register <*n*>
0xC80+4×*n*	R/W		PMOVSCLR<n>	Overflow Flag Status Clear Register <*n*>
0xCC0+4×*n*	R/W		PMOVSSET<n>	Overflow Flag Status Set Register <*n*>
0xD80+4×*n*	R/W		PMIMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xFA8	RO	CS	PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO	CS	PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFBC	RO	CS	PMDEVARCH	Device Architecture Register
0xFCC	RO	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	CS	PMPIDR4	Peripheral Identification Register 4
0xFD4	RO	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	CS	PMPIDR0	Peripheral Identification Register 0
0xFE4	RO	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	CS	PMCIDR3	Component Identification Register 3

4.2 PMAUTHSTATUS, Authentication Status Register

The PMAUTHSTATUS characteristics are:

Purpose
Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Usage constraints
PMAUTHSTATUS reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMAUTHSTATUS is a 32-bit read-only memory-mapped register located at offset `0xFB8` in the Page 0 component.

4.2.1 Field descriptions

The PMAUTHSTATUS bit assignments are:

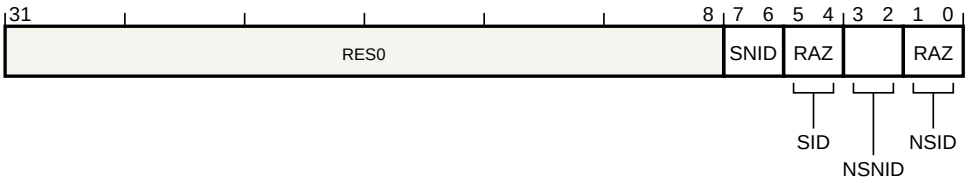


Figure 4.1: PMAUTHSTATUS

Bits [31:8]
Reserved. This field is RES0.

SNID, bits [7:6]
Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled. The defined values of this field are:

`0b00`	Secure non-invasive debug features not implemented.
`0b10`	Implemented and disabled.
`0b11`	Implemented and enabled.

All other values are reserved.

Accessing this field has the following behavior:

- This field reads-as-zeros if Security Extensions are not implemented and the PMU is Non-secure.
- Otherwise, this field is read-only.

SID, bits [5:4]
Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

The possible values of this field are:

0b00	Secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads-as-zero.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled. The defined values of this field are:

0b00	Non-secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

Accessing this field has the following behavior:

- This field reads-as-zeros if Security Extensions are not implemented and the PMU is Secure.
- Otherwise, this field is read-only.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled. The possible values of this field are:

0b00	Non-secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads-as-zero.

4.3 PMCCFILTR, Cycle Counter Filter Register

The PMCCFILTR characteristics are:

Purpose
Configures the Cycle Counter.

Usage constraints
PMCCFILTR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMCCFILTR are UNPREDICTABLE if PMCCFILTR is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations
PMCCFILTR is present only if PMCFGR.CC == `0b1`, cycle counter extension is implemented. PMCCFILTR is RES0 otherwise.

Attributes
PMCCFILTR is a 32-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x47C` in the Page 0 component.

4.3.1 Field descriptions

The PMCCFILTR bit assignments are:

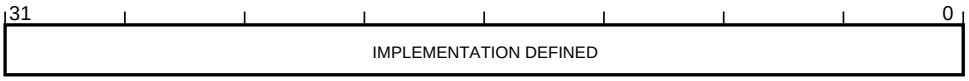


Figure 4.2: PMCCFILTR

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and, if PMCCFILTR is read/write, writes to this field have IMPLEMENTATION DEFINED behavior.

4.4 PMCCNTR, Cycle Count Register (up-to 32 bits)

The PMCCNTR characteristics are:

Purpose

If cycle counting is not prohibited and the cycle counter is enabled, the counter increments for each cycle, according to the configuration specified by PMCCFILTR.

Usage constraints

PMCCNTR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMCCNTR are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations

PMCCNTR is present only if all of the following are true:

- PMCFGR.CC == `0b1`, cycle counter extension is implemented.
- PMCFGR.SIZE <= `0b011111`, all monitors are 32 bits or smaller.

PMCCNTR is RES0 otherwise.

Attributes

PMCCNTR is a 32-bit read/write memory-mapped register located at offset `0x07C` in the Page 1 component.

4.4.1 Field descriptions

The PMCCNTR bit assignments are:

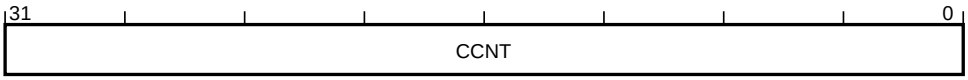


Figure 4.3: PMCCNTR

CCNT, bits [31:0]

Cycle Counter.

The number of implemented bits for PMCCNTR is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

4.5 PMCCNTR, Cycle Count Register (up-to 64 bits)

The PMCCNTR characteristics are:

Purpose

If cycle counting is not prohibited and the cycle counter is enabled, the counter increments for each cycle, according to the configuration specified by PMCCFILTR.

Usage constraints

PMCCNTR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMCCNTR are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations

PMCCNTR is present only if all of the following are true:

- PMCFGR.CC == `0b1`, cycle counter extension is implemented.
- PMCFGR.SIZE > `0b011111`, at least one monitor is larger than 32 bits.

PMCCNTR is RES0 otherwise.

Attributes

PMCCNTR is a 64-bit read/write memory-mapped register located at offset `0x0F8` in the Page 1 component.

4.5.1 Field descriptions

The PMCCNTR bit assignments are:

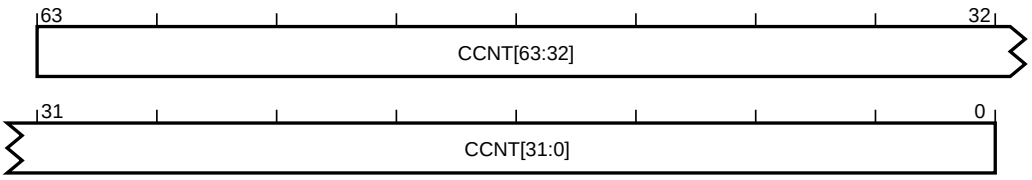


Figure 4.4: PMCCNTR

CCNT, bits [63:0]

Cycle Counter.

The number of implemented bits for PMCCNTR is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

4.6 PMCFGR, Configuration Register

The PMCFGR characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
PMCFGR reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCFGR is a 32-bit read-only memory-mapped register located at offset `0xE00`.

4.6.1 Field descriptions

The PMCFGR bit assignments are:

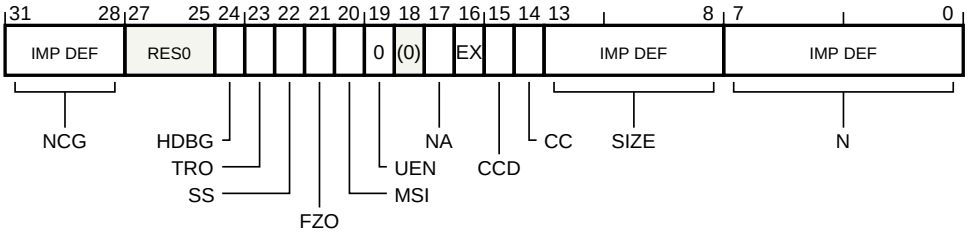


Figure 4.5: PMCFGR

NCG, bits [31:28]
Monitor Groups.

Defines the number of monitor groups implemented, minus one. If this field is zero, then one monitor group is implemented and PMCGCR<n> are not implemented.

Otherwise, for each monitor group <*m*>, PMCGCR<*m* DIV 4>.N<*m* MOD 4> defines the number of monitors in the group.

Locating the first monitor in each group depends on the number of implemented groups and the largest implemented monitor size. Each monitor group starts with monitor:

- PMEVTYPER<*m*×32>, meaning there are at most 32 monitors per group, if either:
  - Monitors are 32 bits or smaller and there are 8 monitor groups or fewer.
  - Monitors are larger-than 32 bits and there are 4 monitor groups or fewer.
- PMEVTYPER<*m*×16>, meaning there are at most 16 monitors per group, if either:
  - Monitors are 32 bits or smaller and there are more than 8 monitor groups.
  - Monitors are larger-than 32 bits and there are between 5 and 8 monitor groups.
- PMEVTYPER<*m*×8>, meaning there are at most 8 monitors per group, if monitors are larger-than 32 bits and there are more than 8 monitor groups.

This field reads as an IMPLEMENTATION DEFINED value.

Bits [27:25,18]
Reserved. This field is RES0.

HDBG, bit [24]
Halt-on-debug feature supported.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	Halt-on-debug feature not supported.
0b1	Halt-on-debug feature supported.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

TRO, bit [23]
Trace features supported.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	Trace features not supported.
0b1	Trace features supported.

The nature of any supported trace features are IMPLEMENTATION DEFINED.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

SS, bit [22]
Snapshot supported. The defined values of this bit are:

0b0	Snapshot mechanism not supported. The locations 0x600-0x7FC and 0xE30-0xE3C are IMPLEMENTATION DEFINED.
0b1	Snapshot mechanism supported. PMSVR<n> and PMSSSCR are implemented.

If the architecture-defined form of snapshot is not implemented, a PMU might include an IMPLEMENTATION DEFINED snapshot mechanism, including one using the IMPLEMENTATION DEFINED registers 0x600-0x7FC and 0xE30-0xE3C.

This bit reads as an IMPLEMENTATION DEFINED value.

FZO, bit [21]
Freeze-on-overflow supported.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	Freeze-on-overflow mechanism not supported. PMCR.FZO is RES0.
0b1	Freeze-on-overflow mechanism supported. PMCR.FZO is RW.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

MSI, bit [20]
Message-signaled interrupts (MSI) supported.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	MSI not supported. PMIRQCR<*n*> and PMIRQSR are reserved.
0b1	MSI supported. PMIRQCR<*n*> is used to configure the MSI, and PMIRQSR shows the MSI status.

If the architecture-defined form of MSI is not implemented, a PMU might nonetheless implement an MSI mechanism, including one located at the IMPLEMENTATION DEFINED registers 0xE80-0xEFC.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

UEN, bit [19]
This feature is not supported. This bit reads-as-zero.

NA, bit [17]
No write access when running.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	The monitor value registers can be written at any time. It is IMPLEMENTATION DEFINED whether the monitor configuration registers are read-only or read/write.
0b1	The monitor value and monitor configuration registers cannot be written when the PMU is not in the STOP state.

The monitor value and monitor configuration registers can be read in any state.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

EX, bit [16]
Export supported.

When the dual-page extension is not implemented or accessed in Page 0
The defined values of this bit are:

0b0	Export is not supported. PMCR.X is RES0.
0b1	Export is supported. PMCR.X is read/write.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

CCD, bit [15]

Cycle counter has pre-scale.

When the cycle counter is implemented, and the dual-page extension is not implemented or accessed in Page 0

The defined values of this bit are:

0b0	Cycle counter only ever counts every cycle. PMCR.D is RES0.
0b1	Cycle counter can count every 64th cycle. PMCR.D is read/write.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise

This bit reads-as-zero.

CC, bit [14]

Dedicated Cycle counter implemented as PMEVCNTR31. The defined values of this bit are:

0b0	If PMEVCNTR31 is implemented, it is a normal monitor. PMCR.C, PMCR.D and PMCFGR.CCD are RES0.
0b1	PMEVCNTR31 is implemented and is a dedicated cycle counter. PMCR.C is write-only.

This bit reads as an IMPLEMENTATION DEFINED value.

SIZE, bits [13:8]

Monitor size. The size of the largest implemented monitor. The defined values of this field are:

0b000111	8-bit monitors.
0b001001	10-bit monitors.
0b001011	12-bit monitors.
0b001111	16-bit monitors.
0b010011	20-bit monitors.
0b010111	24-bit monitors.
0b011111	32-bit monitors.
0b100011	36-bit monitors.
0b100111	40-bit monitors.
0b101011	44-bit monitors.
0b101111	48-bit monitors.
0b110011	52-bit monitors.
0b110111	56-bit monitors.
0b111111	64-bit monitors.

All other values are reserved.

Not all monitors are necessarily this size. For example, an implementation might include a mix of 32-bit and 64-bit monitors.

This field reads as an IMPLEMENTATION DEFINED value.

N, bits [7:0]

Number of monitors, minus one. The defined values of this field are:

0x00	1 monitor.

0x01	2 monitors.
..	..
0xFF	256 monitors.

If PMCFGR.CC == 0b1, PMEVCNTR31, the cycle counter, is one of the (N+1) monitors. For example, if PMCFGR.N == 0x00 and PMCFGR.CC == 0b1, there is a single monitor, PMEVCNTR31, and PMEVCNTR0 is not implemented.

If PMCFGR.NCG != 0b0000, then PMCFGR.N is the *total* number of monitors implemented.

If the monitors are larger-than 32 bits, then the PMU includes at most 128 monitors.

This field reads as an IMPLEMENTATION DEFINED value.

4.7 PMCGCR<*n*>, Counter Group Configuration Register <0-3>

The PMCGCR<0-3> characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
PMCGCR<*n*> reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCGCR<*n*> is present only if PMCFGR.NCG != `0x0`, monitor groups are implemented. PMCGCR<*n*> is RES0 otherwise.

Attributes
PMCGCR<*n*> is a 32-bit read-only memory-mapped register located at offset `0xCE0` + 4×*n* in the Page 0 component.

4.7.1 Field descriptions

The PMCGCR<0-3> bit assignments are:

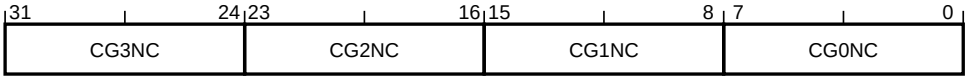


Figure 4.6: PMCGCR<*n*>

CG<*m*>NC[7:0], bits [*m*×8+7:*m*×8], for *m* = 0 to 3
Group (*n*×4+*m*) number of monitors.

The maximum size of each monitor group depends on the number of implemented groups and the largest implemented monitor size. For more information, see PMCFGR.NCG.

4.8 PMCIDR0, Component Identification Register 0

The PMCIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMCIDR0 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCIDR0 is present only if Peripheral Identification scheme is implemented. PMCIDR0 is RES0 otherwise.

Attributes
PMCIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFF0`.

4.8.1 Field descriptions

The PMCIDR0 bit assignments are:

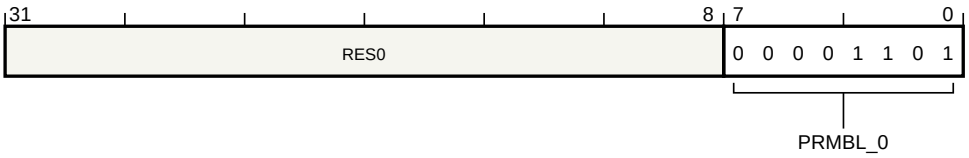


Figure 4.7: PMCIDR0

Bits [31:8]
Reserved. This field is RES0.

PRMBL_0, bits [7:0]
Component identification preamble, segment 0. This field reads as `0x0D`.

4.9 PMCIDR1, Component Identification Register 1

The PMCIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMCIDR1 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCIDR1 is present only if Peripheral Identification scheme is implemented. PMCIDR1 is RES0 otherwise.

Attributes
PMCIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFF4`.

4.9.1 Field descriptions

The PMCIDR1 bit assignments are:

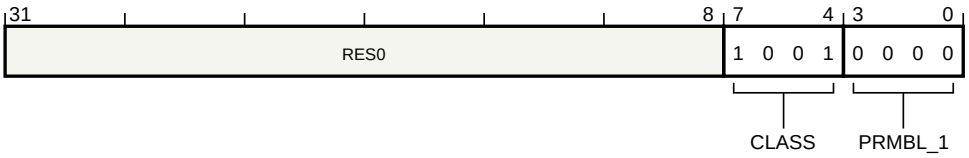


Figure 4.8: PMCIDR1

Bits [31:8]
Reserved. This field is RES0.

CLASS, bits [7:4]
Component class. The defined values of this field are:

`0x9`	CoreSight peripheral.

Other values are defined by the CoreSight Architecture.

This field reads as `0x9`.

PRMBL_1, bits [3:0]
Component identification preamble, segment 1. This field reads as `0x0`.

4.10 PMCIDR2, Component Identification Register 2

The PMCIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMCIDR2 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCIDR2 is present only if Peripheral Identification scheme is implemented. PMCIDR2 is RES0 otherwise.

Attributes
PMCIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFF8`.

4.10.1 Field descriptions

The PMCIDR2 bit assignments are:

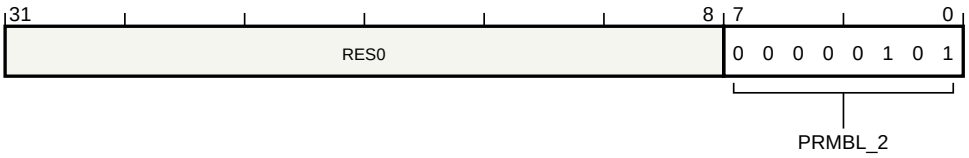


Figure 4.9: PMCIDR2

Bits [31:8]
Reserved. This field is RES0.

PRMBL_2, bits [7:0]
Component identification preamble, segment 2. This field reads as `0x05`.

4.11 PMCIDR3, Component Identification Register 3

The PMCIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMCIDR3 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCIDR3 is present only if Peripheral Identification scheme is implemented. PMCIDR3 is RES0 otherwise.

Attributes
PMCIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFFC`.

4.11.1 Field descriptions

The PMCIDR3 bit assignments are:

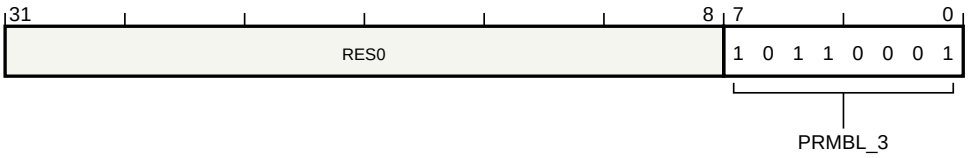


Figure 4.10: PMCIDR3

Bits [31:8]
Reserved. This field is RES0.

PRMBL_3, bits [7:0]
Component identification preamble, segment 3. This field reads as `0xB1`.

4.12 PMCNTENCLR<*n*>, Count Enable Clear Register <*n*>

The PMCNTENCLR<*n*> characteristics are:

Purpose
Disable monitors.

Usage constraints
PMCNTENCLR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCNTENCLR<*n*> is a 32-bit read/write memory-mapped register located at offset `0xC20` + 4×*n* in the Page 0 component.

4.12.1 Field descriptions

The PMCNTENCLR<*n*> bit assignments are:

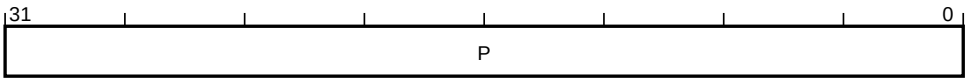


Figure 4.11: PMCNTENCLR<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
PMEVCNTR<m> disable. On writes, allows software to disable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status. The possible values of this bit are:

`0b0`	PMEVCNTR<m> disabled.
`0b1`	PMEVCNTR<m> enabled.

If the Cycle Counter extension is implemented, then PMCNTENCLR<*n*>.P[31] allows software to disable PMCCNTR and query the PMCCNTR enable status.

This bit is read/write-one-to-clear.

4.13 PMCNTENSET<*n*>, Count Enable Set Register <*n*>

The PMCNTENSET<*n*> characteristics are:

Purpose
Enable monitors.

Usage constraints
PMCNTENSET<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCNTENSET<*n*> is a 32-bit read/write memory-mapped register located at offset `0xC00` + 4×*n* in the Page 0 component.

4.13.1 Field descriptions

The PMCNTENSET<*n*> bit assignments are:

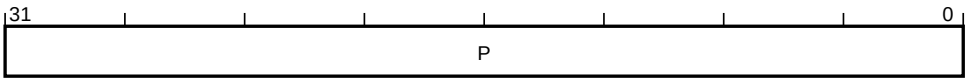


Figure 4.12: PMCNTENSET<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
PMEVCNTR<m> enable. On writes, allows software to enable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status. The possible values of this bit are:

`0b0`	PMEVCNTR<m> disabled.
`0b1`	PMEVCNTR<m> enabled.

If the Cycle Counter extension is implemented, then PMCNTENSET<*n*>.P[31] allows software to enable PMCCNTR and query the PMCCNTR enable status.

This bit is read/write-one-to-set.

4.14 PMCR, Control Register

The PMCR characteristics are:

Purpose
Main control register for the performance monitors.

Usage constraints
PMCR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCR is a 32-bit read/write memory-mapped register located at offset `0xE04` in the Page 0 component.

4.14.1 Field descriptions

The PMCR bit assignments are:

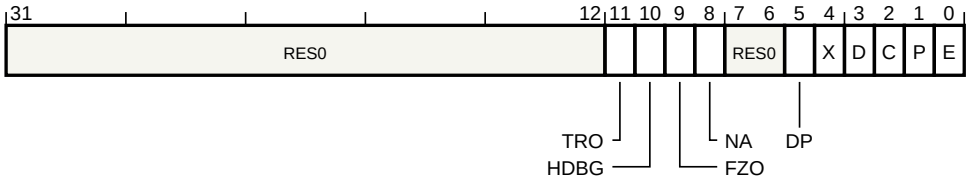


Figure 4.13: PMCR

Bits [31:12,7:6]
Reserved. This field is RES0.

TRO, bit [11]
Trace enable.

When trace features are implemented
Enable trace features. The possible values of this bit are:

`0b0`	Trace disabled.
`0b1`	Trace enabled.

Otherwise
Reserved. This bit is RES0.

HDBG, bit [10]
Halt-on-debug.

When Halt-on-debug is implemented
Stops events being counted when the affine PE or agent is in a halted state such as Debug state. The possible values of this bit are:

0b0	Do not stop counting when agent is halted.
0b1	Stop counting when agent is halted.

Otherwise
Reserved. This bit is RES0.

FZO, bit [9]
Freeze-on-overflow.

When PMCFGR.FZO == 0b1, freeze-on-overflow extension is implemented
Stop monitors on overflow. The possible values of this bit are:

0b0	Do not freeze on overflow.
0b1	Monitors do not count when PMOVS is nonzero.

Otherwise
Reserved. This bit is RES0.

NA, bit [8]
Not accessible.

When the monitors cannot be written at any time
Indicates the monitors are read-only. The defined values of this bit are:

0b0	Monitors are read/write. It is IMPLEMENTATION DEFINED whether the monitor configuration registers are read-only or read/write.
0b1	Monitors and monitor configuration registers are read-only and writes are ignored.

This bit is read-only.

Otherwise
Reserved. This bit is RES0.

DP, bit [5]
Disable cycle counter when event counting is prohibited.

When PMCFGR.CC == 0b1, cycle counter extension is implemented
The possible values of this bit are:

0b0	Cycle counting by PMCCNTR is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR is disabled in prohibited regions. Prohibited regions are IMPLEMENTATION DEFINED regions where event counting is prohibited.

Otherwise
Reserved. This bit is RES0.

X, bit [4]
Export enable.

When export of events is implemented
Permit events to be exported to another debug device, such as a trace unit, over an event bus. The possible values of this bit are:

0b0	Export of events is disabled.
0b1	Export of events is enabled.

This bit does not affect the generation of performance monitor interrupts that can be implemented as a signal exported from the PMU to an interrupt controller.

Otherwise
Reserved. This bit is RES0.

D, bit [3]
Cycle counter divider.

When PMCFGR.CC == 0b1, cycle counter extension is implemented
The possible values of this bit are:

0b0	When enabled, the cycle counter counts every clock cycle.
0b1	When enabled, the cycle counter counts once every 64 clock cycles.

Otherwise
Reserved. This bit is RES0.

C, bit [2]
Cycle counter reset.

When PMCFGR.CC == 0b1, cycle counter extension is implemented
The possible values for writing to this bit are:

0b0	Write is ignored.
0b1	Reset the cycle counter to zero.

This bit is write-only and reads-as-zero.

Note:

Resetting the cycle counter does not affect the cycle counter overflow flag.

Otherwise
Reserved. This bit is RES0.

P, bit [1]
Monitor reset.

When the PMU is not an *Activity Monitor Unit* (AMU)
The possible values for writing to this bit are:

0b0	Write is ignored.
0b1	Reset all monitors . The cycle counter is not reset.

Resetting the monitors does not affect any overflow flags.

Event counters are reset to zero.

Monitors are reset to their reset value or state, if they have one. Otherwise, monitors ignore the write to PMCR.P.

This bit is write-only and reads-as-zero.

Otherwise
This bit reads-as-zero and ignores writes.

E, bit [0]
Count enable.

When the PMU is not an AMU
Controls the performance monitor. The possible values of this bit are:

0b0	All monitors are disabled.
0b1	All monitors are enabled by PMCNTENSET.

Otherwise
This bit reads-as-zero and ignores writes.

4.15 PMDEVAFF, Device Affinity Register

The PMDEVAFF characteristics are:

Purpose

For a monitor that has affinity with a single PE or a group of PEs, PMDEVAFF is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the monitor has affinity with a single PE, the affinity level is 0, then PMDEVAFF reads the same value as MPIDR_EL1, and PMDEVAFF.F0V reads-as-one to indicate affinity level 0.
- If the monitor has affinity with a group of PEs, the affinity level is 1, 2, or 3, then parts of PMDEVAFF reads the same value as parts of MPIDR_EL1, and the rest of PMDEVAFF indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1 then all of the following are true:

- All the PEs in the group have the same values in MPIDR_EL1.{Aff3,Aff2}, and these values are equal to PMDEVAFF.{Aff3,Aff2}.
- PMDEVAFF.Aff1 is nonzero and not `0x80`, and PMDEVAFF.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the monitor has affinity with is indicated by the least-significant set bit in PMDEVAFF.Aff1. In this example, if PMDEVAFF.Aff1[2:0] is `0b100`, then the monitor has affinity with the up-to 8 PEs that have MPIDR_EL1.Aff1[7:3] == PMDEVAFF.Aff1[7:3].

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that PMDEVAFF is read before system firmware has configured the monitor and/or the PE or group of PEs that the monitor has affinity with. When this is the case, PMDEVAFF reads as zero.

Usage constraints

PMDEVAFF reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

PMDEVAFF is present only if the monitor has affinity with a PE or cluster of PEs. PMDEVAFF is RES0 otherwise.

Attributes

PMDEVAFF is a 64-bit read-only memory-mapped register located at offset `0xFA8`.

4.15.1 Field descriptions

The PMDEVAFF bit assignments are:

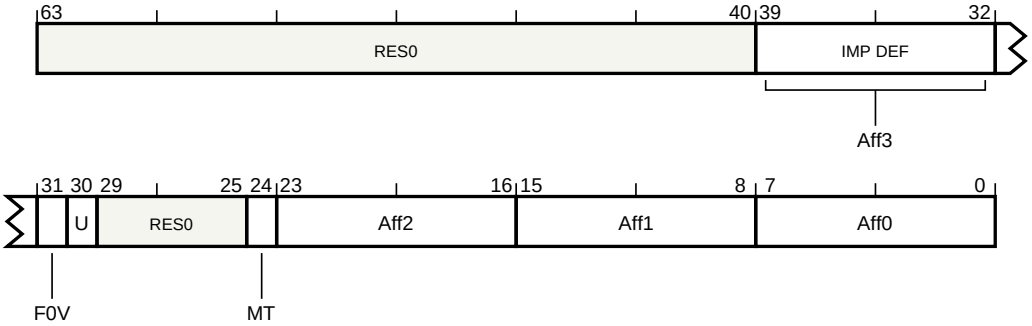


Figure 4.14: PMDEVAFF

Bits [63:40,29:25]
Reserved. This field is RES0.

Aff3, bits [39:32]
PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

This field reads as an IMPLEMENTATION DEFINED value.

F0V, bit [31]
Indicates that the PMDEVAFF.Aff0 field is valid. The defined values of this bit are:

`0b0`	PMDEVAFF.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
`0b1`	PMDEVAFF.Aff0 is valid, and the PE affinity is at level 0.

This bit reads as an IMPLEMENTATION DEFINED value.

U, bit [30]
Uniprocessor.

When PMDEVAFF.F0V == `0b1`
The MPIDR_EL1.U bit, viewed from the highest Exception level of the associated PE. This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is UNKNOWN.

MT, bit [24]
Multithreaded.

When PMDEVAFF.F0V == `0b1`
The MPIDR_EL1.MT bit, viewed from the highest Exception level of the associated PE. This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is UNKNOWN.

Aff2, bits [23:16]
PE affinity level 2.

When affine with a PE or PEs at affinity level 2 or below
The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs. This field reads as an IMPLEMENTATION DEFINED value.

When affine with a sub-set of PEs at affinity level 2
Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated

PEs. The defined values of this field are:

0bxxxxxxx1	PMDEVAFF.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PMDEVAFF.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PMDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PMDEVAFF.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PMDEVAFF.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PMDEVAFF.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PMDEVAFF.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.

This field reads as an IMPLEMENTATION DEFINED value.

Otherwise

Indicates whether the PE affinity is at level 3. The defined values of this field are:

0x80	PE affinity is at level 3.

All other values are reserved.

This field reads as 0x80.

Aff1, bits [15:8]

PE affinity level 1.

When affine with a PE or PEs at affinity level 1 or below

The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs. This field reads as an IMPLEMENTATION DEFINED value.

When affine with a sub-set of PEs at affinity level 1

Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

0bxxxxxxx1	PMDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PMDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PMDEVAFF.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PMDEVAFF.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PMDEVAFF.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PMDEVAFF.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PMDEVAFF.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.

This field reads as an IMPLEMENTATION DEFINED value.

Otherwise

Indicates whether the PE affinity is at level 2. The defined values of this field are:

0x00	PE affinity is above level 2 or a subset of level 2.
0x80	PE affinity is at level 2.

This field reads as an IMPLEMENTATION DEFINED value.

Aff0, bits [7:0]

PE affinity level 0.

When affine with a PE at affinity level 0

The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE. This field reads as an IMPLEMENTATION DEFINED value.

When affine with a sub-set of PEs at affinity level 0

Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

0bxxxxxxx1	PMDEVAFF.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PMDEVAFF.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PMDEVAFF.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PMDEVAFF.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PMDEVAFF.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PMDEVAFF.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PMDEVAFF.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.

This field reads as an IMPLEMENTATION DEFINED value.

Otherwise

Indicates whether the PE affinity is at level 1. The defined values of this field are:

0x00	PE affinity is above level 1 or a subset of level 1.
0x80	PE affinity is at level 1.

This field reads as an IMPLEMENTATION DEFINED value.

4.16 PMDEVARCH, Device Architecture Register

The PMDEVARCH characteristics are:

Purpose
Provides discovery information for the component.

Usage constraints
PMDEVARCH reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == 0b0.

Configurations
Always implemented.

Attributes
PMDEVARCH is a 32-bit read-only memory-mapped register located at offset 0xFBC.

4.16.1 Field descriptions

When PMDEVARCH is implemented, the PMDEVARCH bit assignments are:

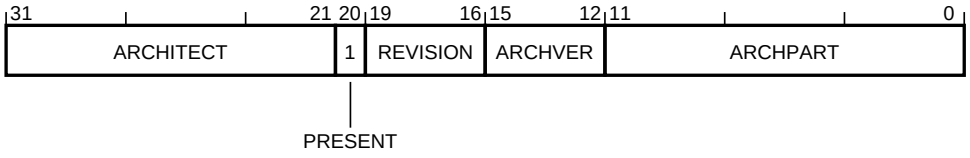


Figure 4.15: PMDEVARCH (PMDEVARCH is implemented)

When PMDEVARCH is not implemented, the PMDEVARCH bit assignments are:

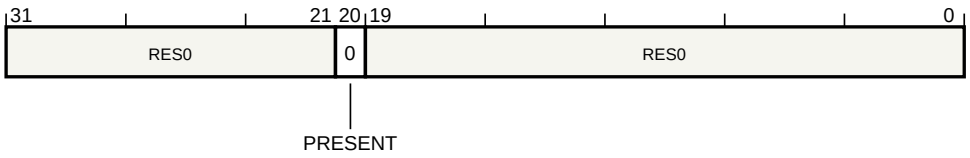


Figure 4.16: PMDEVARCH (PMDEVARCH is not implemented)

Bits [31:21,19:0], when PMDEVARCH is not implemented
Reserved. This field is RES0.

ARCHITECT, bits [31:21], when PMDEVARCH is implemented
Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code. The defined values of this field are:

0x23B	JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.
Other values	Defined by the JEDEC JEP106 standard.

When Generic CoreSight PMU
This field reads as 0x23B.

When Generic AMU
This field reads as 0x23B.

Otherwise
This field reads as an IMPLEMENTATION DEFINED value.

PRESENT, bit [20]
DEVARCH present. Defines that PMDEVARCH register is present. The defined values of this bit are:

0b0	Device Architecture information not present.
0b1	Device Architecture information present.

If PMDEVARCH is not present, the register is RES0.

REVISION, bits [19:16], when PMDEVARCH is implemented
Revision. Defines the architecture revision of the component.

When Generic CoreSight PMU
The defined values of this field are:

0b0000	Revision 0.

All other values are reserved.

This field reads as 0b0000.

When Generic AMU
The defined values of this field are:

0b0000	Revision 0.

All other values are reserved.

This field reads as 0b0000.

Otherwise
This field reads as an IMPLEMENTATION DEFINED value.

ARCHVER, bits [15:12], when PMDEVARCH is implemented
Architecture Version. Defines the architecture version of the component.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

When Generic CoreSight PMU
The defined values of this field are:

0b0000	Revision 0.

All other values are reserved.

This field reads as 0b0000.

When Generic AMU

The defined values of this field are:

0b0000	Revision 0.

All other values are reserved.

This field reads as 0b0000.

Otherwise

This field reads as an IMPLEMENTATION DEFINED value.

ARCHPART, bits [11:0], when PMDEVARCH is implemented

Architecture Part. Defines the architecture of the component.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

When Generic CoreSight PMU

The defined values of this field are:

0xA66	Generic AMU, 64-bit programmers' model extension not implemented.
0xAF0	Generic CoreSight PMU, dual-page extension not implemented, 64-bit programmers' model extension not implemented.
0xAF1	Generic CoreSight PMU Page 0, dual-page extension implemented, 64-bit programmers' model extension not implemented.
0xAF2	Generic CoreSight PMU Page 1, dual-page extension implemented, 64-bit programmers' model extension not implemented.

This field reads as an IMPLEMENTATION DEFINED value.

Otherwise

This field reads as an IMPLEMENTATION DEFINED value.

4.17 PMDEVID, Device Configuration Register

The PMDEVID characteristics are:

Purpose
Provides discovery information for the component.

Usage constraints
PMDEVID reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMDEVID is a 32-bit read-only memory-mapped register located at offset `0xFC8` in the Page 0 component.

4.17.1 Field descriptions

The PMDEVID bit assignments are:

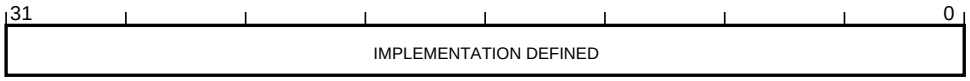


Figure 4.17: PMDEVID

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value.

4.18 PMDEVTYPE, Device Type Register

The PMDEVTYPE characteristics are:

Purpose
Provides discovery information for the component. If the part number field is not recognized, a debugger can report the information that is provided by PMDEVTYPE about the component instead.

Usage constraints
PMDEVTYPE reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMDEVTYPE is a 32-bit read-only memory-mapped register located at offset `0xFCC`.

4.18.1 Field descriptions

The PMDEVTYPE bit assignments are:

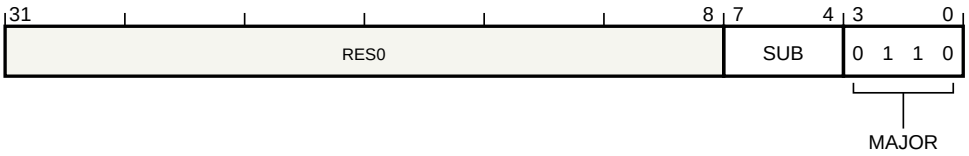


Figure 4.18: PMDEVTYPE

Bits [31:8]
Reserved. This field is RES0.

SUB, bits [7:4]
Component sub-type.

When MAJOR == `0x6` (Performance monitor), the defined values of this field are:

`0x0`	Other.
`0x1`	Associated with a PE.
`0x2`	Associated with a DSP.
`0x3`	Associated with a Data Engine or coprocessor.
`0x4`	Associated with a bus or stimulus derived from bus activity.
`0x5`	Associated with a memory management unit conforming to the Arm System MMU architecture.
`0x7`	Derived from generic signals.

Other values are defined by the CoreSight Architecture.

MAJOR, bits [3:0]

Component major type. The defined values of this field are:

`0x6`	Performance monitor.

Other values are defined by the CoreSight Architecture.

This field reads as `0x6`.

4.19 PMEVCNTR<*n*>, Event Count Register <*n*> (up-to 32 bits)

The PMEVCNTR<*n*> characteristics are:

Purpose

Monitor value <*n*>. If monitoring is not prohibited and the monitor is enabled, the monitor monitors the component as configured by PMEVTYPER<n> and PMEVFILTR<n>.

Usage constraints

PMEVCNTR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVCNTR<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations

PMEVCNTR<*n*> is present only if PMCFGR.SIZE <= `0b011111`, all monitors are 32 bits or smaller. PMEVCNTR<*n*> is RES0 otherwise.

Attributes

PMEVCNTR<*n*> is a 32-bit read/write memory-mapped register located at offset `0x000` + 4×*n* in the Page 1 component.

4.19.1 Field descriptions

The PMEVCNTR<*n*> bit assignments are:

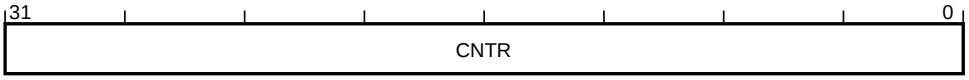


Figure 4.19: PMEVCNTR<*n*>

CNTR, bits [31:0]

Monitor value.

The number of implemented bits for PMEVCNTR<*n*> is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

4.20 PMEVCNTR<*n*>, Event Count Register <*n*> (up-to 64 bits)

The PMEVCNTR<*n*> characteristics are:

Purpose

Monitor value <*n*>. If monitoring is not prohibited and the monitor is enabled, the monitor monitors the component as configured by PMEVTYPER<n> and PMEVFILTR<n>.

Usage constraints

PMEVCNTR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVCNTR<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations

PMEVCNTR<*n*> is present only if PMCFGR.SIZE > `0b011111`, at least one monitor is larger than 32 bits. PMEVCNTR<*n*> is RES0 otherwise.

Attributes

PMEVCNTR<*n*> is a 64-bit read/write memory-mapped register located at offset `0x000` + 8×*n* in the Page 1 component.

4.20.1 Field descriptions

The PMEVCNTR<*n*> bit assignments are:

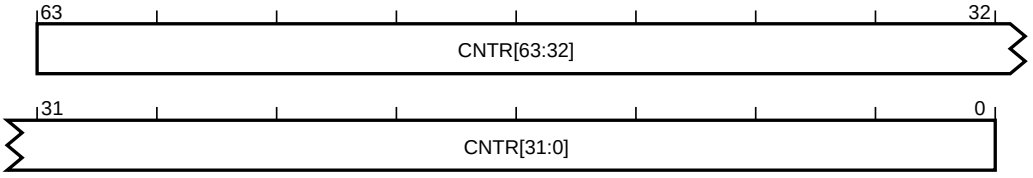


Figure 4.20: PMEVCNTR<*n*>

CNTR, bits [63:0]

Monitor value.

The number of implemented bits for PMEVCNTR<*n*> is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

4.21 PMEVFILT2R<*n*>, Event Filter 2 Register <*n*>

The PMEVFILT2R<*n*> characteristics are:

Purpose

For performance monitors requiring event selection controls in addition to the PMEVTYPER<n> and PMEVFILTR<n> registers.

Usage constraints

PMEVFILT2R<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

Always implemented.

Attributes

PMEVFILT2R<*n*> is a 32-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x800` + 4×*n* in the Page 0 component.

4.21.1 Field descriptions

The PMEVFILT2R<*n*> bit assignments are:

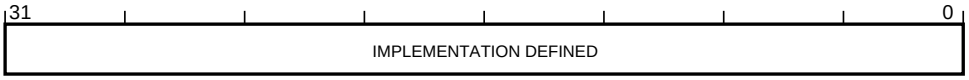


Figure 4.21: PMEVFILT2R<*n*>

Bits [31:0]

This field reads as an IMPLEMENTATION DEFINED value and, if PMEVFILT2R<*n*> is read/write, writes to this field have IMPLEMENTATION DEFINED behavior.

4.22 PMEVFILTR<*n*>, Event Filter Register <*n*>

The PMEVFILTR<*n*> characteristics are:

Purpose

For performance monitors requiring event selection controls in addition to the PMEVTYPER<n> register.

Usage constraints

PMEVFILTR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVFILTR<*n*> are UNPREDICTABLE if PMEVFILTR<*n*> is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations

Always implemented.

Attributes

PMEVFILTR<*n*> is a 32-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0xA00` + 4×*n* in the Page 0 component.

4.22.1 Field descriptions

The PMEVFILTR<*n*> bit assignments are:

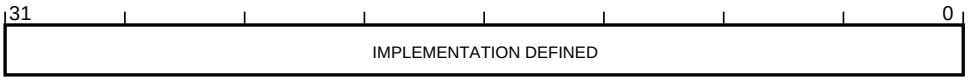


Figure 4.22: PMEVFILTR<*n*>

Bits [31:0]

This field reads as an IMPLEMENTATION DEFINED value and, if PMEVFILTR<*n*> is read/write, writes to this field have IMPLEMENTATION DEFINED behavior.

4.23 PMEVTYPER<*n*>, Event Type Select Register <*n*>

The PMEVTYPER<*n*> characteristics are:

Purpose
Configures monitor <*n*>.

Usage constraints
PMEVTYPER<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVTYPER<*n*> are UNPREDICTABLE if PMEVTYPER<*n*> is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMEVTYPER<*n*> is a 32-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x400` + 4×*n* in the Page 0 component.

4.23.1 Field descriptions

The PMEVTYPER<*n*> bit assignments are:

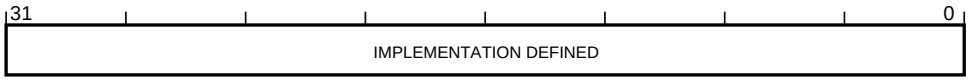


Figure 4.23: PMEVTYPER<*n*>

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and, if PMEVTYPER<*n*> is read/write, writes to this field have IMPLEMENTATION DEFINED behavior.

4.24 PMIIDR, Implementation Identification Register

The PMIIDR characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMIIDR reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
It is IMPLEMENTATION DEFINED whether PMIIDR is present. PMIIDR is RES0 if not present.

Attributes
PMIIDR is a 32-bit read-only memory-mapped register located at offset `0xE08`.

4.24.1 Field descriptions

The PMIIDR bit assignments are:

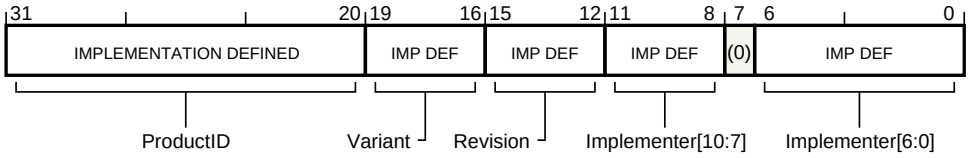


Figure 4.24: PMIIDR

ProductID, bits [31:20]
Part number, bits [11:0]. The part number is selected by the designer of the component.

Matches the {PMPIDR1.PART_1, PMPIDR0.PART_0} fields, if PMPIDR0 and PMPIDR1 are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Variant, bits [19:16]
Component major revision.

Defines either a variant of the component defined by PMIIDR.ProductID, or the major revision of the component.

When defining a major revision, PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component, with PMIIDR.Variant being the most significant part and PMIIDR.Revision the least significant part. When a component is changed, PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component. If PMIIDR.Variant is increased then PMIIDR.Revision should be set to `0b0000`.

Matches the PMPIDR2.REVISION field, if PMPIDR2 is also present.

This field reads as an IMPLEMENTATION DEFINED value.

Revision, bits [15:12]

Component minor revision.

When a component is changed:

- If PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component then:
  - PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.
  - If Variant is increased then Revision should be set to `0b0000`.
- Otherwise, PMIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.

Matches the PMPIDR3.REVAND field, if PMPIDR3 is also present.

This field reads as an IMPLEMENTATION DEFINED value.

Implementer, bits [11:8,6:0]

JEDEC-assigned JEP106 identification code of the designer of the component.

PMIIDR[11:8] is the JEP106 bank identifier minus 1 and PMIIDR[6:0] is the JEP106 identification code for the designer of the component. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note:

For example, for a component designed by Arm Limited, the JEP106 bank is 5, and the JEP106 identification code is `0x3B`, meaning PMIIDR[11:0] has the value `0x43B`.

Zero is not a valid JEP106 identification code, meaning a value of zero for PMIIDR indicates this register is not implemented.

PMIIDR[11:8] matches PMPIDR4.DES_2 and PMIIDR[6:0] match the {PMPIDR2.DES_1, PMPIDR1.DES_0} fields, if PMPIDR{1,2,4} are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Bit [7]

Reserved. This bit is RES0.

4.25 PMIMPDEF<*n*>, IMPLEMENTATION DEFINED Register <0-31>

The PMIMPDEF<0-31> characteristics are:

Purpose

IMPLEMENTATION DEFINED extensions.

Usage constraints

PMIMPDEF<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

If the dual-page extension is implemented, then for each PMIMPDEF<*n*>, it is IMPLEMENTATION DEFINED whether the register is a Page 0 register, a Page 1 register, or both.

Attributes

PMIMPDEF<*n*> is a 32-bit read/write memory-mapped register located at offset `0xD80` + 4×*n*.

A pair of PMIMPDEF<*n*> registers at a doubleword-aligned offset might be defined as a single 64 bit register.

4.26 PMINTENCLR<*n*>, Interrupt Enable Clear Register <*n*>

The PMINTENCLR<*n*> characteristics are:

Purpose
Disable interrupt on monitor overflow status.

Usage constraints
PMINTENCLR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMINTENCLR<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMINTENCLR<*n*> is a 32-bit read/write memory-mapped register located at offset `0xC60` + 4×*n* in the Page 0 component.

4.26.1 Field descriptions

The PMINTENCLR<*n*> bit assignments are:

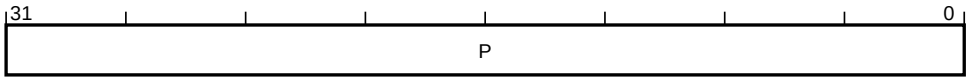


Figure 4.25: PMINTENCLR<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
Interrupt on overflow status of PMEVCNTR<m> disable. On writes, allows software to disable the interrupt on overflow status of PMEVCNTR<m>. On reads, returns the interrupt on overflow status of PMEVCNTR<m> enable status. The possible values of this bit are:

`0b0`	Interrupt on overflow status of PMEVCNTR<m> disabled.
`0b1`	Interrupt on overflow status of PMEVCNTR<m> enabled.

If the Cycle Counter extension is implemented, then PMINTENCLR<*n*>.P[31] allows software to disable the interrupt on overflow status of PMCCNTR and query the interrupt on overflow status of PMCCNTR enable status.

This bit is read/write-one-to-clear.

4.27 PMINTENSET<*n*>, Interrupt Enable Set Register <*n*>

The PMINTENSET<*n*> characteristics are:

Purpose
Enable interrupt on monitor overflow status.

Usage constraints
PMINTENSET<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMINTENSET<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMINTENSET<*n*> is a 32-bit read/write memory-mapped register located at offset `0xC40` + 4×*n* in the Page 0 component.

4.27.1 Field descriptions

The PMINTENSET<*n*> bit assignments are:

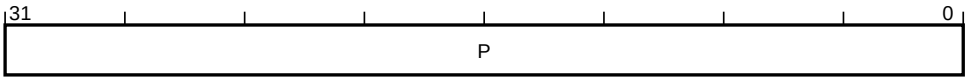


Figure 4.26: PMINTENSET<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
Interrupt on overflow status of PMEVCNTR<m> enable. On writes, allows software to enable the interrupt on overflow status of PMEVCNTR<m>. On reads, returns the interrupt on overflow status of PMEVCNTR<m> enable status. The possible values of this bit are:

`0b0`	Interrupt on overflow status of PMEVCNTR<m> disabled.
`0b1`	Interrupt on overflow status of PMEVCNTR<m> enabled.

If the Cycle Counter extension is implemented, then PMINTENSET<*n*>.P[31] allows software to enable the interrupt on overflow status of PMCCNTR and query the interrupt on overflow status of PMCCNTR enable status.

This bit is read/write-one-to-set.

4.28 PMIRQCR0, Interrupt Configuration Register 0

The PMIRQCR0 characteristics are:

Purpose
Interrupt configuration register.

Usage constraints
PMIRQCR0 reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

PMIRQCR0 ignores writes if all of the following are true:

- Any of the following are true:
  - PMSCR is not implemented and PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure.
  - PMSCR is implemented and PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure.
- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.

Permitted writes to PMIRQCR0 are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
PMIRQCR0 is present only if PMCFGR.MSI == `0b1`, message signaled interrupts are implemented. PMIRQCR0 is RES0 otherwise.

Attributes
PMIRQCR0 is a 64-bit read/write memory-mapped register located at offset `0xE80` in the Page 0 component.

4.28.1 Field descriptions

The PMIRQCR0 bit assignments are:

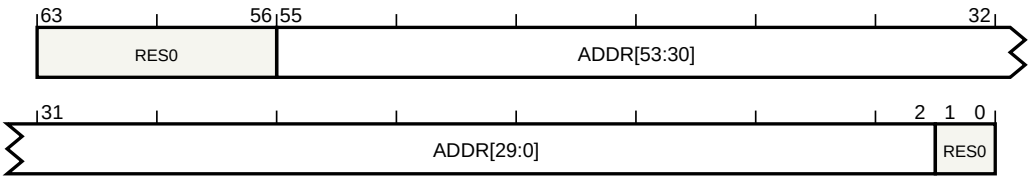


Figure 4.27: PMIRQCR0

Bits [63:56,1:0]
Reserved. This field is RES0.

ADDR, bits [55:2]
Message Signaled Interrupt address. (PMIRQCR0.ADDR << 2) is the address that the PMU writes to when signaling the Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the PMU is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

4.29 PMIRQCR1, Interrupt Configuration Register 1

The PMIRQCR1 characteristics are:

Purpose
Interrupt configuration register.

Usage constraints
PMIRQCR1 reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

PMIRQCR1 ignores writes if all of the following are true:

- Any of the following are true:
  - PMSCR is not implemented and PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure.
  - PMSCR is implemented and PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure.
- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.

Permitted writes to PMIRQCR1 are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
PMIRQCR1 is present only if PMCFGR.MSI == `0b1`, message signaled interrupts are implemented. PMIRQCR1 is RES0 otherwise.

Attributes
PMIRQCR1 is a 32-bit read/write memory-mapped register located at offset `0xE88` in the Page 0 component.

4.29.1 Field descriptions

The PMIRQCR1 bit assignments are:

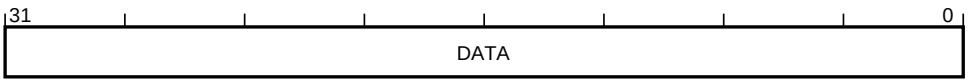


Figure 4.28: PMIRQCR1

DATA, bits [31:0]
Payload for the message signaled interrupt.

4.30 PMIRQCR2, Interrupt Configuration Register 2

The PMIRQCR2 characteristics are:

Purpose
Interrupt control and configuration register.

Usage constraints
PMIRQCR2 reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == 0b0.

PMIRQCR2 ignores writes if all of the following are true:

- Any of the following are true:
  - PMSCR is not implemented and PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure.
  - PMSCR is implemented and PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure.
- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.

Permitted writes to PMIRQCR2 are UNPREDICTABLE if PMCFGR.NA == 0b1 and PMCR.NA == 0b1.

Configurations
PMIRQCR2 is present only if PMCFGR.MSI == 0b1, message signaled interrupts are implemented. PMIRQCR2 is RES0 otherwise.

Attributes
PMIRQCR2 is a 32-bit read/write memory-mapped register located at offset 0xE8C in the Page 0 component.

4.30.1 Field descriptions

The PMIRQCR2 bit assignments are:

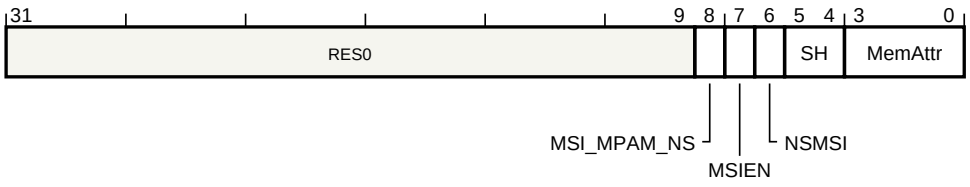


Figure 4.29: PMIRQCR2

Bits [31:9]
Reserved. This field is RES0.

MSI_MPAM_NS, bit [8]
Memory System Resource Partitioning and Monitoring (MPAM) Non-secure signaling for Secure MSIs.

When PMSCR is implemented, or PMIRQCR2.NSMSI configures MSIs as Non-secure MSIs
This bit is RES0.

When the PMU supports MPAM on message signaled interrupts
Defines the MPAM_NS setting used for Secure message signaled interrupts. The possible values of this bit are:

0b0	Secure MSIs are issued with MPAM_NS set to 0b0.
0b1	Secure MSIs are issued with MPAM_NS set to 0b1.

This bit resets to 0b0.

Otherwise
This bit is RES0.

MSIEN, bit [7]
Message signaled interrupt enable.

When the PMU supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

0b0	Disabled.
0b1	Enabled.

This bit resets to 0b0.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [6]
Non-secure message signaled interrupt.

When PMSCR is implemented
This bit is RES0.

When the PMU supports configuring the physical address space for message signaled interrupts
Defines the physical address space for message signaled interrupts. The possible values of this bit are:

0b0	Secure physical address space.
0b1	Non-secure physical address space.

Accessing this bit has the following behavior:

- This bit ignores writes if any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- Otherwise, this bit is read/write.

This bit resets to an IMPLEMENTATION DEFINED value.

Otherwise
The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [5:4]
Shareability.

When the PMU supports configuring the Shareability domain for message signaled interrupts
Defines the Shareability domain for message signaled interrupts. The possible values of this field are:

0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when PMIRQCR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

This field resets to an architecturally UNKNOWN value.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [3:0]
Memory type.

When the PMU supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

Note:

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

Otherwise
The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

4.31 PMIRQSR, Interrupt Status Register

The PMIRQSR characteristics are:

Purpose
Interrupt status register.

Usage constraints
PMIRQSR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

PMIRQSR ignores writes if all of the following are true:

- Any of the following are true:
  - PMSCR is not implemented and PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure.
  - PMSCR is implemented and PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure.
- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.

Configurations
PMIRQSR is present only if PMCFGR.MSI == `0b1`, message signaled interrupts are implemented. PMIRQSR is RES0 otherwise.

Attributes
PMIRQSR is a 64-bit read/write memory-mapped register located at offset `0xEF8` in the Page 0 component.

4.31.1 Field descriptions

The PMIRQSR bit assignments are:

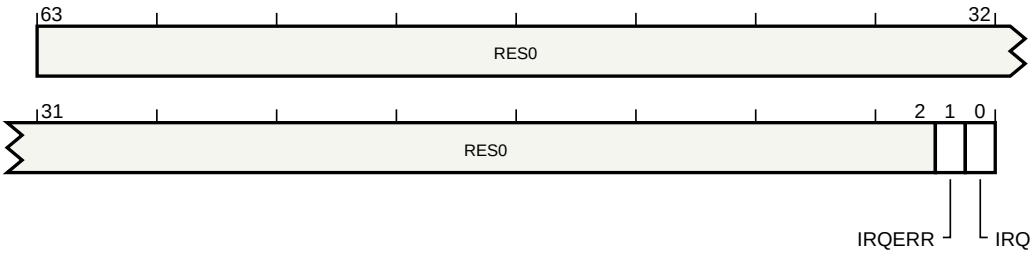


Figure 4.30: PMIRQSR

Bits [63:2]
Reserved. This field is RES0.

IRQERR, bit [1]
Interrupt Error. The possible values of this bit are:

`0b0`	Interrupt write has not returned an error since this bit was last cleared to zero.

0b1	Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

IRQ, bit [0]
PMU Overflow Interrupt write in progress. The defined values of this bit are:

0b0	PMU Overflow Interrupt write not in progress.
0b1	PMU Overflow Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

Note:

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual PMOVS<*n*> and PMINTEN<*n*> registers.

4.32 PMOVSCLR<*n*>, Overflow Flag Status Clear Register <*n*>

The PMOVSCLR<*n*> characteristics are:

Purpose
Clear PMU monitor overflow status flags.

Usage constraints
PMOVSCLR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMOVSCLR<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMOVSCLR<*n*> is a 32-bit read/write memory-mapped register located at offset `0xC80` + 4×*n* in the Page 1 component.

4.32.1 Field descriptions

The PMOVSCLR<*n*> bit assignments are:

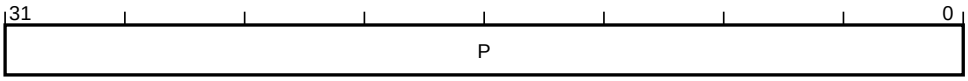


Figure 4.31: PMOVSCLR<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
Overflow status flag for PMEVCNTR<m> clear. On writes, allows software to clear the overflow status flag for PMEVCNTR<m> to 0. On reads, returns the overflow status flag for PMEVCNTR<m> overflow status. The possible values of this bit are:

`0b0`	PMEVCNTR<m> has not overflowed.
`0b1`	PMEVCNTR<m> has overflowed.

If the Cycle Counter extension is implemented, then PMOVSCLR<*n*>.P[31] allows software to clear the overflow status flag for PMCCNTR to 0 and query the overflow status flag for PMCCNTR overflow status.

This bit is read/write-one-to-clear.

4.33 PMOVSSET<*n*>, Overflow Flag Status Set Register <*n*>

The PMOVSSET<*n*> characteristics are:

Purpose
Set PMU monitor overflow status flags.

Usage constraints
PMOVSSET<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMOVSSET<*n*> are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMOVSSET<*n*> is a 32-bit read/write memory-mapped register located at offset `0xCC0` + 4×*n* in the Page 1 component.

4.33.1 Field descriptions

The PMOVSSET<*n*> bit assignments are:

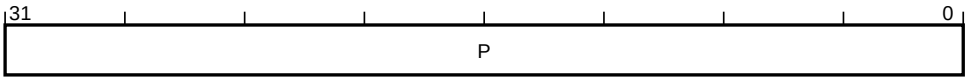


Figure 4.32: PMOVSSET<*n*>

P<*m*>, bit [*m*], for *m* = 0 to 31
Overflow status flag for PMEVCNTR<m> set. On writes, allows software to set the overflow status flag for PMEVCNTR<m> to 1. On reads, returns the overflow status flag for PMEVCNTR<m> overflow status. The possible values of this bit are:

`0b0`	PMEVCNTR<m> has not overflowed.
`0b1`	PMEVCNTR<m> has overflowed.

If the Cycle Counter extension is implemented, then PMOVSSET<*n*>.P[31] allows software to set the overflow status flag for PMCCNTR to 1 and query the overflow status flag for PMCCNTR overflow status.

This bit is read/write-one-to-set.

4.34 PMOVSSR<*n*>, Overflow Status Snapshot Register <*n*>

The PMOVSSR<*n*> characteristics are:

Purpose

Captured copy of PMOVS. Once captured, the value in PMOVSSR is unaffected by writes to PMOVSSET and PMOVSCLR.

Usage constraints

PMOVSSR<*n*> is one of the PMSVR<n> registers. The location of PMOVSSR<*n*> within the PMSVR<n> registers is IMPLEMENTATION DEFINED.

PMOVSSR<*n*> reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

PMOVSSR<*n*> is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMOVSSR<*n*> is RES0 otherwise.

PMOVSSR<*n*> is an optional one of the PMSVR<n> registers. If PMSSRR is implemented, Arm recommends that PMOVSSR<*n*> is implemented, as this indicates whether a counter that is reset has overflowed during the sampling period. If PMSSRR is not implemented, counters are free-running across samples without being reset and could overflow at any time, meaning there is less benefit from sampling PMOVS.

Attributes

PMOVSSR<*n*> is a 32-bit read-only memory-mapped register located at offset `0x600` + IMPLEMENTATION DEFINED + 4×*n* in the Page 1 component.

4.35 PMPIDR0, Peripheral Identification Register 0

The PMPIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR0 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR0 is present only if Peripheral Identification scheme is implemented. PMPIDR0 is RES0 otherwise.

Attributes
PMPIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFE0`.

4.35.1 Field descriptions

The PMPIDR0 bit assignments are:

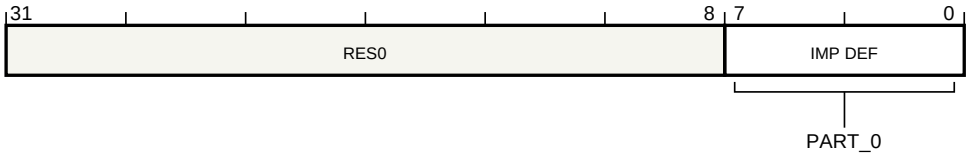


Figure 4.33: PMPIDR0

Bits [31:8]
Reserved. This field is RES0.

PART_0, bits [7:0]
Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 8 bits, PMPIDR2.REVISION and PMPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

4.36 PMPIDR1, Peripheral Identification Register 1

The PMPIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR1 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR1 is present only if Peripheral Identification scheme is implemented. PMPIDR1 is RES0 otherwise.

Attributes
PMPIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFE4`.

4.36.1 Field descriptions

The PMPIDR1 bit assignments are:

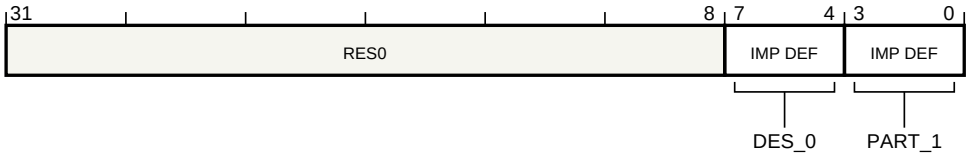


Figure 4.34: PMPIDR1

Bits [31:8]
Reserved. This field is RES0.

DES_0, bits [7:4]
Designer, JEP106 identification code, bits [3:0]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

PART_1, bits [3:0]
Part number, bits [11:8].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 8 bits, PMPIDR2.REVISION and PMPIDR3.REVAND, available to define the revision of the component.

- If a 16-bit part number is used, then it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

4.37 PMPIDR2, Peripheral Identification Register 2

The PMPIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR2 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR2 is present only if Peripheral Identification scheme is implemented. PMPIDR2 is RES0 otherwise.

Attributes
PMPIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFE8`.

4.37.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

Figure 4.35: PMPIDR2 the component uses a 12-bit part number

Bits [31:8]
Reserved. This field is RES0.

REVISION, bits [7:4]
Component major revision. PMPIDR2.REVISION and PMPIDR3.REVAND together form the revision number of the component, with PMPIDR2.REVISION being the most significant part and PMPIDR3.REVAND the least significant part. When a component is changed, PMPIDR2.REVISION or PMPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. PMPIDR3.REVAND should be set to `0b0000` when PMPIDR2.REVISION is increased.

This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]
JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

DES_1, bits [2:0]
Designer, JEP106 identification code, bits [6:4]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which

might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

4.37.2 The component uses a 16-bit part number

Configurations
Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

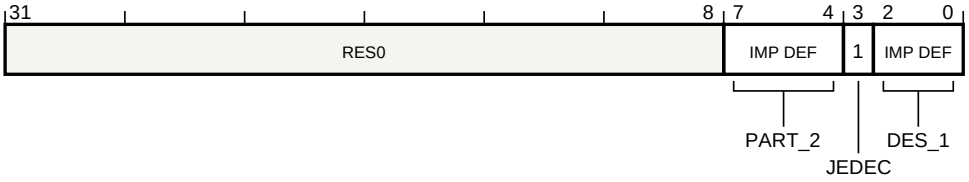


Figure 4.36: PMPIDR2 the component uses a 16-bit part number

Bits [31:8]
Reserved. This field is RES0.

PART_2, bits [7:4]
Part number, bits [15:12].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 8 bits, PMPIDR2.REVISION and PMPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]
JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

DES_1, bits [2:0]
Designer, JEP106 identification code, bits [6:4]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

4.38 PMPIDR3, Peripheral Identification Register 3

The PMPIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR3 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR3 is present only if Peripheral Identification scheme is implemented. PMPIDR3 is RES0 otherwise.

Attributes
PMPIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFEC`.

4.38.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

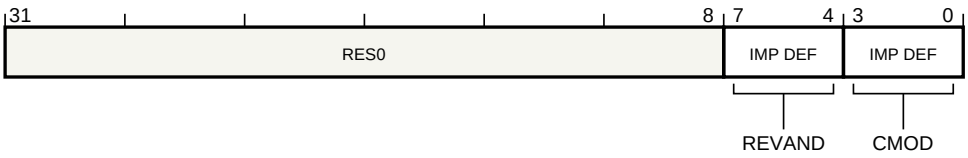


Figure 4.37: PMPIDR3 the component uses a 12-bit part number

Bits [31:8]
Reserved. This field is RES0.

REVAND, bits [7:4]
Component minor revision. PMPIDR2.REVISION and PMPIDR3.REVAND together form the revision number of the component, with PMPIDR2.REVISION being the most significant part and PMPIDR3.REVAND the least significant part. When a component is changed, PMPIDR2.REVISION or PMPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. PMPIDR3.REVAND should be set to `0b0000` when PMPIDR2.REVISION is increased.

This field reads as an IMPLEMENTATION DEFINED value.

CMOD, bits [3:0]
Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components is zero then the components are identical.
- If the CMOD fields of both components have the same nonzero value then this does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is nonzero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

4.38.2 The component uses a 16-bit part number

Configurations
Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

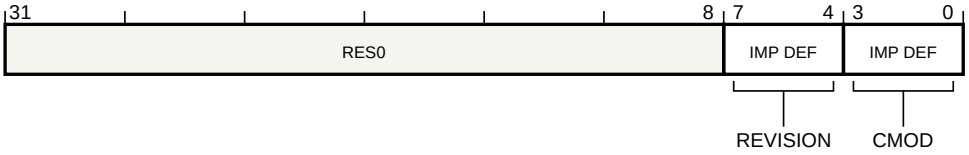


Figure 4.38: PMPIDR3 the component uses a 16-bit part number

Bits [31:8]
Reserved. This field is RES0.

REVISION, bits [7:4]
Component revision. When a component is changed, PMPIDR3.REVISION is increased to ensure that software can differentiate the different revisions of the component.

This field reads as an IMPLEMENTATION DEFINED value.

CMOD, bits [3:0]
Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components is zero then the components are identical.
- If the CMOD fields of both components have the same nonzero value then this does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is nonzero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

4.39 PMPIDR4, Peripheral Identification Register 4

The PMPIDR4 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR4 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR4 is present only if Peripheral Identification scheme is implemented. PMPIDR4 is RES0 otherwise.

Attributes
PMPIDR4 is a 32-bit read-only memory-mapped register located at offset `0xFD0`.

4.39.1 Field descriptions

The PMPIDR4 bit assignments are:

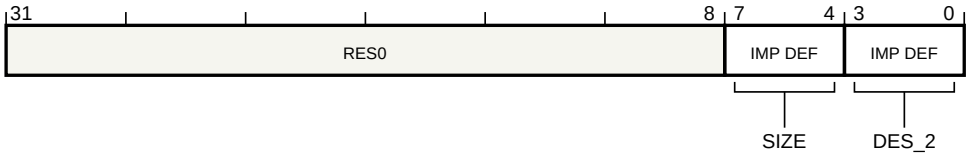


Figure 4.39: PMPIDR4

Bits [31:8]
Reserved. This field is RES0.

SIZE, bits [7:4]
Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of `0b0000` means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{PMPIDR4.SIZE}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field reads as an IMPLEMENTATION DEFINED value.

DES_2, bits [3:0]
Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not

the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

4.40 PMPIDR5, Peripheral Identification Register 5

The PMPIDR5 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR5 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR5 is present only if Peripheral Identification scheme is implemented. PMPIDR5 is RES0 otherwise.

Attributes
PMPIDR5 is a 32-bit read-only memory-mapped register located at offset `0xFD4`.

4.40.1 Field descriptions

The PMPIDR5 bit assignments are:

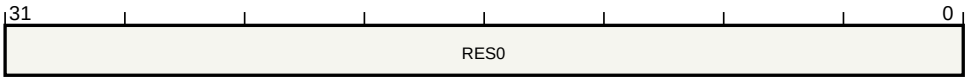


Figure 4.40: PMPIDR5

Bits [31:0]
This field is RES0.

4.41 PMPIDR6, Peripheral Identification Register 6

The PMPIDR6 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR6 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR6 is present only if Peripheral Identification scheme is implemented. PMPIDR6 is RES0 otherwise.

Attributes
PMPIDR6 is a 32-bit read-only memory-mapped register located at offset `0xFD8`.

4.41.1 Field descriptions

The PMPIDR6 bit assignments are:

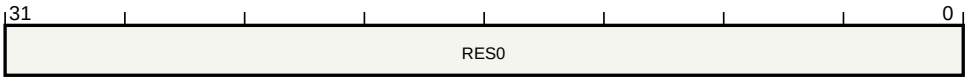


Figure 4.41: PMPIDR6

Bits [31:0]
This field is RES0.

4.42 PMPIDR7, Peripheral Identification Register 7

The PMPIDR7 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
PMPIDR7 reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMPIDR7 is present only if Peripheral Identification scheme is implemented. PMPIDR7 is RES0 otherwise.

Attributes
PMPIDR7 is a 32-bit read-only memory-mapped register located at offset `0xFDC`.

4.42.1 Field descriptions

The PMPIDR7 bit assignments are:

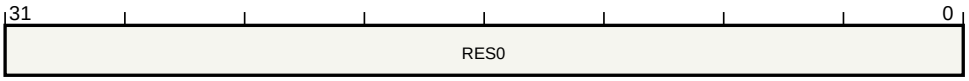


Figure 4.42: PMPIDR7

Bits [31:0]
This field is RES0.

4.43 PMROOTCR, Root and Realm Control Register

The PMROOTCR characteristics are:

Purpose
Controls observability of Root and Realm events by the performance monitor.

Usage constraints
PMROOTCR reads-as-zero and ignores writes if the access is not Root.

Configurations
It is IMPLEMENTATION DEFINED whether PMROOTCR is present if Root and Realm states are implemented. PMROOTCR is not present otherwise. PMROOTCR is RES0 if not present.

Attributes
PMROOTCR is a 64-bit read/write memory-mapped register located at offset `0xE48` in the Page 0 component.

PMROOTCR might be implemented at an IMPLEMENTATION DEFINED offset in a component other than the PMU Page 0 and Page 1 components.

4.43.1 Field descriptions

The PMROOTCR bit assignments are:

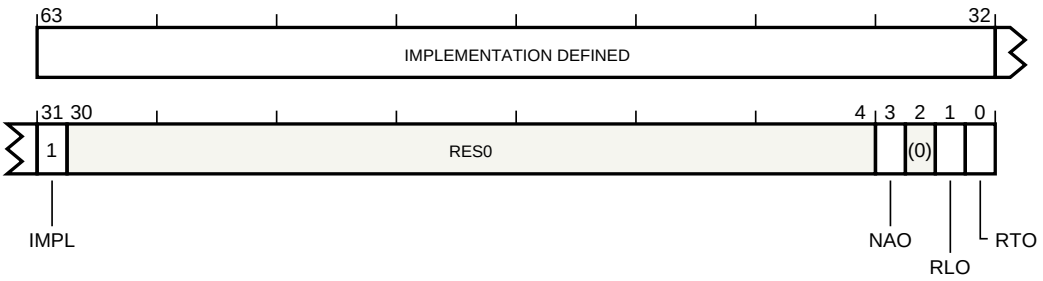


Figure 4.43: PMROOTCR

Bits [63:32]
IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

IMPL, bit [31]
Indicates PMROOTCR is present. This bit reads-as-one.

Bits [30:4,2]
Reserved. This field is RES0.

NAO, bit [3]
Non-attributable Observation.

When the PMU can count or monitor non-attributable events
Controls whether events or monitorable characteristics not attributable with any source can be monitored. The possible values of this bit are:

`0b0`	Events not attributable with any event source are not counted.

0b1	Counting non-attributable events is not prevented by this bit.

When both PMROOTCR and PMSCR are implemented, non-attributable events are counted only if both PMROOTCR.NAO is 0b1 and PMSCR.{NAO, SO} is nonzero.

PMROOTCR.NAO has the opposite reset polarity to PMSCR.NAO.

This bit resets to 0b1.

Otherwise
Reserved. This bit is RES0.

RLO, bit [1]
Realm Observation. Controls whether events or monitorable characteristics attributable to a Realm event source can be monitored. The possible values of this bit are:

0b0	Events attributable to a Realm event source are not counted.
0b1	Events attributable to a Realm event source are counted.

This bit resets to 0b0.

RTO, bit [0]
Root Observation. Controls whether events or monitorable characteristics attributable to a Root event source can be monitored. The possible values of this bit are:

0b0	Events attributable to a Root event source are not counted.
0b1	Events attributable to a Root event source are counted.

This bit resets to 0b0.

4.44 PMSCR, Secure Control Register

The PMSCR characteristics are:

Purpose

Controls observability of Secure events by the performance monitor, and optionally controls Secure attributes for message signaled interrupts and Non-secure access to the performance monitor registers.

Usage constraints

A PMU might include further IMPLEMENTATION DEFINED access controls that can either:

- Limit access to this register to only Root accesses. Such controls must only be accessible to Root accesses.
- Enable access to this register to Non-secure accesses, Realm accesses, or both. Such controls must only be accessible to Secure or Root accesses.

Any such controls are outside the scope of the CoreSight Performance Monitoring Unit architecture.

PMSCR reads-as-zero and ignores writes if any of the following are true:

- The access is Non-secure.
- The access is Realm.

Configurations

It is IMPLEMENTATION DEFINED whether PMSCR is present if Secure state is implemented. PMSCR is not present otherwise. PMSCR is RES0 if not present.

If PMSCR.NSMSI is implemented, the corresponding field in PMIRQCR2 is RES0.

If PMSCR.MSI_MPAM_NS is implemented, the corresponding field in PMIRQCR2 is RES0.

Attributes

PMSCR is a 64-bit read/write memory-mapped register located at offset `0xE40` in the Page 0 component.

PMSCR might be implemented at an IMPLEMENTATION DEFINED offset in a component other than the PMU Page 0 and Page 1 components.

4.44.1 Field descriptions

The PMSCR bit assignments are:

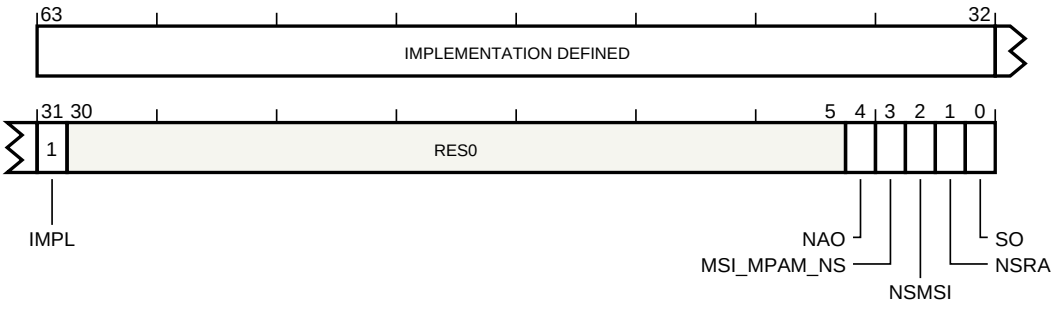


Figure 4.44: PMSCR

Bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

IMPL, bit [31]
Indicates PMSCR is present. This bit reads-as-one.

Bits [30:5]
Reserved. This field is RES0.

NAO, bit [4]
Non-attributable Observation.

When the PMU can count or monitor non-attributable events
Controls whether events or monitorable characteristics not attributable with any source can be monitored. The possible values of this bit are:

0b0	Events not attributable with any event source are not counted, unless overridden by PMSCR.SO.
0b1	Counting non-attributable events is not prevented by this bit.

When both PMROOTCR and PMSCR are implemented, non-attributable events are counted only if both PMROOTCR.NAO is 0b1 and PMSCR.{NAO, SO} is nonzero.

PMSCR.NAO has the opposite reset polarity to PMROOTCR.NAO.

This bit is optional if Root and Realm states are not implemented. When this bit is not implemented, the PMU behaves as if PMSCR.NAO is 0b0, and whether events or monitorable characteristics not attributable with any source can be monitored is controlled by PMSCR.SO.

This bit resets to 0b0.

Otherwise
Reserved. This bit is RES0.

MSI_MPAM_NS, bit [3]
MPAM Non-secure signaling for Secure MSIs.

When PMCFGR.MSI == 0b0, message-signaled interrupts are not implemented, or PMSCR.NSMSI configures MSIs as Non-secure MSIs
This bit is RES0.

When the PMU supports MPAM on message signaled interrupts
Defines the MPAM_NS setting used for Secure message signaled interrupts. The possible values of this bit are:

0b0	Secure MSIs are issued with MPAM_NS set to 0b0.
0b1	Secure MSIs are issued with MPAM_NS set to 0b1.

This bit resets to 0b0.

Otherwise
This bit is RES0.

NSMSI, bit [2]
Non-secure message signaled interrupt.

When PMCFGR.MSI == 0b0, message-signaled interrupts are not implemented
This bit is RES0.

When PMSCR.NSRA == 0b1

The physical address space used for message signaled interrupts is Non-secure.

This bit is RES0.

Otherwise

Defines the physical address space for message signaled interrupts. The possible values of this bit are:

0b0	Secure physical address space.
0b1	Non-secure physical address space.

This bit resets to 0b1.

NSRA, bit [1]

Non-secure and Realm register access.

When the PMU allows configuration of Non-secure and Realm Register Accesses

The possible values of this bit are:

0b0	Non-secure and Realm Register Access is disabled. Non-secure and Realm access to any PMU register is RAZ/WI.
0b1	Non-secure and Realm Register Access is enabled. If the PMU supports MSIs, generated MSIs are always Non-secure.

Note:

Non-secure and Realm Register Access is enabled from reset.

This bit resets to 0b1.

Otherwise

Reserved. This bit is RES0.

SO, bit [0]

Secure Observation. Controls whether events or monitorable characteristics attributable to a Secure event source can be monitored. The possible values of this bit are:

0b0	Events attributable to a Secure event source are not counted.
0b1	Events attributable to a Secure event source are counted.

Also controls whether events or monitorable characteristics not attributable with any source can be monitored. See PMSCR.NAO.

This bit resets to 0b0.

4.45 PMSSCR, Snapshot Capture Register (PMSSSR implemented)

The PMSSCR characteristics are:

Purpose
Provides a mechanism for software to initiate a sample.

Usage constraints
PMSSCR ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMSSCR is present only if all of the following are true:

- PMSSSR is implemented.
- PMCFGR.SS == `0b1`, snapshot extension is implemented.

PMSSCR is RES0 otherwise.

In some Arm PE implementations where the snapshot feature is an IMPLEMENTATION DEFINED extension, PMSSCR is located at offset `0x6F0`.

Attributes
PMSSCR is a 32-bit write-only memory-mapped register located at offset `0xE30` in the Page 0 component.

4.45.1 Field descriptions

The PMSSCR bit assignments are:

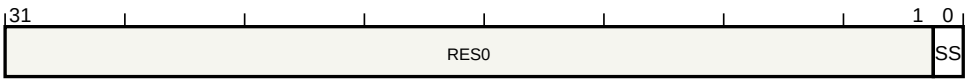


Figure 4.45: PMSSCR

Bits [31:1]
Reserved. This field is RES0.

SS, bit [0]
Capture now. The possible values for writing to this bit are:

`0b0`	Ignored.
`0b1`	Initiate a capture immediately.

4.46 PMSSCR, Snapshot Control Register (PMSSSR not implemented)

The PMSSCR characteristics are:

Purpose

Provides a mechanism for software to initiate a sample, and holds status information about the captured monitors.

Usage constraints

PMSSCR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == 0b0.

Configurations

PMSSCR is present only if all of the following are true:

- PMCFGR.SS == 0b1, snapshot extension is implemented.
- PMSSSR is not implemented.

PMSSCR is RES0 otherwise.

In some configurations, PMSSCR[64:32] is replaced by a Snapshot Status Register, PMSSSR. In such an implementation, PMSSSR is one of the PMSVR<n> registers and PMSSCR is a 32-bit register located at offset 0xE30 in the Page 0 component.

Attributes

PMSSCR is a 64-bit read/write memory-mapped register located at offset 0xE30 in the Page 0 component.

4.46.1 Field descriptions

The PMSSCR bit assignments are:

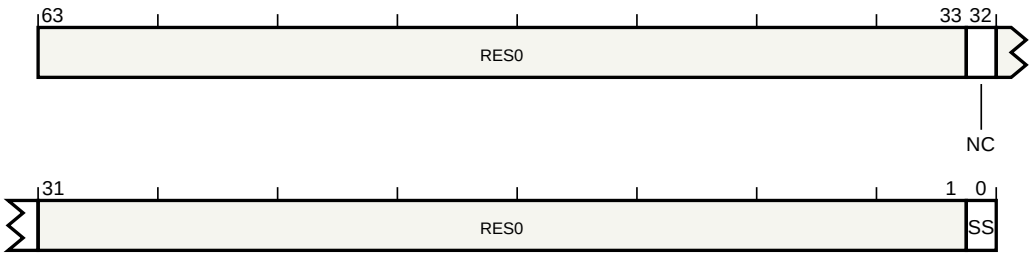


Figure 4.46: PMSSCR

Bits [63:33,31:1]

Reserved. This field is RES0.

NC, bit [32]

No capture. Indicates whether the PMU monitors have been captured. The possible values of this bit are:

0b0	PMU monitors captured.
0b1	PMU monitors not captured.

The monitors are only not captured by the PMU if there is a security violation. The consumer of the captured data is responsible for keeping track of whether it managed to read the snapshot registers from the PMU.

PMSSCR.NC is reset to `0b1` by reset, but is overwritten at the first capture. Tools need to be aware that capturing over reset or power-down might lose data, and rely on software saving and restoring the PMU state.

This bit resets to `0b1`.

SS, bit [0]

Capture now. The possible values of this bit are:

`0b0`	Ignored.
`0b1`	Initiate a capture immediately.

4.47 PMSSRR, Snapshot Reset Register

The PMSSRR characteristics are:

Purpose
Configure Snapshot to reset monitors after each sample is taken.

Usage constraints
PMSSRR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
It is IMPLEMENTATION DEFINED whether PMSSRR is present if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSRR is not present otherwise. PMSSRR is RES0 if not present.

Support for the capability to reset counters after each sample is taken is optional. Arm recommends this feature is not implemented where the PMU might also be used in a non-snapshot mode, for example, in a PE.

If the capability is not implemented and the snapshot feature is implemented, this register is RAZ/WI.

In some Arm PE implementations where the snapshot feature is an IMPLEMENTATION DEFINED extension, PMSSRR is located at offset `0x6F4`.

Attributes
PMSSRR is a 64-bit read/write memory-mapped register located at offset `0xE38` in the Page 0 component.

4.47.1 Field descriptions

The PMSSRR bit assignments are:

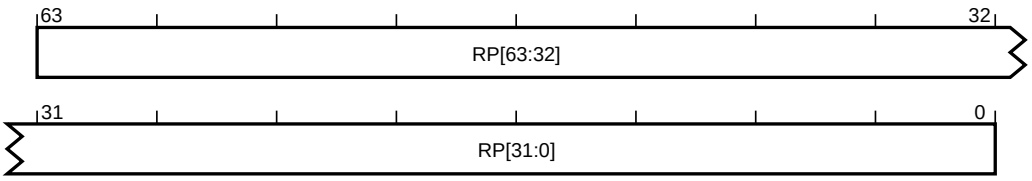


Figure 4.47: PMSSRR

RP<*m*>, bit [*m*], for *m* = 0 to 63
Reset monitor. If *m* >= PMCFGR.N, the number of implemented monitors, then RP[*m*] is RAZ/WI. Otherwise, indicates whether PMEVCNTR<m> and PMOVS[*m*] are to be reset after a capture. The possible values of this bit are:

`0b0`	Do not reset PMEVCNTR<m> and PMOVS[*m*] on capture.
`0b1`	Reset PMEVCNTR<m> and PMOVS[*m*] to zero on capture.

If the Cycle Counter extension is implemented, then PMSSRR.RP[31] controls reset of the cycle counter.

4.48 PMSSSR, Snapshot Status Register

The PMSSSR characteristics are:

Purpose
Holds status information about the captured monitors.

Usage constraints
The location of PMSSSR within the PMSVR<n> registers is IMPLEMENTATION DEFINED. Arm recommends that PMSSSR is read after the other PMSVR<n> registers, meaning it might be placed last in the PMSVR<n> registers.

PMSSSR reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
It is IMPLEMENTATION DEFINED whether PMSSSR is present if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSSR is not present otherwise. PMSSSR is RES0 if not present.

Attributes
PMSSSR is a 32-bit read-only memory-mapped register located at offset `0x600` + IMPLEMENTATION DEFINED in the Page 1 component.

4.48.1 Field descriptions

The PMSSSR bit assignments are:

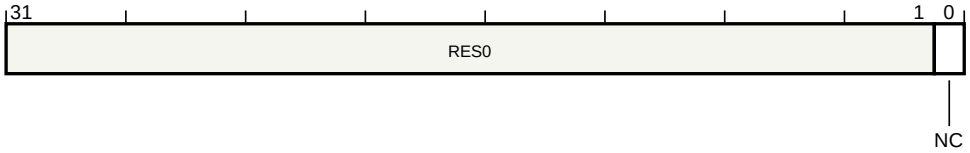


Figure 4.48: PMSSSR

Bits [31:1]
Reserved. This field is RES0.

NC, bit [0]
No capture. Indicates whether the PMU monitors have been captured. The defined values of this bit are:

`0b0`	PMU monitors captured.
`0b1`	PMU monitors not captured.

The monitors are only not captured by the PMU if there is a security violation. The consumer of the captured data is responsible for keeping track of whether it managed to read the snapshot registers from the PMU.

PMSSSR.NC is reset to `0b1` by reset, but is overwritten at the first capture. Tools need to be aware that capturing over reset or power-down might lose data, and rely on software saving and restoring the PMU state.

This bit resets to `0b1`.

4.49 PMSVR<*n*>, Saved Value Register <0-127>

The PMSVR<0-127> characteristics are:

Purpose

Captured copy of performance monitor registers. The content of these registers is IMPLEMENTATION DEFINED. These registers contain all the captured state of the PMU, including:

- The event counters PMEVCNTR<n>, including PMCCNTR, if implemented.
- The overflow status flags, captured in PMOVSSR<n>, if PMSSRR is implemented.
- PMSSSR, if implemented.
- Any additional IMPLEMENTATION DEFINED syndrome information captured by the PMU.

Once captured, the values in these registers are unaffected by direct or indirect writes to PMCR or any of the captured registers.

Usage constraints

PMSVR<*n*> reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

PMSVR<*n*> is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSVR<*n*> is RES0 otherwise.

Attributes

PMSVR<*n*> is a 32-bit read-only memory-mapped register located at offset `0x600` + 4×*n* in the Page 1 component.

A pair of PMSVR<*n*> registers at a doubleword-aligned offset might be defined as a single 64 bit register.

Chapter 5
Programmers' Model, 64-bit programmers' model extension implemented

This section is *normative*.

R_{JSQDQ} When the direct access to enables and overflows extension and 64-bit programmers' model extension are implemented, the *Performance Monitoring Unit* (PMU) register map is as described by the following sections.

The register descriptions in this section describe the differences in location, layout, and content of the register when the 3.13 *64-bit programmers' model extension* is implemented. For other details, see Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented*.

See also:

- 2.5 *Accessing registers*.
- 3.12 *Direct access to enables and overflows extension*.
- 3.13 *64-bit programmers' model extension*.
- Chapter 4 *Programmers' Model, 64-bit programmers' model extension not implemented*.

5.1 Memory-mapped registers

R_ZJZYP The values in the *Version* columns of the register indices refer to the following extensions:

CC 3.3 *Fixed-function cycle counter extension*.
CS CoreSight registers.
MG
3.4 *Monitor group extension*.
MSI
Message-signaled interrupts. See R_FDLKQ.
RME
Realm Management Extensions. The component incorporating the PMU implements Root and Realm operating states. See 2.6 *Security* and 3.14 *Observability and access control extension*.
SS snapshot extension.
TZ Security extensions. The component incorporating the PMU implements a Secure operating state. See 2.6 *Security* and 3.14 *Observability and access control extension*.

5.1.1 When the dual-page extension is not implemented

Table 5.1: Memory-mapped register map

Offset	Access	Size	Version	Register	Description
0x000+8×*n*	R/W	64		PMEVCNTR<*n*>	Event Count Register <*n*>
0x0F8	R/W	64	CC	PMCCNTR	Cycle Count Register
0x400+8×*n*	RO or R/W	64		PMEVTYPER<*n*>	Event Type Select Register <*n*>
0x4F8	RO or R/W	64	CC	PMCCFILTR	Cycle Counter Filter Register
0x600+8×*n*	RO	64	SS	PMSVR<*n*>	Saved Value Register <*n*>
0x600+IMPDEF	RO	64	SS	PMOVSSR<*n*>	Overflow Status Snapshot Register
0x800+8×*n*	RO or R/W	64		PMEVFILT2R<*n*>	Event Filter 2 Register <*n*>
0xA00+8×*n*	RO or R/W	64		PMEVFILTR<*n*>	Event Filter Register <*n*>
0xC00	R/W	64		PMCNTENSET	Count Enable Set Register
0xC10	R/W	64		PMCNTEN	Count Enable Register
0xC20	R/W	64		PMCNTENCLR	Count Enable Clear Register
0xC40	R/W	64		PMINTENSET	Interrupt Enable Set Register
0xC50	R/W	64		PMINTEN	Interrupt Enable Register
0xC60	R/W	64		PMINTENCLR	Interrupt Enable Clear Register
0xC80	R/W	64		PMOVSCLR	Overflow Flag Status Clear Register
0xC90	R/W	64		PMOVS	Overflow Flag Status Register
0xCC0	R/W	64		PMOVSSET	Overflow Flag Status Set Register
0xCE0+8×*n*	RO	64	MG	PMCGCR<*n*>	Counter Group Configuration Register <*n*>
0xD80+8×*n*	R/W	64		PMIMPDEF<*n*>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO	64		PMCFGR	Configuration Register
0xE08	RO	64		PMIIDR	Implementation Identification Register
0xE10	R/W	64		PMCR	Control Register

Offset	Access	Size	Version	Register	Description
0xE30	R/W	64	SS	PMSSCR	Snapshot Control Register
0xE38	R/W	64	SS	PMSSRR	Snapshot Reset Register
0xE40	R/W	64	TZ	PMSCR	Secure Control Register
0xE48	R/W	64	RME	PMROOTCR	Root and Realm Control Register
0xE80	R/W	64	MSI	PMIRQCR0	Interrupt Configuration Register 0
0xE88	R/W	64	MSI	PMIRQCR12	Interrupt Configuration Register 12
0xEF8	R/W	64	MSI	PMIRQSR	Interrupt Status Register
0xFA8	RO	64	CS	PMDEVAFF	Device Affinity Register
0xFB8	RO	32	CS	PMAUTHSTATUS	Authentication Status Register
0xFBC	RO	32	CS	PMDEVARCH	Device Architecture Register
0xFC8	RO	32	CS	PMDEVID	Device Configuration Register
0xFCC	RO	32	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	32	CS	PMPIDR4	Peripheral Identification Register 4
0xFD4	RO	32	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	32	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	32	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	32	CS	PMPIDR0	Peripheral Identification Register 0
0xFE4	RO	32	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	32	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	32	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	32	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	32	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	32	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	32	CS	PMCIDR3	Component Identification Register 3

5.1.2 Page 0, when the dual-page extension is implemented

Table 5.2: Memory-mapped register map

Offset	Access	Size	Version	Register	Description
0x400+8×*n*	RO or R/W	64		PMEVTYPER<*n*>	Event Type Select Register <*n*>
0x4F8	RO or R/W	64	CC	PMCCFILTR	Cycle Counter Filter Register
0x800+8×*n*	RO or R/W	64		PMEVFILT2R<*n*>	Event Filter 2 Register <*n*>
0xA00+8×*n*	RO or R/W	64		PMEVFILTR<*n*>	Event Filter Register <*n*>
0xC00	R/W	64		PMCNTENSET	Count Enable Set Register
0xC10	R/W	64		PMCNTEN	Count Enable Register
0xC20	R/W	64		PMCNTENCLR	Count Enable Clear Register
0xC40	R/W	64		PMINTENSET	Interrupt Enable Set Register
0xC50	R/W	64		PMINTEN	Interrupt Enable Register
0xC60	R/W	64		PMINTENCLR	Interrupt Enable Clear Register
0xCE0+8×*n*	RO	64	MG	PMCGCR<*n*>	Counter Group Configuration Register <*n*>
0xD80+8×*n*	R/W	64		PMIMPDEF<*n*>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO	64		PMCFGR	Configuration Register

Offset	Access	Size	Version	Register	Description
0xE08	RO	64		PMIIDR	Implementation Identification Register
0xE10	R/W	64		PMCR	Control Register
0xE30	R/W	64	SS	PMSSCR	Snapshot Control Register
0xE38	R/W	64	SS	PMSSRR	Snapshot Reset Register
0xE40	R/W	64	TZ	PMSCR	Secure Control Register
0xE48	R/W	64	RME	PMROOTCR	Root and Realm Control Register
0xE80	R/W	64	MSI	PMIRQCR0	Interrupt Configuration Register 0
0xE88	R/W	64	MSI	PMIRQCR12	Interrupt Configuration Register 12
0xEF8	R/W	64	MSI	PMIRQSR	Interrupt Status Register
0xFA8	RO	64	CS	PMDEVAFF	Device Affinity Register
0xFB8	RO	32	CS	PMAUTHSTATUS	Authentication Status Register
0xFBC	RO	32	CS	PMDEVARCH	Device Architecture Register
0xFC8	RO	32	CS	PMDEVID	Device Configuration Register
0xFCC	RO	32	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	32	CS	PMPIDR4	Peripheral Identification Register 4
0xFD4	RO	32	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	32	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	32	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	32	CS	PMPIDR0	Peripheral Identification Register 0
0xFE4	RO	32	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	32	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	32	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	32	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	32	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	32	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	32	CS	PMCIDR3	Component Identification Register 3

5.1.3 Page 1, when the dual-page extension is implemented

Table 5.3: Memory-mapped register map

Offset	Access	Size	Version	Register	Description
0x000+8×*n*	R/W	64		PMEVCNTR<*n*>	Event Count Register <*n*>
0x0F8	R/W	64	CC	PMCCNTR	Cycle Count Register
0x600+8×*n*	RO	64	SS	PMSVR<*n*>	Saved Value Register <*n*>
0x600+IMP DEF	RO	64	SS	PMOVSSR<*n*>	Overflow Status Snapshot Register
0xC80	R/W	64		PMOVSCLR	Overflow Flag Status Clear Register
0xC90	R/W	64		PMOVS	Overflow Flag Status Register
0xCC0	R/W	64		PMOVSSET	Overflow Flag Status Set Register
0xD80+8×*n*	R/W	64		PMIMPDEF<*n*>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO	64		PMCFGR	Configuration Register
0xE08	RO	64		PMIIDR	Implementation Identification Register
0xFA8	RO	64	CS	PMDEVAFF	Device Affinity Register
0xFBC	RO	32	CS	PMDEVARCH	Device Architecture Register
0xFCC	RO	32	CS	PMDEVTYPE	Device Type Register
0xFD0	RO	32	CS	PMPIDR4	Peripheral Identification Register 4

Offset	Access	Size	Version	Register	Description
0xFD4	RO	32	CS	PMPIDR5	Peripheral Identification Register 5
0xFD8	RO	32	CS	PMPIDR6	Peripheral Identification Register 6
0xFDC	RO	32	CS	PMPIDR7	Peripheral Identification Register 7
0xFE0	RO	32	CS	PMPIDR0	Peripheral Identification Register 0
0xFE4	RO	32	CS	PMPIDR1	Peripheral Identification Register 1
0xFE8	RO	32	CS	PMPIDR2	Peripheral Identification Register 2
0xFEC	RO	32	CS	PMPIDR3	Peripheral Identification Register 3
0xFF0	RO	32	CS	PMCIDR0	Component Identification Register 0
0xFF4	RO	32	CS	PMCIDR1	Component Identification Register 1
0xFF8	RO	32	CS	PMCIDR2	Component Identification Register 2
0xFFC	RO	32	CS	PMCIDR3	Component Identification Register 3

5.2 PMCCFILTR, Cycle Counter Filter Register

The PMCCFILTR characteristics are:

Purpose
Configures the Cycle Counter.

Usage constraints
PMCCFILTR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMCCFILTR are UNPREDICTABLE if PMCCFILTR is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations
PMCCFILTR is present only if PMCFGR.CC == `0b1`, cycle counter extension is implemented. PMCCFILTR is RES0 otherwise.

Attributes
PMCCFILTR is a 64-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x4F8` in the Page 0 component.

5.2.1 Field descriptions

The PMCCFILTR bit assignments are:

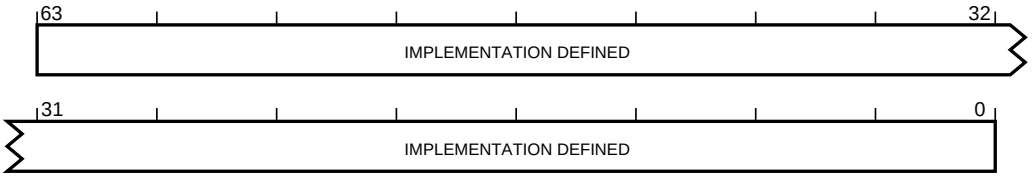


Figure 5.1: PMCCFILTR

All fields are unchanged.

5.3 PMCFGR, Configuration Register

The PMCFGR characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
PMCFGR reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCFGR is a 64-bit read-only memory-mapped register located at offset `0xE00`.

5.3.1 Field descriptions

The PMCFGR bit assignments are:

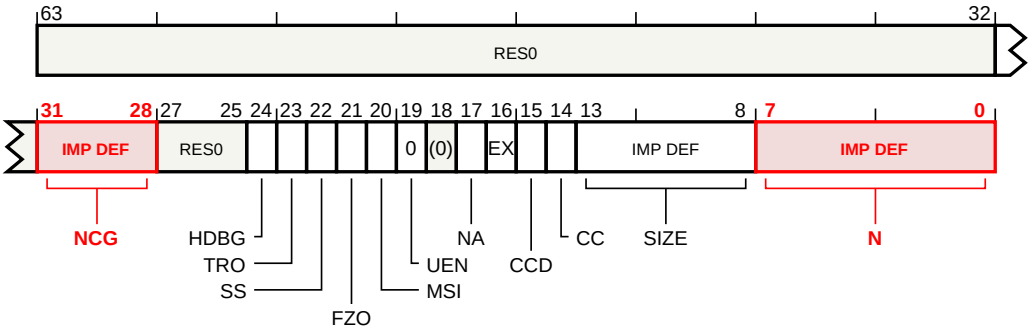


Figure 5.2: PMCFGR

NCG, bits [31:28]
Monitor Groups.

Defines the number of monitor groups implemented, minus one. If this field is zero, then one monitor group is implemented and PMCGCR<*n*> are not implemented.

Otherwise, for each monitor group <*m*>, PMCGCR<*m* DIV 8>.N<*m* MOD 8> defines the number of monitors in the group.

Locating the first monitor in each group depends on the number of implemented groups. Each monitor group starts with monitor:

- PMEVTYPER<*m*×32>, meaning there are at most 32 monitors per group, if there are 2 monitor groups.
- PMEVTYPER<*m*×16>, meaning there are at most 16 monitors per group, if there are 3 or 4 monitor groups.
- PMEVTYPER<*m*×8>, meaning there are at most 8 monitors per group, if there are between 5 and 8 monitor groups.

- PMEVTYPER<*m*×4>, meaning there are at most 4 monitors per group, if there are more than 8 monitor groups.

This field reads as an IMPLEMENTATION DEFINED value.

N, bits [7:0]

Number of monitors, minus one. The defined values of this field are:

`0x00`	1 monitor.
`0x01`	2 monitors.
..	..
`0x3F`	64 monitors.

All other values are reserved.

If PMCFGR.CC == `0b1`, PMEVCNTR31, the cycle counter, is one of the (N+1) monitors. For example, if PMCFGR.N == `0x00` and PMCFGR.CC == `0b1`, there is a single monitor, PMEVCNTR31, and PMEVCNTR0 is not implemented.

If PMCFGR.NCG != `0b0000`, then PMCFGR.N is the *total* number of monitors implemented.

This field reads as an IMPLEMENTATION DEFINED value.

Other fields are unchanged.

5.4 PMCGCR<*n*>, Counter Group Configuration Register <0-1>

The PMCGCR<0-1> characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
PMCGCR<*n*> reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
PMCGCR<*n*> is present only if PMCFGR.NCG != `0x0`, monitor groups are implemented. PMCGCR<*n*> is RES0 otherwise.

Attributes
PMCGCR<*n*> is a 64-bit read-only memory-mapped register located at offset `0xCE0` + 8×*n* in the Page 0 component.

5.4.1 Field descriptions

The PMCGCR<0-1> bit assignments are:

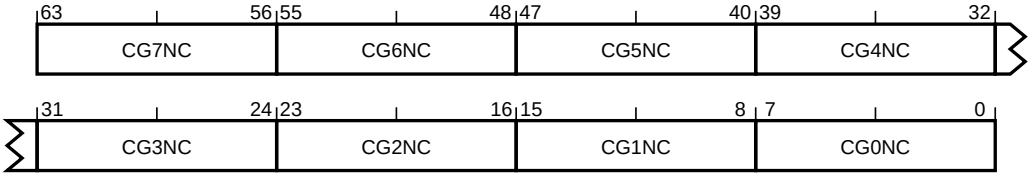


Figure 5.3: PMCGCR<*n*>

CG<*m*>NC[7:0], bits [*m*×8+7:*m*×8], for *m* = 0 to 7
Group (*n*×8+*m*) number of monitors.

The maximum size of each monitor group depends on the number of implemented groups and the largest implemented monitor size. For more information, see PMCFGR.NCG.

5.5 PMCNTEN, Count Enable Register

The PMCNTEN characteristics are:

Purpose
Monitor enables.

Usage constraints
PMCNTEN reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCNTEN is a 64-bit read/write memory-mapped register located at offset `0xC10` in the Page 0 component.

5.5.1 Field descriptions

The PMCNTEN bit assignments are:

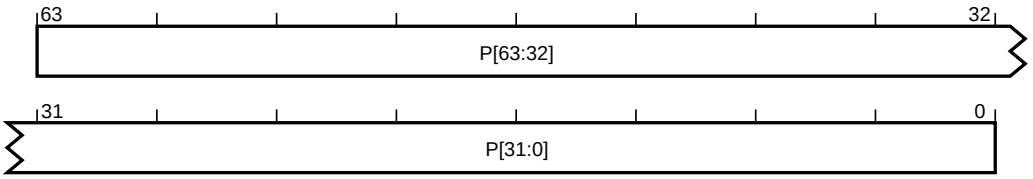


Figure 5.4: PMCNTEN

P<*m*>, bit [*m*], for *m* = 0 to 63
PMEVCNTR<*m*> enable. The possible values of this bit are:

`0b0`	PMEVCNTR<*m*> disabled.
`0b1`	PMEVCNTR<*m*> enabled.

If the Cycle Counter extension is implemented, then PMCNTEN.P[31] is the PMCCNTR enable.

5.6 PMCNTENCLR, Count Enable Clear Register

The PMCNTENCLR characteristics are:

Purpose
Disable monitors.

Usage constraints
PMCNTENCLR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCNTENCLR is a 64-bit read/write memory-mapped register located at offset `0xC20` in the Page 0 component.

5.6.1 Field descriptions

The PMCNTENCLR bit assignments are:

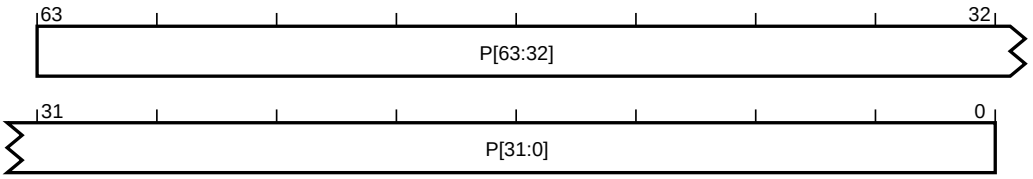


Figure 5.5: PMCNTENCLR

All fields are unchanged.

5.7 PMCNTENSET, Count Enable Set Register

The PMCNTENSET characteristics are:

Purpose
Enable monitors.

Usage constraints
PMCNTENSET reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCNTENSET is a 64-bit read/write memory-mapped register located at offset `0xC00` in the Page 0 component.

5.7.1 Field descriptions

The PMCNTENSET bit assignments are:

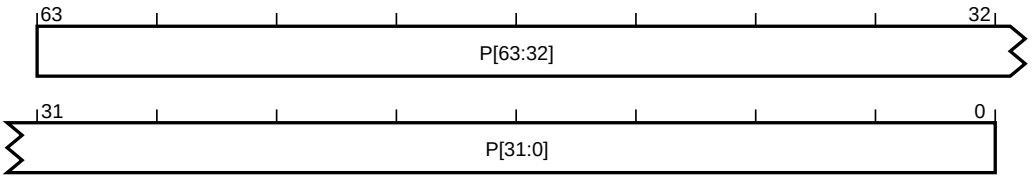


Figure 5.6: PMCNTENSET

All fields are unchanged.

5.8 PMCR, Control Register

The PMCR characteristics are:

Purpose
Main control register for the performance monitors.

Usage constraints
PMCR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMCR is a 64-bit read/write memory-mapped register located at offset `0xE10` in the Page 0 component.

5.8.1 Field descriptions

The PMCR bit assignments are:

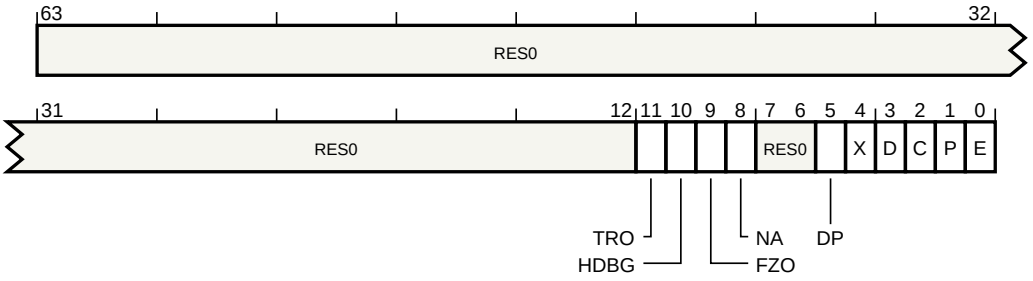


Figure 5.7: PMCR

All fields are unchanged.

5.9 PMDEVARCH, Device Architecture Register

The PMDEVARCH characteristics are:

Purpose
Provides discovery information for the component.

Usage constraints
PMDEVARCH reads-as-zero if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations
Always implemented.

Attributes
PMDEVARCH is a 32-bit read-only memory-mapped register located at offset `0xFBC`.

5.9.1 Field descriptions

When PMDEVARCH is implemented, the PMDEVARCH bit assignments are:

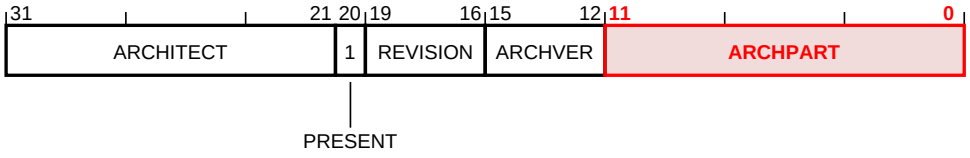


Figure 5.8: PMDEVARCH (PMDEVARCH is implemented)

When PMDEVARCH is not implemented, the PMDEVARCH bit assignments are:

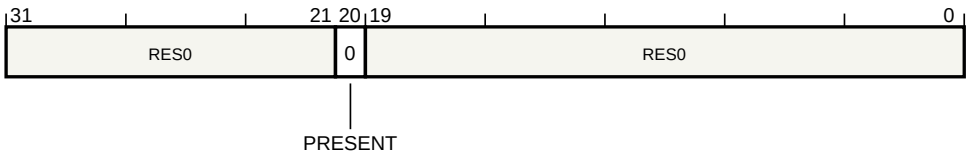


Figure 5.9: PMDEVARCH (PMDEVARCH is not implemented)

ARCHPART, bits [11:0], when PMDEVARCH is implemented
Architecture Part. Defines the architecture of the component.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

When Generic CoreSight PMU
The defined values of this field are:

`0xA67`	Generic *Activity Monitor Unit* (AMU), 64-bit programmers' model extension implemented.

0xAF4	Generic CoreSight PMU, dual-page extension not implemented, 64-bit programmers' model extension implemented.
0xAF5	Generic CoreSight PMU Page 0, dual-page extension implemented, 64-bit programmers' model extension implemented.
0xAF6	Generic CoreSight PMU Page 1, dual-page extension implemented, 64-bit programmers' model extension implemented.

This field reads as an IMPLEMENTATION DEFINED value.

Otherwise

This field reads as an IMPLEMENTATION DEFINED value.

Other fields are unchanged.

5.10 PMEVFILT2R<*n*>, Event Filter 2 Register <*n*>

The PMEVFILT2R<*n*> characteristics are:

Purpose

For performance monitors requiring event selection controls in addition to the PMEVTYPER<*n*> and PMEVFILTR<*n*> registers.

Usage constraints

PMEVFILT2R<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

Always implemented.

Attributes

PMEVFILT2R<*n*> is a 64-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x800` + 8×*n* in the Page 0 component.

5.10.1 Field descriptions

The PMEVFILT2R<*n*> bit assignments are:

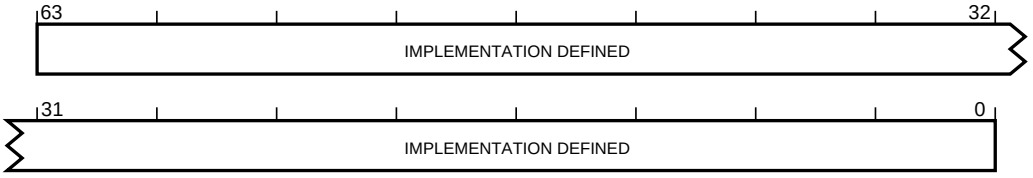


Figure 5.10: PMEVFILT2R<*n*>

All fields are unchanged.

5.11 PMEVFILTR<*n*>, Event Filter Register <*n*>

The PMEVFILTR<*n*> characteristics are:

Purpose
For performance monitors requiring event selection controls in addition to the PMEVTYPER<*n*> register.

Usage constraints
PMEVFILTR<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVFILTR<*n*> are UNPREDICTABLE if PMEVFILTR<*n*> is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMEVFILTR<*n*> is a 64-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0xA00` + 8×*n* in the Page 0 component.

5.11.1 Field descriptions

The PMEVFILTR<*n*> bit assignments are:

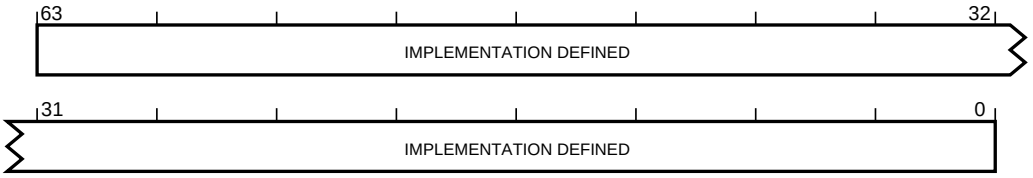


Figure 5.11: PMEVFILTR<*n*>

All fields are unchanged.

5.12 PMEVTYPER<*n*>, Event Type Select Register <*n*>

The PMEVTYPER<*n*> characteristics are:

Purpose
Configures monitor <*n*>.

Usage constraints
PMEVTYPER<*n*> reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMEVTYPER<*n*> are UNPREDICTABLE if PMEVTYPER<*n*> is read/write, PMCFGR.NA == `0b1`, and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMEVTYPER<*n*> is a 64-bit IMPLEMENTATION DEFINED read-only or read/write memory-mapped register located at offset `0x400` + 8×*n* in the Page 0 component.

5.12.1 Field descriptions

The PMEVTYPER<*n*> bit assignments are:

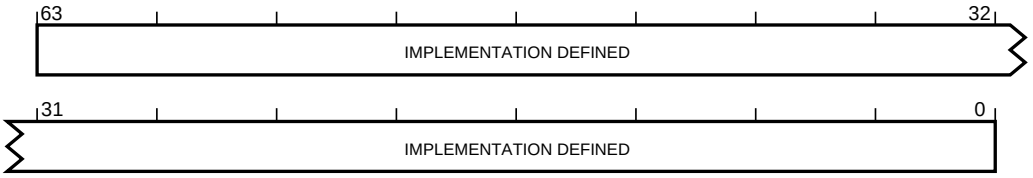


Figure 5.12: PMEVTYPER<*n*>

All fields are unchanged.

5.13 PMINTEN, Interrupt Enable Register

The PMINTEN characteristics are:

Purpose
Interrupt on monitor overflow status.

Usage constraints
PMINTEN reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMINTEN are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMINTEN is a 64-bit read/write memory-mapped register located at offset `0xC50` in the Page 0 component.

5.13.1 Field descriptions

The PMINTEN bit assignments are:

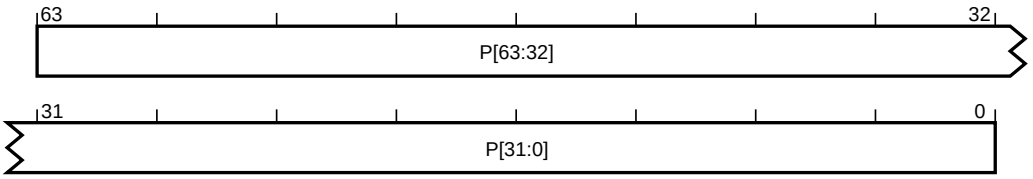


Figure 5.13: PMINTEN

P<*m*>, bit [*m*], for *m* = 0 to 63
Interrupt on overflow status of PMEVCNTR<*m*> enable. The possible values of this bit are:

`0b0`	Interrupt on overflow status of PMEVCNTR<*m*> disabled.
`0b1`	Interrupt on overflow status of PMEVCNTR<*m*> enabled.

If the Cycle Counter extension is implemented, then PMINTEN.P[31] is the interrupt on overflow status of PMCCNTR enable.

5.14 PMINTENCLR, Interrupt Enable Clear Register

The PMINTENCLR characteristics are:

Purpose
Disable interrupt on monitor overflow status.

Usage constraints
PMINTENCLR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMINTENCLR are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMINTENCLR is a 64-bit read/write memory-mapped register located at offset `0xC60` in the Page 0 component.

5.14.1 Field descriptions

The PMINTENCLR bit assignments are:

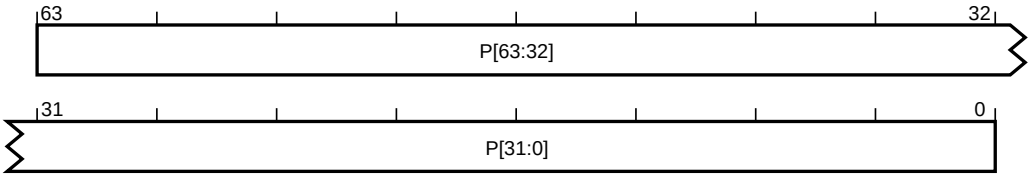


Figure 5.14: PMINTENCLR

All fields are unchanged.

5.15 PMINTENSET, Interrupt Enable Set Register

The PMINTENSET characteristics are:

Purpose
Enable interrupt on monitor overflow status.

Usage constraints
PMINTENSET reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMINTENSET are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMINTENSET is a 64-bit read/write memory-mapped register located at offset `0xC40` in the Page 0 component.

5.15.1 Field descriptions

The PMINTENSET bit assignments are:

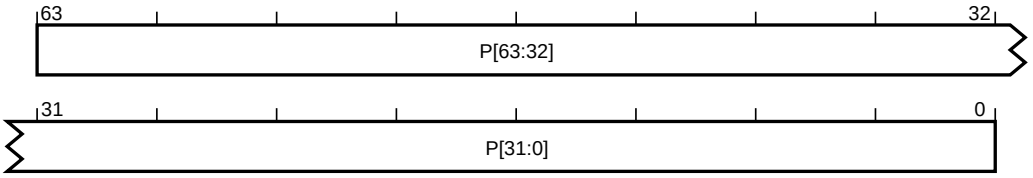


Figure 5.15: PMINTENSET

All fields are unchanged.

5.16 PMIRQCR12, Interrupt Configuration Register 12

The PMIRQCR12 characteristics are:

Purpose
Interrupt control and configuration register.

Usage constraints
PMIRQCR12 reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

PMIRQCR12 ignores writes if all of the following are true:

- Any of the following are true:
  - PMSCR is not implemented and PMIRQCR12.NSMSI configures the physical address space for a message signaled interrupt as Secure.
  - PMSCR is implemented and PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure.
- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.

Permitted writes to PMIRQCR12 are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
PMIRQCR12 is present only if PMCFGR.MSI == `0b1`, message signaled interrupts are implemented. PMIRQCR12 is RES0 otherwise.

Attributes
PMIRQCR12 is a 64-bit read/write memory-mapped register located at offset `0xE88` in the Page 0 component.

5.16.1 Field descriptions

The PMIRQCR12 bit assignments are:

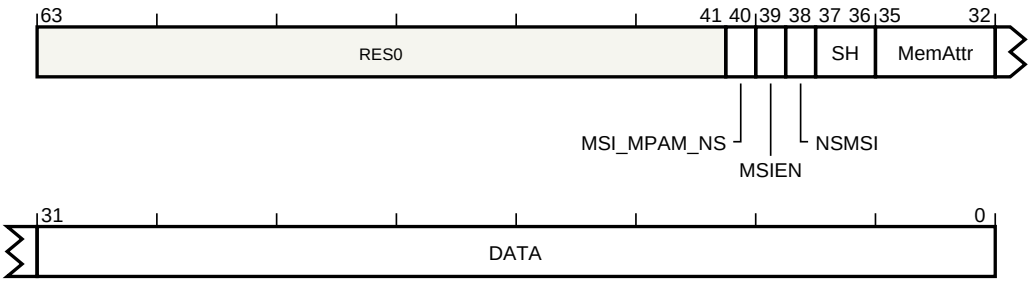


Figure 5.16: PMIRQCR12

Bits [63:41]
Reserved. This field is RES0.

MSI_MPAM_NS, bit [40]
Memory System Resource Partitioning and Monitoring (MPAM) Non-secure signaling for Secure MSIs.

When PMSCR is implemented, or PMIRQCR12.NSMSI configures MSIs as Non-secure MSIs
This bit is RES0.

When the PMU supports MPAM on message signaled interrupts
Defines the MPAM_NS setting used for Secure message signaled interrupts. The possible values of this bit are:

0b0	Secure MSIs are issued with MPAM_NS set to 0b0.
0b1	Secure MSIs are issued with MPAM_NS set to 0b1.

This bit resets to 0b0.

Otherwise
This bit is RES0.

MSIEN, bit [39]
Message signaled interrupt enable.

When the PMU supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

0b0	Disabled.
0b1	Enabled.

This bit resets to 0b0.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [38]
Non-secure message signaled interrupt.

When PMSCR is implemented
This bit is RES0.

When the PMU supports configuring the physical address space for message signaled interrupts
Defines the physical address space for message signaled interrupts. The possible values of this bit are:

0b0	Secure physical address space.
0b1	Non-secure physical address space.

Accessing this bit has the following behavior:

- This bit ignores writes if any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- Otherwise, this bit is read/write.

This bit resets to an IMPLEMENTATION DEFINED value.

Otherwise
The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [37:36]
Shareability.

When the PMU supports configuring the Shareability domain for message signaled interrupts
Defines the Shareability domain for message signaled interrupts. The possible values of this field are:

0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when PMIRQCR12.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

This field resets to an architecturally UNKNOWN value.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [35:32]
Memory type.

When the PMU supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

Note:

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

Otherwise
The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

DATA, bits [31:0]
Payload for the message signaled interrupt.

5.17 PMOVS, Overflow Flag Status Register

The PMOVS characteristics are:

Purpose
Monitor overflow status flags.

Usage constraints
PMOVS reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMOVS are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMOVS is a 64-bit read/write memory-mapped register located at offset `0xC90` in the Page 1 component.

5.17.1 Field descriptions

The PMOVS bit assignments are:

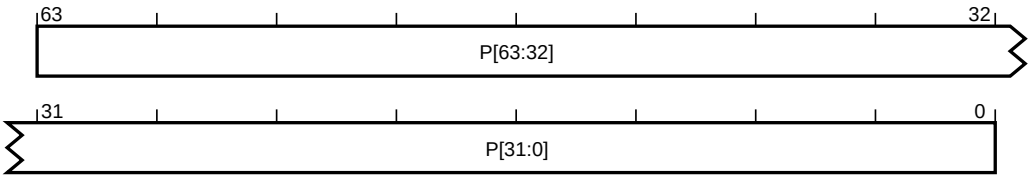


Figure 5.17: PMOVS

P<*m*>, bit [*m*], for *m* = 0 to 63
Overflow status flag for PMEVCNTR<*m*> overflow status. The possible values of this bit are:

`0b0`	PMEVCNTR<*m*> has not overflowed.
`0b1`	PMEVCNTR<*m*> has overflowed.

If the Cycle Counter extension is implemented, then PMOVS.P[31] is the overflow status flag for PMCCNTR overflow status.

5.18 PMOVSCLR, Overflow Flag Status Clear Register

The PMOVSCLR characteristics are:

Purpose
Clear PMU monitor overflow status flags.

Usage constraints
PMOVSCLR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMOVSCLR are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMOVSCLR is a 64-bit read/write memory-mapped register located at offset `0xC80` in the Page 1 component.

5.18.1 Field descriptions

The PMOVSCLR bit assignments are:

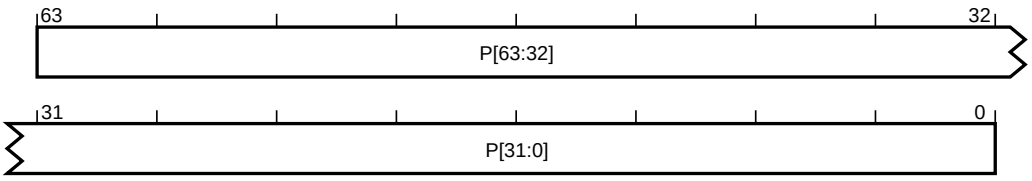


Figure 5.18: PMOVSCLR

All fields are unchanged.

5.19 PMOVSSET, Overflow Flag Status Set Register

The PMOVSSET characteristics are:

Purpose
Set PMU monitor overflow status flags.

Usage constraints
PMOVSSET reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Permitted writes to PMOVSSET are UNPREDICTABLE if PMCFGR.NA == `0b1` and PMCR.NA == `0b1`.

Configurations
Always implemented.

Attributes
PMOVSSET is a 64-bit read/write memory-mapped register located at offset `0xCC0` in the Page 1 component.

5.19.1 Field descriptions

The PMOVSSET bit assignments are:

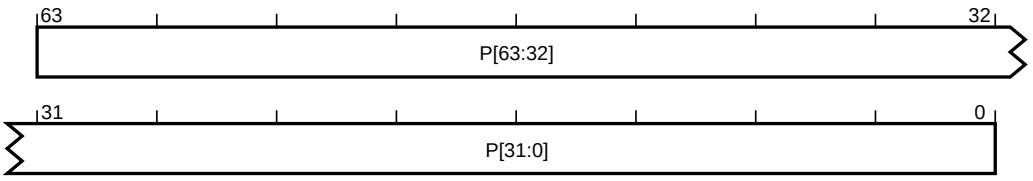


Figure 5.19: PMOVSSET

All fields are unchanged.

5.20 PMSSCR, Snapshot Control Register

The PMSSCR characteristics are:

Purpose

Provides a mechanism for software to initiate a sample, and holds status information about the captured monitors.

Usage constraints

PMSSCR reads-as-zero and ignores writes if all of the following are true:

- Any of the following are true:
  - The access is Non-secure.
  - The access is Realm.
- PMSCR is implemented.
- PMSCR.NSRA == `0b0`.

Configurations

PMSSCR is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSCR is RES0 otherwise.

Attributes

PMSSCR is a 64-bit read/write memory-mapped register located at offset `0xE30` in the Page 0 component.

5.20.1 Field descriptions

The PMSSCR bit assignments are:

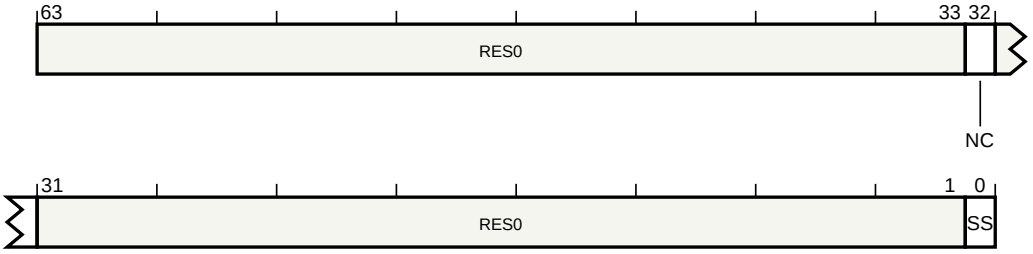


Figure 5.20: PMSSCR

All fields are unchanged.

Glossary

Activity Monitor Unit

A PMU that includes only activity monitors.

AMU

Activity Monitor Unit

Event-based sampling

The location in the program, or some other measurement, is recorded when a *sampled* event occurs. This builds up a statistical model of where events occur.

FDO

Feedback-directed Optimization

Feedback-directed Optimization

See Profile-guided Optimization.

Hardware Performance Monitor

A hardware resource that helps an engineer measure and profile software.

HPC

High-performance Computing

HPM

Hardware Performance Monitors

Little's Law

Little's Law ($L = \lambda W$) relates Arrival rate (λ) and Average occupancy (L) with average Response time (W).

Memory System Resource Partitioning and Monitoring

See *Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture* [2].

MPAM

Memory System Resource Partitioning and Monitoring

PE

Processing Element

PGO

Profile-guided Optimization

PMU

Performance Monitoring Unit

PPI

Private Peripheral Interrupt

Processing Element (PE)

The abstract machine defined in the Arm architecture, as documented in an Arm Architecture Reference Manual. An implementation of a PE that is compliant with the Arm architecture conforms with the behaviors described in the corresponding Arm Architecture Reference Manual.

Profile-guided Optimization

Profile-guided optimization, also referred to as feedback-directed optimization, is a compiler optimization technique that uses profiling to improve program runtime performance. For example, the profile guides the compiler for which areas of the program are executed more frequently, and which areas are executed less frequently.

R/W

Read/write.

R/W1C

Read, write-one-to-clear.

R/W1S

Read, write-one-to-set.

RO

Read-only.

Time-based sampling

Software periodically records values from the performance monitors and records the location in the program. The changes are tracked over time through phases of software execution. The engineer looks for correlations between phases and recorded events.

WO

Write-only.