



# Arm® CoreSight™ Architecture

## Performance Monitoring Unit Architecture

Document number	ARM IHI 0091
Document version	B.a
Document confidentiality	Non-confidential
Document build information	4029579c2

*Copyright © 2005-2025 Arm Limited or its affiliates. All rights reserved.*

# CoreSight Performance Monitoring Unit Architecture

## Release information

Date	Version	Changes
2025/Apr/25	B.a	• Third Non-Confidential EAC release.
2022/Oct/05	A.b	• Second Non-Confidential EAC release.
2020/Nov/04	A.a	• First Non-Confidential Beta release.

## Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited (“Arm”). **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Copyright © 2005-2025 Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-20349

8 March 2024



## Contents

# CoreSight Performance Monitoring Unit Architecture

CoreSight Performance Monitoring Unit Architecture . . . . .	ii
Release information . . . . .	ii
Non-Confidential Proprietary Notice . . . . .	iii

## Preface

Document status . . . . .	viii
Additional reading . . . . .	ix
About this book . . . . .	x
Using this book . . . . .	xi
Conventions . . . . .	xii
Typographical conventions . . . . .	xii
Numbers . . . . .	xii
Rules-based writing . . . . .	xii
Feedback . . . . .	xiv
Feedback on this book . . . . .	xiv
Progressive terminology statement . . . . .	xiv

## Chapter 1

### Description

1.1 About Performance Monitors . . . . .	16
1.1.1 Profiling and software optimization . . . . .	16
1.1.2 Monitoring . . . . .	17
1.1.3 Sampling . . . . .	17
1.2 Rationale for a standard PMU architecture . . . . .	20
1.3 What to measure . . . . .	21
1.4 <i>Memory System Resource Partitioning and Monitoring (MPAM)</i> . . . . .	24
1.4.1 Overview . . . . .	24

## Chapter 2

### Specification

2.1 Organization . . . . .	26
2.2 Operation . . . . .	30
2.2.1 Event counting . . . . .	30
2.2.2 State or event monitoring . . . . .	31
2.2.3 Mapping controls and fixed-function monitors . . . . .	32
2.2.4 Interrupt signaling . . . . .	32
2.3 Accuracy . . . . .	33
2.4 Activity Monitor Units (AMUs) . . . . .	34
2.5 Accessing registers . . . . .	36
2.6 Security . . . . .	38

## Chapter 3

### Extensions

3.1 Features summary . . . . .	42
3.2 Freeze on overflow extension . . . . .	45
3.3 Halt-on-debug extension . . . . .	47
3.4 Fixed-function cycle counter extension . . . . .	48
3.5 Monitor group extension . . . . .	49
3.6 Counter chaining extension . . . . .	50
3.7 Event counter threshold and edge detection extensions . . . . .	51
3.7.1 Pseudocode . . . . .	55
3.8 Reusable event filter definitions . . . . .	56

3.8.1	Security operating state filtering . . . . .	56
3.8.2	Exception level filtering . . . . .	58
3.8.3	VMID filtering . . . . .	63
3.8.4	MPAM filtering . . . . .	64
3.9	Snapshot extension . . . . .	67
3.10	Trace generation extension . . . . .	69
3.11	Export extension . . . . .	70
3.12	Dual-page extension . . . . .	71
3.13	Observability and access control extension . . . . .	73

## Chapter 4

### Programmers' Model

4.1	Memory-mapped registers summary . . . . .	76
4.2	When the 64-bit programmers' model extension is not implemented . . . . .	77
4.2.1	When the dual-page extension is not implemented . . . . .	77
4.2.2	Page 0, when the dual-page extension is implemented . . . . .	78
4.2.3	Page 1, when the dual-page extension is implemented . . . . .	79
4.3	When the 64-bit programmers' model extension is implemented . . . . .	81
4.3.1	When the dual-page extension is not implemented . . . . .	81
4.3.2	Page 0, when the dual-page extension is implemented . . . . .	82
4.3.3	Page 1, when the dual-page extension is implemented . . . . .	83
4.4	Register descriptions . . . . .	85
4.4.1	PMAUTHSTATUS, Authentication Status Register . . . . .	86
4.4.2	PMCCFILTR, Cycle Counter Filter Register . . . . .	88
4.4.3	PMCCNTR, Cycle Count Register . . . . .	90
4.4.4	PMCFGFR, Configuration Register . . . . .	92
4.4.5	PMCGCR<n>, Counter Group Configuration Registers, n = 0 - 3 . . . . .	102
4.4.6	PMCIDR0, Component Identification Register 0 . . . . .	104
4.4.7	PMCIDR1, Component Identification Register 1 . . . . .	105
4.4.8	PMCIDR2, Component Identification Register 2 . . . . .	107
4.4.9	PMCIDR3, Component Identification Register 3 . . . . .	108
4.4.10	PMCNTEN, Count Enable Set Register . . . . .	109
4.4.11	PMCNTENCLR<m>, Count Enable Clear Registers, m = 0 - 7 . . . . .	110
4.4.12	PMCNTENSET<m>, Count Enable Set Registers, m = 0 - 7 . . . . .	112
4.4.13	PMCR, Control Register . . . . .	114
4.4.14	PMDEVAFF, Device Affinity Register . . . . .	122
4.4.15	PMDEVARCH, Device Architecture Register . . . . .	127
4.4.16	PMDEVID, Device Configuration Register . . . . .	131
4.4.17	PMDEVTYPE, Device Type Register . . . . .	132
4.4.18	PMEVCNTR<n>, Event Count Registers, n = 0 - 127 . . . . .	134
4.4.19	PMEVFILT2R<n>, Event Filter 2 Registers, n = 0 - 127 . . . . .	136
4.4.20	PMEVFILTR<n>, Event Filter Registers, n = 0 - 127 . . . . .	138
4.4.21	PMEVTYPER<n>, Event Type Select Registers, n = 0 - 127 . . . . .	140
4.4.22	PMIIDR, Implementation Identification Register . . . . .	142
4.4.23	PMIMPDEF<n>, IMPLEMENTATION DEFINED Registers, n = 0 - 63 . . . . .	146
4.4.24	PMINTEN, Interrupt Enable Set Register . . . . .	148
4.4.25	PMINTENCLR<m>, Interrupt Enable Clear Registers, m = 0 - 7 . . . . .	149
4.4.26	PMINTENSET<m>, Interrupt Enable Set Registers, m = 0 - 7 . . . . .	151
4.4.27	PMIRQCR0, Interrupt Configuration Register 0 . . . . .	153
4.4.28	PMIRQCR1, Interrupt Configuration Register 1 . . . . .	154
4.4.29	PMIRQCR2, Interrupt Configuration Register 2 . . . . .	155
4.4.30	PMIRQCR12, Interrupt Configuration Register 12 . . . . .	158
4.4.31	PMIRQSR, Interrupt Status Register . . . . .	162
4.4.32	PMOVS, Overflow Status Snapshot Register . . . . .	164
4.4.33	PMOVSCLR<m>, Overflow Flag Status Clear Registers, m = 0 - 7 . . . . .	165
4.4.34	PMOVSSSET<m>, Overflow Flag Status Set Registers, m = 0 - 7 . . . . .	167

4.4.35	PMOVSSR<n>, Overflow Status Snapshot Registers, n = 0 - 7 . . . . .	169
4.4.36	PMPIDR0, Peripheral Identification Register 0 . . . . .	171
4.4.37	PMPIDR1, Peripheral Identification Register 1 . . . . .	172
4.4.38	PMPIDR2, Peripheral Identification Register 2 . . . . .	174
4.4.39	PMPIDR3, Peripheral Identification Register 3 . . . . .	177
4.4.40	PMPIDR4, Peripheral Identification Register 4 . . . . .	179
4.4.41	PMPIDR5, Peripheral Identification Register 5 . . . . .	181
4.4.42	PMPIDR6, Peripheral Identification Register 6 . . . . .	182
4.4.43	PMPIDR7, Peripheral Identification Register 7 . . . . .	183
4.4.44	PMROOTCR, Root and Realm Control Register . . . . .	184
4.4.45	PMSCR, Secure Control Register . . . . .	187
4.4.46	PMSSCR, Snapshot Control Register . . . . .	191
4.4.47	PMSSRR, Snapshot Reset Register . . . . .	193
4.4.48	PMSSSR, Snapshot Status Register . . . . .	195
4.4.49	PMSVR<n>, Saved Value Registers, n = 0 - 127 . . . . .	197

## Glossary

# 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.



## 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 System Component Specification; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture*. (ARM IHI 0099) 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.

## About this book

I<sub>BBMHH</sub>

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<sub>GNQRW</sub>

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

I\_WZYHY

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***

*Normative* section 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
- A URL, for example <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:

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

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

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and 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 Declarations and 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:  $L_{iiii}$

- $L$  is a label that indicates the content class of the content item.
- $iiii$  is the identifier of the content item.

## Content item classes

### **Declaration**

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

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behavior.

A Declaration is rendered with the label  $D$ .

### **Rule**

A Rule is a statement that describes the behavior of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label  $R$ .

### **Goal**

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label  $G$ .

### **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*.

## **Feedback**

Arm welcomes feedback on its documentation.

### **Feedback on this book**

If you have comments or queries about our documentation, create a ticket at <https://support.developer.arm.com>. As part of the ticket, include:

- The title, *CoreSight Performance Monitoring Unit Architecture*.
- The number, ARM IHI 0091 B.a.
- The page number(s) to which your comments apply.
- The rule identifier(s) 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](mailto: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 explicitly 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

Sampling is a technique used in software profiling. Two commonly used techniques for sampling with performance monitors are Time-based sampling and Event-based sampling:

#### **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.

#### **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.

Sampling provides a developer with a statistical representation of the performance of the software or system under test.

#### **Examples**

The following figures show examples of sampling tool user interfaces.

## Chapter 1. Description

### 1.1. About Performance Monitors

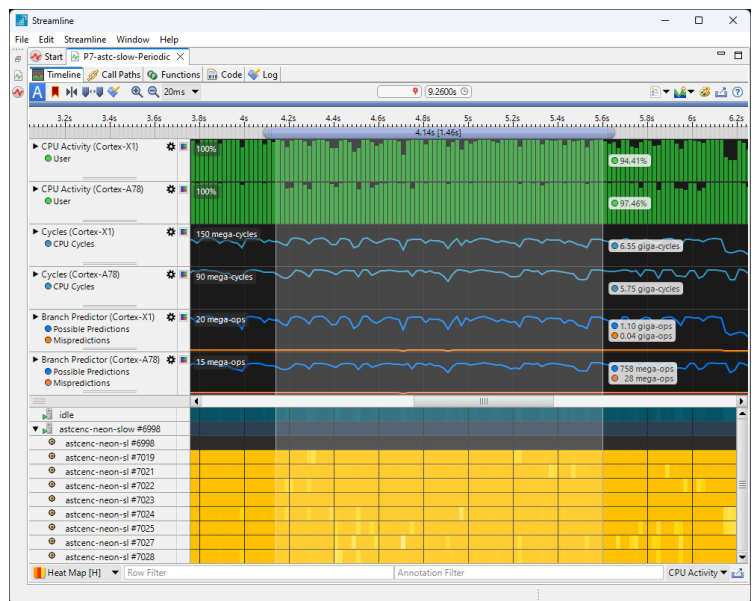


Figure 1.1: Time-based sampling with Arm Streamline Performance Analyzer

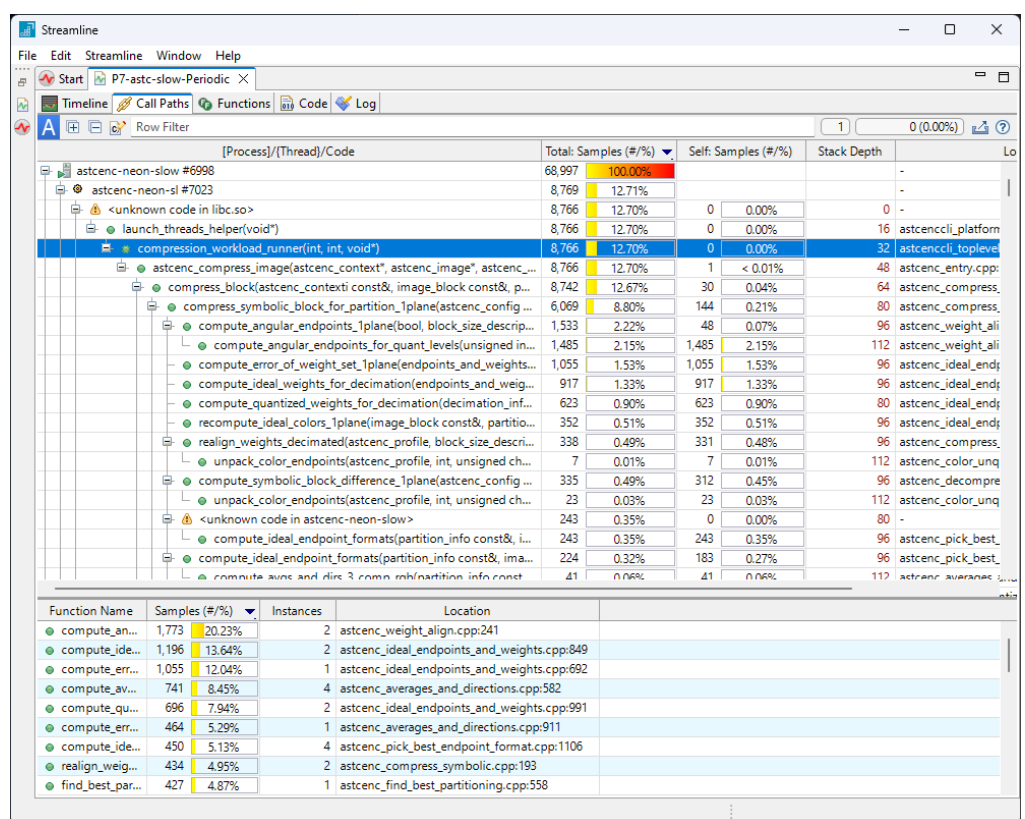


Figure 1.2: Call paths with hierarchy with Arm Streamline Performance Analyzer

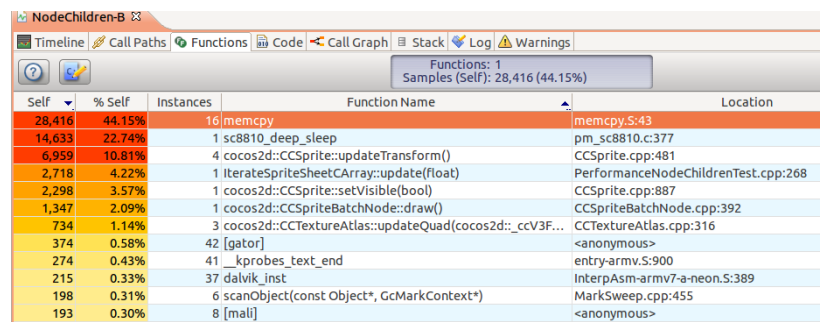


Figure 1.3: Hot-spot analysis from event-based sampling with Arm Streamline Performance Analyzer

## 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 ( $\rho$ ) 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 ( $\rho$ ), 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.

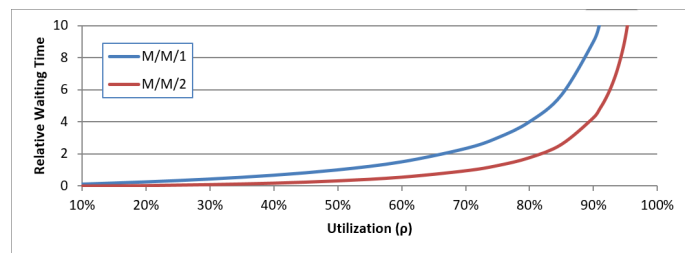


Figure 1.4: Queuing time as a function of utilization for M/M/1 and M/M/2 queues

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 evaluating 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 comparatively 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 System Component Specification; 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* ( $\mu$ ) 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* ( $\lambda$ ) 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](#) ( $\lambda$ ) 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 System Component Specification; Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture* [2].

### 1.4.1 Overview

Computer systems running multiple applications or virtual machines (VMs) concurrently and on shared memory often have one or more of the following requirements:

- A requirement to control the performance effects of non-conforming software on the performance of other software.
- A requirement to bound the performance impact on software by other software.
- A requirement to minimize the performance impact of some software on other software.

These scenarios are common in enterprise networking and server systems. The Memory System Resource Partitioning and Monitoring (MPAM) architecture 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 meet 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 MPAM memory-system component architecture describes:

- 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.
- An 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 might be required to monitor the partitioning of memory-system resources.



## Chapter 2

# Specification

This section is *normative*.

## 2.1 Organization

R <sub>KZZTB</sub>	A <i>Performance Monitoring Unit</i> (PMU) contains one or more <a href="#">monitors</a> that count events or otherwise monitor characteristics of one or more components in a system.
R <sub>DRPFZ</sub>	Each monitor <n> comprises: <ul style="list-style-type: none"> <li>• A <i>monitor value</i> register, <a href="#">PMEVCNTR&lt;n&gt;</a>.</li> <li>• Optional <i>monitor configuration</i> registers, to control event selection and filtering: <ul style="list-style-type: none"> <li>– <a href="#">PMEVTYPE&lt;n&gt;</a>.</li> <li>– <a href="#">PMEVFILTR&lt;n&gt;</a>.</li> <li>– <a href="#">PMEVFILT2R&lt;n&gt;</a>.</li> </ul> </li> <li>• A monitor enable bit, <a href="#">PMCNTEN[n]</a>.</li> </ul>
R <sub>LFSLV</sub>	Each monitor might define an optional <i>overflow condition</i> . If the monitor is an <a href="#">event counter</a> , the optional overflow condition is unsigned overflow of the event counter.
I <sub>MHCBB</sub>	An <a href="#">activity monitor</a> is an example of an <a href="#">event counter</a> that does not implement an <a href="#">overflow condition</a> . See <a href="#">2.4 Activity Monitor Units (AMUs)</a> .
R <sub>WRKBB</sub>	If a monitor <n> defines an <a href="#">overflow condition</a> , the monitor also comprises: <ul style="list-style-type: none"> <li>• A monitor overflow flag, <a href="#">PMOVS[n]</a>.</li> <li>• An interrupt enable bit, <a href="#">PMINTEN[n]</a>.</li> </ul>
R <sub>YCLQB</sub>	If a monitor <n> does not define an <a href="#">overflow condition</a> , <a href="#">PMOVS[n]</a> and <a href="#">PMINTEN[n]</a> are RES0.
I <sub>ZJTZQ</sub>	The names of registers in a functionally-compliant implementation might differ from those in this document. In particular: <ul style="list-style-type: none"> <li>• The mnemonic and name might include some reference to the component being monitored.</li> <li>• The mnemonics and names in this document are appropriate only for monitors that implement counters.</li> </ul>
R <sub>KRLSW</sub>	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 <a href="#">Arm® Architecture Reference Manual, for A-profile architecture [3]</a> .
R <sub>JFFVG</sub>	If <a href="#">FEAT_CSPMU_EXT32</a> is implemented, the PMU might include Counter Event Identification 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 <a href="#">Arm® Architecture Reference Manual, for A-profile architecture [3]</a> .
R <sub>KBRSS</sub>	When any <a href="#">monitor</a> defines an <a href="#">overflow condition</a> , the PMU includes an <a href="#">overflow interrupt request</a> signal.
R <sub>MRBCQ</sub>	Monitors might be split into <a href="#">monitor groups</a> .
R <sub>VPSKV</sub>	The size of each monitor value is IMPLEMENTATION DEFINED, up-to 64 bits.
S <sub>JRSCZ</sub>	The size of the largest monitor value is discoverable through <a href="#">PMCFGR.SIZE</a> .
U <sub>NNHQG</sub>	When <a href="#">event counters</a> are implemented, Arm recommends that all event counters are the same size.  Exceptions can be made for <i>fixed-function</i> 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 <sub>DPSDJ</sub>	When <a href="#">event counters</a> are implemented, the choice of counter size is usually determined by the periodicity and impact of an event counter unsigned overflow: <ul style="list-style-type: none"> <li>• The expected period between overflows is <math>\frac{2^{SZ}}{f \times E(N)}</math>, where SZ is the size of the counter, <math>E(N)</math> is the expected average number of events per counting cycle, and <math>f</math> is the counting frequency.</li> </ul>

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<sub>BNQPR</sub>

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

- If [FEAT\\_CSPMU\\_EXT32](#) is 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<sub>HKCFV</sub>

The number of monitors is IMPLEMENTATION DEFINED and is discoverable through [PMCFGR.N](#). If [FEAT\\_CSPMU\\_CG](#), [FEAT\\_CSPMU\\_EXT64](#), and [FEAT\\_CSPMU\\_SS](#) 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<sub>XDDYT</sub>

If any of [FEAT\\_CSPMU\\_CG](#), [FEAT\\_CSPMU\\_EXT64](#), or [FEAT\\_CSPMU\\_SS](#) are implemented, the maximum number of counters that can be implemented might be fewer than described by [R<sub>HKCFV</sub>](#). See the descriptions of the extensions for more information.

R<sub>GZMZD</sub>

When [FEAT\\_CSPMU\\_EXT32](#) is 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 × m)] bit to 0b1.
- A write of 0b1 to [PM{fn}CLR<m>](#)[q] clears the [PM{fn}](#)[q + (32 × 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 × m)].

R<sub>CVRCT</sub>

When [FEAT\\_CSPMU\\_EXT64](#) is 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 [FEAT\\_CSPMU\\_EXT64](#) 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 [FEAT\\_CSPMU\\_EXT32](#) is implemented. This register is required if [FEAT\\_CSPMU\\_EXT64](#) 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.

I\_PFGYY

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<sub>MFDCE</sub>

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 compatible 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<sub>GFGGZ</sub>

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

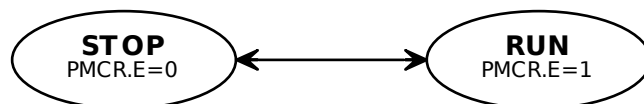
- Controls to partition the counters between those used by an Operating System and those used by a Hypervisor.
- Controls to configure overflow from either the bottom 32 bits or full 64 bits of the counter.

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](#).

## 2.2 Operation

**R<sub>WXSQT</sub>** The PMU has a *RUN* state and a *STOP* state. [Figure 2.1](#) shows this.



**Figure 2.1: PMU state-machine**

**R<sub>HLLTDF</sub>** The PMU operating state is determined by the [PMCR.E](#) control bit.

PMCR.E	State
0b0	<a href="#">STOP</a>
0b1	<a href="#">RUN</a>

**I<sub>KNMNQ</sub>** [FEAT\\_CSPMU\\_FZO](#) describes a third state, [WAIT](#).

**I<sub>CZDHC</sub>** An [AMU](#) does not implement the [STOP](#) state.

**S<sub>FJDNY</sub>** Software configures the monitors using the [monitor configuration](#) registers, [PMEVTYPEPER<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<sub>QLMFG</sub>](#) and [R<sub>HSHSJ</sub>](#).

**I<sub>SDBSN</sub>** 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<sub>LXYWS</sub>** 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<sub>TGVMJ</sub>** Increments of event counters by the PMU are atomic operations on the event counter.

**R<sub>XKQWT</sub>** Event counters are unsigned integer counters.

**R<sub>QLMFG</sub>** 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 [3.8 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: <ul style="list-style-type: none"> <li>The overflow status flag <a href="#">PMOVS</a>[<math>n</math>] is set to 0b1, if the optional status flag is implemented.</li> <li>The event counter wraps through zero.</li> </ul>
$I_{SCFBG}$	If <a href="#">FEAT_CSPMU_FZO</a> is not implemented or <i>freeze-on-overflow</i> is not enabled, the event counter continues counting events. Counting continues as long as the event counter is enabled, regardless of any overflows.
$I_{GKWHYH}$	Unsigned overflow of an event counter might generate a countable event. See <a href="#">3.6 Counter chaining extension</a> .
$S_{RBDW}$	Software can reset all event counters to zero in a single operation, by writing 0b1 to <a href="#">PMCR.P</a> .

**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: <ul style="list-style-type: none"> <li>The PMU is in the <a href="#">RUN</a> state.</li> <li>The monitor is enabled by <a href="#">PMCNTEN</a>[<math>n</math>].</li> <li>Monitoring is not prohibited. See <a href="#">2.6 Security</a>.</li> </ul>
$R_{MBSSC}$	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 <a href="#">monitor configuration</a> registers configure what the monitor monitors, including: <ul style="list-style-type: none"> <li>The state or events monitored by the monitor.</li> <li>Any filtering of the state or events, for example, on the context of the component being monitored. See also <a href="#">3.8 Reusable event filter definitions</a>.</li> </ul>
$I_{WNNQM}$	Depending on the implementation of the PMU, one or more of the <a href="#">monitor configuration</a> registers for a monitor might be RES0, meaning software does not need to program the register before enabling the monitor.
$R_{ZFGGW}$	The <a href="#">monitor value</a> register for a monitor is either read-only or read/write. If the <a href="#">monitor value</a> 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 <a href="#">PMOVS</a> [ $n$ ] is set to 0b1.
$I_{LPGWN}$	If <a href="#">FEAT_CSPMU_FZO</a> is not implemented or <i>freeze-on-overflow</i> is not enabled, the monitor continues operating after recording the overflow condition.
$R_{GHXCP}$	When software writes 0b1 to <a href="#">PMCR.P</a> : <ul style="list-style-type: none"> <li>If a monitor has a defined reset value or state, then the monitor is reset to its reset value or state.</li> <li>Otherwise, the monitor ignores the write to <a href="#">PMCR.P</a>.</li> </ul>

### 2.2.3 Mapping controls and fixed-function monitors

$R_{YSQVN}$	The mapping of the <a href="#">monitor configuration</a> register contents to controls is IMPLEMENTATION DEFINED, and might differ between monitors.
$I_{XNQJD}$	<a href="#">R<sub>YSQVN</sub></a> means that monitors might be dynamically programmable, but also that PMUs can have fixed monitors with IMPLEMENTATION DEFINED configurations.
$U_{KDGTO}$	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, nonzero value.

### 2.2.4 Interrupt signaling

$R_{MDCYL}$	<p>If the PMU implements an overflow interrupt then it is asserted when all of the following are true for any given monitor, <math>n</math>:</p> <ul style="list-style-type: none"> <li>• The overflow status flag <a href="#">PMOVS</a>[<math>n</math>] is 0b1.</li> <li>• The interrupt enable bit <a href="#">PMINTEN</a>[<math>n</math>] is 0b1.</li> <li>• The PMU is in the <a href="#">RUN</a> or <a href="#">WAIT</a> state.</li> </ul>
$S_{VFGSS}$	<p>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.</p> <p>Software is responsible for deasserting the overflow interrupt request. For example, by clearing the overflow status flags to 0b0.</p>
$I_{QQFDG}$	One overflow interrupt request is implemented for each PMU.
$R_{RSZKZ}$	The mechanism for signaling overflow interrupt requests is IMPLEMENTATION DEFINED.
$R_{FDLKQ}$	<p>When <a href="#">FEAT_CSPMU_MSI</a> is implemented, the PMU includes the <a href="#">PMIRQCR0</a>, <a href="#">PMIRQCR1</a>, and <a href="#">PMIRQCR2</a> registers to configure a message-signaled interrupt request.</p> <p>When <a href="#">FEAT_CSPMU_MSI</a> is implemented, when the PMU asserts the overflow interrupt request, the PMU writes the value specified in the <a href="#">PMIRQCR1</a> register to the address location specified by the <a href="#">PMIRQCR0</a> and <a href="#">PMIRQCR2</a> registers.</p> <p>The status of the message is indicated in <a href="#">PMIRQSR</a>. If this standard form of message-signaled interrupts are implemented then <a href="#">PMCFGR</a>.MSI reads as 0b1.</p>
$I_{VBWMC}$	<p>When an interrupt request signal is implemented, Arm strongly recommends that:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• If the PMU has close affinity to a single PE, then the interrupt request signal is configured as a <i>Private Peripheral Interrupt</i> (PPI) for that PE.</li> </ul>



## 2.3 Accuracy

I <sub>SDBRN</sub>	The PMU provides approximately accurate performance information.
R <sub>YTRMH</sub>	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 <sub>DYQXP</sub>	<p>There is no exact definition of <i>reasonable degree of inaccuracy</i>, but the following guidelines are recommended:</p> <ul style="list-style-type: none"><li>• Under normal operating conditions, the monitors must present an accurate value of the monitored characteristic.</li><li>• In exceptional circumstances, such as changes in Security state or other boundary conditions, it is acceptable for the value to be inaccurate.</li><li>• 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.</li></ul>

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.

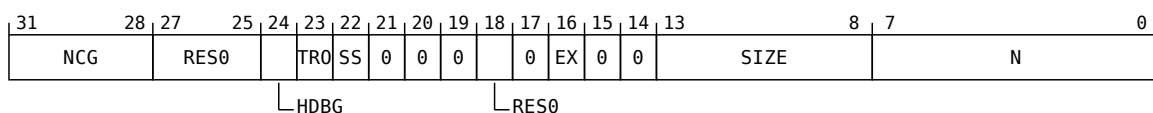
The permitted inaccuracy limits the possible uses of the PMU. In particular, the point in an operation pipeline where the monitor is updated is not defined relative to the point where a read of the monitor is made. This means that pipelining effects can cause some imprecision. An implementation must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

## 2.4 Activity Monitor Units (AMUs)

D <sub>RTBQV</sub>	An Activity Monitor Unit (AMU) is a PMU that includes only <a href="#">activity monitors</a> .
D <sub>DYHKN</sub>	<p>An <i>activity monitor</i> is a monitor with the following properties:</p> <ul style="list-style-type: none"> <li>Once initialized, the monitor is not reprogrammed again, unless reset by a hardware reset. If the monitor is an <a href="#">event counter</a>, this means the counter is <i>free-running</i>.</li> <li>The monitor does not implement an <a href="#">overflow condition</a>. This means the <a href="#">PMOVS</a> and <a href="#">PMINTEN</a> registers, and the <a href="#">overflow interrupt request</a> signal are not implemented.</li> <li>The monitor is often <i>fixed-function</i>, although this is not required.</li> </ul>
R <sub>HXNVJ</sub>	When the <a href="#">activity monitor</a> is enabled, writes to AMU <a href="#">monitor value</a> or <a href="#">monitor configuration</a> registers are not allowed and might be ignored or UNPREDICTABLE.
S <sub>LVLGJ</sub>	<p>System firmware programs an <a href="#">AMU</a> 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.</p> <p>This model guarantees a consistent view, supporting multiple accessing agents, for system management and monitoring.</p> <p>The system firmware is responsible for providing this guarantee. For instance, if an activity <a href="#">monitor configuration</a> 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.</p> <p>Allowing system firmware to configure an activity <a href="#">monitor configuration</a> 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.</p>
U <sub>ZSFGG</sub>	<p>The behavior of accesses to AMU registers when the <a href="#">AMU</a> is powered off or in a low-power retention state is IMPLEMENTATION DEFINED, as defined by <a href="#">I<sub>SDBSN</sub></a>.</p> <p>Arm recommends that AMU registers are RAZ/WI when the <a href="#">AMU</a> is powered off or in a low-power retention state.</p>
R <sub>QMNHC</sub>	<p>An <a href="#">AMU</a> optionally supports the following PMU extensions:</p> <ul style="list-style-type: none"> <li><a href="#">FEAT_CSPMU_HDBG</a>.</li> <li><a href="#">FEAT_CSPMU_SS</a>.</li> <li><a href="#">FEAT_CSPMU_CG</a>.</li> <li><a href="#">FEAT_CSPMU_EXT64</a>.</li> <li><a href="#">FEAT_CSPMU_ACR</a>.</li> </ul> <p>An AMU might also include <a href="#">FEAT_CSPMU_TRO</a> and <a href="#">FEAT_CSPMU_EX</a>, although these are not expected in most implementations.</p>
R <sub>BGHGF</sub>	<p>An <a href="#">AMU</a> does not support the following PMU features and extensions:</p> <ul style="list-style-type: none"> <li>The <a href="#">overflow interrupt request</a> signal.</li> <li><a href="#">FEAT_CSPMU_FZO</a>.</li> <li><a href="#">FEAT_CSPMU_CCNTR</a>. Although an AMU does typically include a fixed-function cycle counter, this is implemented as one of the general event counters.</li> <li><a href="#">Counter chaining</a>.</li> <li>The <a href="#">threshold extension</a> and <a href="#">edge-detect extension</a>.</li> <li>The <a href="#">MPAM filtering extension</a>.</li> <li><a href="#">FEAT_CSPMU_DUALPAGE</a>.</li> </ul>
I <sub>PWNVC</sub>	AMU registers typically have names starting <i>AM</i> . 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<m>	AMCNTENCLR<m>	
PMCNTENSET<m>	AMCNTENSET<m>	
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.
PMROOTCR	AMROOTCR	Optional.
PMSCR	AMSCR	Optional.

$R_{QZBSN}$  The following figure shows the bit assignments for the AMCFGR register. All other fields from PMCFGR are RAZ.

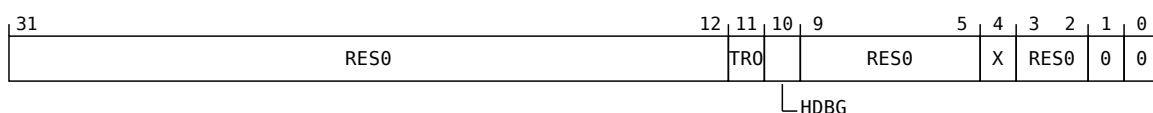


When the AMU implements FEAT\_CSPMU\_EXT64, AMCFGR is a 64-bit register.

$R_{DPGPN}$  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.

The following figure shows the bit assignments for the AMCR register, showing the optional control fields for FEAT\_CSPMU\_TRO, FEAT\_CSPMU\_HDBG, and FEAT\_CSPMU\_EX.



When the AMU implements FEAT\_CSPMU\_EXT64, AMCR is a 64-bit register.

$I_{NMXHP}$  If the AMROOTCR or AMSCR register is implemented, it follows the behavior described in 3.13 Observability and access control extension. However, for an AMU, the following observability controls are not implemented, as an activity monitor necessarily observes all activity:

- If AMROOTCR is implemented, AMROOTCR.{NAO, RLO, RTO}.
- If AMSCR is implemented, AMSCR.{NAO, SO}.

## 2.5 Accessing registers

R <sub>WVQDZ</sub>	It is IMPLEMENTATION DEFINED whether the <b>monitor value</b> registers, <b>PMEVCNTR&lt;n&gt;</b> , are writable through the register interface.
R <sub>RNPJY</sub>	It is IMPLEMENTATION DEFINED whether the <b>monitor configuration</b> registers, <b>PMEVTYPEPER&lt;n&gt;</b> , <b>PMEVFILTR&lt;n&gt;</b> , and <b>PMEVFILT2R&lt;n&gt;</b> , are writable through the register interface.
R <sub>CGWBM</sub>	If monitors are writable, then it is IMPLEMENTATION DEFINED and might be CONSTRAINED UNPREDICTABLE whether all <b>monitor value</b> and <b>monitor configuration</b> registers can be written to in states other than the <b>STOP</b> state.
R <sub>YNJLW</sub>	If monitors are writable and the PMU might not allow any <b>monitor value</b> or <b>monitor configuration</b> registers be written to in states other than the <b>STOP</b> state then <b>PMCFGR.NA</b> reads as 0b1. Otherwise, <b>PMCFGR.NA</b> reads as 0b0.
R <sub>GBTNB</sub>	If monitors are writable and <b>PMCFGR.NA</b> is 0b0, then it is further IMPLEMENTATION DEFINED whether the PMU allows writes to individual <b>monitor value</b> and <b>monitor configuration</b> registers in states other than the <b>STOP</b> state for a monitor that is enabled.
R <sub>CLXGJ</sub>	A write to a <b>monitor value</b> or <b>monitor configuration</b> register that is not allowed by <b>R<sub>CGWBM</sub></b> or <b>R<sub>GBTNB</sub></b> might do any of: <ul style="list-style-type: none"> <li>• Be ignored by the PMU.</li> <li>• Cause the monitor value to become incorrect.</li> </ul> <p>This rule does not apply to <b>PMCR</b> and the <b>FEAT_CSPMU_SS</b> registers.</p>
R <sub>YHTDZ</sub>	A write to a <b>monitor value</b> and <b>monitor configuration</b> register that is allowed by <b>R<sub>CGWBM</sub></b> and <b>R<sub>GBTNB</sub></b> in a state other than the <b>STOP</b> state takes effect in finite time. That is: <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
R <sub>JQPHY</sub>	A write to a <b>monitor value</b> and <b>monitor configuration</b> register in the <b>STOP</b> state takes effect before leaving the <b>STOP</b> state.
R <sub>QBXTC</sub>	Reads of <b>monitor value</b> and <b>monitor configuration</b> 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 <sub>PYSHB</sub>	A write to the register through the register interface will not be overwritten by a concurrent hardware update of the <b>monitor value</b> register, for example, when incrementing an event counter. See <b>R<sub>TGVMJ</sub></b> and <b>R<sub>NBWKD</sub></b> .  A following read of the <b>monitor value</b> 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 <sub>PVPYD</sub>	An implementation might define additional constraints on accessing PMU registers.
I <sub>DCPZN</sub>	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 <i>Arm® Architecture Reference Manual, for A-profile architecture</i> [3].
R <sub>ZWQLG</sub>	A memory-mapped PMU must implement the supported access sizes defined by the section <i>Supported access sizes</i> in the <i>Arm® Architecture Reference Manual, for A-profile architecture</i> [3].

R <sub>ZQQKP</sub>	When <a href="#">FEAT_CSPMU_EXT32</a> is 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 <sub>ZNTBP</sub>	When <a href="#">FEAT_CSPMU_EXT32</a> is implemented, it is IMPLEMENTATION DEFINED and recommended that doubleword-aligned 64-bit accesses are supported to the following pairs of 32-bit registers, where $m$ is even and both registers are implemented: <ul style="list-style-type: none"> <li>• <a href="#">PMCNTENSET&lt;m&gt;</a> and <a href="#">PMCNTENSET&lt;m+1&gt;</a>.</li> <li>• <a href="#">PMCNTENCLR&lt;m&gt;</a> and <a href="#">PMCNTENCLR&lt;m+1&gt;</a>.</li> <li>• <a href="#">PMINTENSET&lt;m&gt;</a> and <a href="#">PMINTENSET&lt;m+1&gt;</a>.</li> <li>• <a href="#">PMINTENCLR&lt;m&gt;</a> and <a href="#">PMINTENCLR&lt;m+1&gt;</a>.</li> <li>• <a href="#">PMOVSSET&lt;m&gt;</a> and <a href="#">PMOVSSET&lt;m+1&gt;</a>.</li> <li>• <a href="#">PMOVSCLR&lt;m&gt;</a> and <a href="#">PMOVSCLR&lt;m+1&gt;</a>.</li> </ul>
I <sub>MSFBY</sub>	<a href="#">R<sub>ZNTBP</sub></a> means that it is IMPLEMENTATION DEFINED and recommended that the PMU treats each $PM\{fn\}\{SETICLR\}\langle m \rangle$ and $PM\{fn\}\{SETICLR\}\langle m+1 \rangle$ register pair as a 64-bit register. This recommendation includes the case where the $\langle m+1 \rangle$ register is reserved because $32 \times (m + 1)$ or fewer monitors are implemented.
R <sub>DDZMD</sub>	When <a href="#">FEAT_CSPMU_EXT32</a> is 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 <sub>WHSBV</sub>	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 <sub>SDWBN</sub>	When <a href="#">FEAT_CSPMU_EXT64</a> is implemented, permitted doubleword-aligned 64-bit accesses are single-copy atomic at doubleword granularity.
R <sub>ZWZGC</sub>	A memory-mapped PMU implements the memory-mapped component synchronization rules defined by the section <i>Synchronization of memory-mapped registers</i> in the <a href="#">Arm® Architecture Reference Manual, for A-profile architecture</a> [3].
R <sub>KRVRO</sub>	A memory-mapped PMU implements the access requirements for reserved and unallocated registers defined by the section <i>Access requirements for reserved and unallocated registers</i> in the <a href="#">Arm® Architecture Reference Manual, for A-profile architecture</a> [3].

See also:

- [Arm® Architecture Reference Manual, for A-profile architecture](#) [3].
- [2.6 Security](#).
- [Chapter 4 Programmers' Model](#).

## 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<sub>JGBVD</sub> A PMU must not count events or expose characteristics when counting that event or monitoring that characteristic is [prohibited](#).
- R<sub>VHTGC</sub> 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<sub>PPGYJ</sub> The [Arm® Architecture Reference Manual, for A-profile architecture \[3\]](#) defines the following Security [operating states](#) for a PE and corresponding physical address spaces (PAS):
- Non-secure state and the Non-secure physical address space.
  - Secure state and the Secure physical address space.
  - Realm state and the Realm physical address space, when FEAT\_RME is implemented.
  - Root state and the Root physical address space, when FEAT\_RME is implemented.
- R<sub>WRKQV</sub> 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<sub>JVJPQ</sub> 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<sub>GRNVM</sub> 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 <sub>BLDJB</sub>	FEAT_CSPMU_ACR defines standard control registers that a PMU can implement to provide authentication controls.
R <sub>NHTQL</sub>	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 <sub>VSQRY</sub>	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 <sub>DBXDJ</sub>	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 <sub>GMSMS</sub>	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 <sub>CJBXN</sub>	R <sub>NHTQL</sub> , R <sub>VSQRY</sub> , and R <sub>DBXDJ</sub> 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 <sub>WZSYV</sub>	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 <sub>THBHR</sub>	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 <sub>RYNLT</sub>	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 <sub>BZLYW</sub>	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 <sub>RRLNM</sub>	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 <sub>WTVDK</sub>	The CoreSight architecture describes the NIDEN, DBGEN, SPIDEN, and SPNIDEN authentication interface signals. These signals control: <ul style="list-style-type: none"> <li>• Invasive and non-invasive debug authentication.</li> <li>• Secure and Non-secure debug authentication.</li> </ul>
I <sub>DWKKS</sub>	The Realm Management Extension describes the RLPIDEN and RTPIDEN authentication interface signals. These signals control Root and Realm debug authentication.

I<sub>DNMVG</sub>

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.

I<sub>TZLYG</sub>

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 Realm Management Extension \(RME\) System Architecture](#) [6].
- [ARM CoreSight Architecture Specification](#) [4].
- [3.13 Observability and access control extension](#).



## Chapter 3

# Extensions

## 3.1 Features summary

Each feature description includes:

- A feature name.
- Whether the implementation of the feature is mandatory or OPTIONAL.
- Dependencies on the implementation of other features, if any.
- The register field that identifies the presence of the feature, if any.

### FEAT\_CSPMU\_ACR, Observability and Access Control Extension

[FEAT\\_CSPMU\\_ACR](#) provides programmable authentication controls for privileged software.

[FEAT\\_CSPMU\\_ACR](#) is OPTIONAL.

For more information, see [3.13 Observability and access control extension](#).

### FEAT\_CSPMU\_CCNTR, Fixed-function Cycle Counter Extension

[FEAT\\_CSPMU\\_CCNTR](#) provides a fixed-function cycle counter that counts unhalting clock cycles using monitor 31.

[FEAT\\_CSPMU\\_CCNTR](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_CCNTR](#):

- PMCFGR.CC.

For more information, see [x 3.4 Fixed-function cycle counter extension](#).

### FEAT\_CSPMU\_CG, Monitor Group Extension

[FEAT\\_CSPMU\\_CG](#) enables support for monitor groups, allowing the *Performance Monitoring Unit* (PMU) to partition monitors into up-to 16 monitor groups. If [FEAT\\_CSPMU\\_CG](#) is implemented, then the number of monitor groups is IMPLEMENTATION DEFINED, and discoverable through [PMCFGR.NCG](#).

[FEAT\\_CSPMU\\_CG](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_CG](#):

- PMCFGR.NCG.

For more information, see [3.5 Monitor group extension](#).

### FEAT\_CSPMU\_DUALPAGE, Dual Page Extension

[FEAT\\_CSPMU\\_DUALPAGE](#) defines a dual-page register layout for the PMU, allowing certain registers to be accessed from a second page of memory-mapped addresses in addition to the primary page.

[FEAT\\_CSPMU\\_DUALPAGE](#) is OPTIONAL.

For more information, see [3.12 Dual-page extension](#).

### FEAT\_CSPMU\_EX, Export Extension

[FEAT\\_CSPMU\\_EX](#) exports events to an external monitoring agent, such as a trace unit, over an event bus, to provide triggering information.

[FEAT\\_CSPMU\\_EX](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_EX](#):

- PMCFGR.EX.

For more information, see [3.11 Export extension](#).

### **FEAT\_CSPMU\_EXT, Programmers' Model Extension**

[FEAT\\_CSPMU\\_EXT](#) defines the structure of a PMU, derived from the Armv8-A Performance Monitors Extension.

[FEAT\\_CSPMU\\_EXT](#) is OPTIONAL.

If [CSPMUv1p0](#) is implemented, then [FEAT\\_CSPMU\\_EXT](#) is implemented.

If [FEAT\\_CSPMU\\_EXT](#) is implemented, then [CSPMUv1p0](#) is implemented.

If [FEAT\\_CSPMU\\_EXT](#) is implemented, then [FEAT\\_CSPMU\\_EXT32](#) or [FEAT\\_CSPMU\\_EXT64](#) is implemented.

For more information, see [Chapter 4 Programmers' Model](#).

### **FEAT\_CSPMU\_EXT32, 64-bit Programmers' Model Extension not implemented**

[FEAT\\_CSPMU\\_EXT32](#) defines the location, layout, and content of PMU registers when the 64-bit programmers' model extension is not implemented

[FEAT\\_CSPMU\\_EXT32](#) is OPTIONAL.

If [FEAT\\_CSPMU\\_EXT32](#) is implemented, then [FEAT\\_CSPMU\\_EXT](#) is implemented.

If [FEAT\\_CSPMU\\_EXT32](#) is implemented, then [FEAT\\_CSPMU\\_EXT64](#) is not implemented.

For more information, see [4.2 When the 64-bit programmers' model extension is not implemented](#).

### **FEAT\_CSPMU\_EXT64, 64-bit Programmers' Model Extension**

[FEAT\\_CSPMU\\_EXT64](#) defines a 64-bit programmers' model where most PMU registers are 64-bit and accessed at doubleword-aligned addresses, enabling up to 64 monitors and supporting direct access to monitor control and status bits.

[FEAT\\_CSPMU\\_EXT64](#) is OPTIONAL.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, then [FEAT\\_CSPMU\\_EXT](#) is implemented.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, then [FEAT\\_CSPMU\\_EXT32](#) is not implemented.

For more information, see [4.3 When the 64-bit programmers' model extension is implemented](#).

### **FEAT\_CSPMU\_FZO, Freeze on Overflow Extension**

[FEAT\\_CSPMU\\_FZO](#) disables (freezes) monitors when any monitor overflows.

[FEAT\\_CSPMU\\_FZO](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_FZO](#):

- PMCFGR.FZO.

For more information, see [3.2 Freeze on overflow extension](#).

### **FEAT\_CSPMU\_HDBG, Halt on Debug Extension**

[FEAT\\_CSPMU\\_HDBG](#) stops events being counted when the affine PE or agent is in a halted state such as Debug state.

[FEAT\\_CSPMU\\_HDBG](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_HDBG](#):

- PMCFGR.HDBG.

For more information, see [3.3 Halt-on-debug extension](#).

## FEAT\_CSPMU\_MSI, Message-signaled Interrupts Extension

[FEAT\\_CSPMU\\_MSI](#) allows the PMU to trigger an interrupt by writing a specified value to a configured address on overflow, using dedicated registers to configure the address, data, and attributes for the write.

[FEAT\\_CSPMU\\_MSI](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_MSI](#):

- PMCFGR.MSI.

For more information, see [2.2 Operation](#).

## FEAT\_CSPMU\_SS, CSPMU Snapshot Extension

[FEAT\\_CSPMU\\_SS](#) provides a snapshot extension that captures PMU state, including monitor values, overflow flags, and optional syndrome data into snapshot registers on a capture event, using a configurable set of 32-bit or 64-bit registers depending on the programmers' model.

[FEAT\\_CSPMU\\_SS](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_SS](#):

- PMCFGR.SS.

For more information, see [3.9 Snapshot extension](#).

## FEAT\_CSPMU\_TRO, Trace Generation Extension

[FEAT\\_CSPMU\\_TRO](#) enables the PMU to generate a trace of events or event monitor data.

[FEAT\\_CSPMU\\_TRO](#) is OPTIONAL.

The following field identifies the presence of [FEAT\\_CSPMU\\_TRO](#):

- PMCFGR.TRO.

For more information, see [3.10 Trace generation extension](#).

## FEAT\_PERIPHERAL\_ID

[FEAT\\_PERIPHERAL\\_ID](#) implements the Peripheral Identification scheme, which uses Peripheral ID and Component ID registers to provide information about the configuration of the peripheral.

[FEAT\\_PERIPHERAL\\_ID](#) is OPTIONAL.

## MAX\_COUNTER\_SIZE

The size in bits of the largest implemented monitor register, [PMEVCNTR](#).

For more information, see x [2.1 Organization](#).

## NUM\_CSPMU\_COUNTERS

The number of [PMEVCNTR](#) monitors implemented.

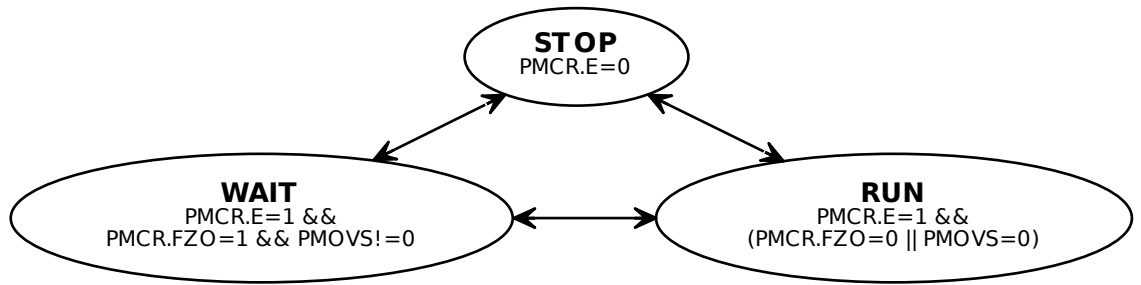
For more information, see x [2.1 Organization](#).

## NUM\_IMPDEF\_REGISTERS

The number of [PMIMPDEF](#) registers implemented.

## 3.2 Freeze on overflow extension

- R<sub>YTDDD</sub>** When **FEAT\_CSPMU\_FZO** is implemented, the PMU implements a control, **PMCR.FZO**, to disable (freeze) monitors when any monitor overflows.
- R<sub>GYWBJ</sub>** If **FEAT\_CSPMU\_FZO** is implemented then **PMCFGR.FZO** reads as 0b1.
- I<sub>FSDCY</sub>** If **FEAT\_CSPMU\_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<sub>MLXVL</sub>** 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<sub>JKHCM</sub>** and **R<sub>VPDQG</sub>**. This includes overflow flags for disabled monitors.

- I<sub>DHXMS</sub>** 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<sub>JKHCM</sub>** If a PMU implements **FEAT\_CSPMU\_FZO** and the **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<sub>NXGXL</sub>** 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<sub>DPGBH</sub>

If a PMU implements [FEAT\\_CSPMU\\_FZO](#) and [FEAT\\_CSPMU\\_CCNTR](#), 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<sub>VPDGO</sub>

If a PMU implements [FEAT\\_CSPMU\\_FZO](#) and [FEAT\\_CSPMU\\_CCNTR](#) 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:

- [3.4 Fixed-function cycle counter extension](#).
- [3.6 Counter chaining extension](#).

## 3.3 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.
$R_{SQJTW}$	<p>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:</p> <ul style="list-style-type: none"><li>• Events are counted in the Debug operating state.</li><li>• <a href="#">FEAT_CSPMU_HDBG</a> is implemented. The PMU includes a control, <a href="#">PMCR.HDBG</a>, that controls whether events are counted in the Debug operating state.</li><li>• Events are not counted in the Debug operating state.</li></ul> <p>This might differ for groups of events.</p>
$R_{XPPSF}$	If the PMU implements <a href="#">FEAT_CSPMU_HDBG</a> then <a href="#">PMCFGR.HDBG</a> reads as 0b1.

## 3.4 Fixed-function cycle counter extension

I <sub>SZJRL</sub>	<p><a href="#">FEAT_CSPMU_CCNTR</a> 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.</p> <p>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.</p>
R <sub>KFQDK</sub>	When <a href="#">FEAT_CSPMU_CCNTR</a> is implemented, the PMU includes a dedicated <i>cycle counter</i> , <a href="#">PMCCNTR</a> . <a href="#">PMCCNTR</a> counts unhalted clock cycles in the clock domain of the PMU.
R <sub>KFHHP</sub>	If <a href="#">FEAT_CSPMU_CCNTR</a> is implemented then <a href="#">PMCFGR</a> .CC reads as 0b1.
R <sub>HJZYK</sub>	For compatibility with the Arm architecture PMU, the cycle counter is always an alias for <a href="#">PMEVCNTR</a> 31.
R <sub>CMLXS</sub>	If <a href="#">FEAT_CSPMU_CCNTR</a> 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 <a href="#">PMCFGR</a> .CCD reads as 0b1.
I <sub>JNMYM</sub>	If <a href="#">FEAT_CSPMU_CCNTR</a> is not implemented then monitor 31 (if implemented) is a regular monitor.
R <sub>VZKXS</sub>	<p>If <a href="#">FEAT_CSPMU_CCNTR</a> is implemented and the agent being monitored has two operating states then the cycle counter:</p> <ul style="list-style-type: none"> <li>• Counts when <a href="#">PMCR</a>.DP is 0b0 or the agent being monitored is in the first state.</li> <li>• Does not count when <a href="#">PMCR</a>.DP is 0b1 and the agent being monitored is in the second state.</li> </ul> <p>The definitions of the first and second states are IMPLEMENTATION DEFINED.</p>
I <sub>BNDHS</sub>	<p>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.</p> <ul style="list-style-type: none"> <li>• Depending on the features of the PE and the values of the <a href="#">MDCR_EL3.SPME</a> and <a href="#">MDCR_EL3.HPMD</a> controls, the second state might include Secure state or EL2.</li> <li>• If <a href="#">FEAT_PMUv3p5</a> is implemented, <a href="#">MDCR_EL3.SCCD</a> disables the cycle counter in Secure state and <a href="#">MDCR_EL2.HCCD</a> disables the cycle counter at EL2, regardless of the value of <a href="#">PMCR</a>.DP.</li> </ul>
I <sub>SVGVM</sub>	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](#).
- [3.2 Freeze on overflow extension](#).



## 3.5 Monitor group extension

$I_{GPQWD}$	When <a href="#">FEAT_CSPMU_CG</a> is implemented, the PMU includes up-to 16 <i>monitor groups</i> . Monitor groups are a mechanism for a PMU to partition monitors.
$R_{KZNGG}$	If <a href="#">FEAT_CSPMU_CG</a> is implemented, the number of monitor groups is IMPLEMENTATION DEFINED, and discoverable through <a href="#">PMCFGR.NCG</a> .
$R_{ZSFHF}$	If <a href="#">FEAT_CSPMU_CG</a> is implemented then <a href="#">PMCFGR.NCG</a> 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 <a href="#">FEAT_CSPMU_EXT64</a> is implemented.

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

- Configuration **A**, when [FEAT\\_CSPMU\\_EXT32](#) is implemented, and monitors are 32 bits or smaller.
- Configuration **B**, when [FEAT\\_CSPMU\\_EXT32](#) is implemented, and at least one monitor is larger-than 32 bits.
- Configuration **C**, when [FEAT\\_CSPMU\\_EXT64](#) 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 <a href="#">PMEVCNTR&lt;<math>m \times max</math>&gt;</a> , where max is the value defined by <a href="#">R<sub>TFNPJ</sub></a> .
$I_{HGRPD}$	For example, an implementation might define: <ul style="list-style-type: none"> <li>• <a href="#">PMCFGR.NCG</a> reads as 0b0001, indicating 2 <i>monitor groups</i>.</li> <li>• <a href="#">PMCGCR0</a> reads as 0x00000604, indicating: <ul style="list-style-type: none"> <li>– Group 0 has four monitors: <a href="#">PMEVCNTR0</a> to <a href="#">PMEVCNTR3</a>.</li> <li>– Group 1 has six monitors: <a href="#">PMEVCNTR32</a> to <a href="#">PMEVCNTR37</a>.</li> </ul> </li> <li>• <a href="#">PMCFGR.N</a> reads as 0x09, indicating 10 monitors are implemented overall.</li> </ul>
$I_{JJQRM}$	The System MMU architecture implements a different model of monitor groups.

## 3.6 Counter chaining extension

I <sub>TCBXV</sub>	Support for larger event counters can be provided by <i>counter chaining</i> .
R <sub>YBGZK</sub>	If <a href="#">counter chaining</a> is implemented, then an odd-numbered counter <a href="#">PMEVCNTR&lt;n+1&gt;</a> configured to count the CHAIN event increments when the even-numbered counter <a href="#">PMEVCNTR&lt;n&gt;</a> has an unsigned overflow on counting an event.
S <sub>PGJDB</sub>	For example where two 16-bit counters, <a href="#">PMEVCNTR0</a> and <a href="#">PMEVCNTR1</a> are provided, they can be chained to form a 32-bit counter by programming <a href="#">PMEVTYPE1</a> to count the CHAIN event.
R <sub>DJHLY</sub>	<p>It is IMPLEMENTATION DEFINED whether the increment of the odd-numbered counter <a href="#">PMEVCNTR&lt;n+1&gt;</a> occurs atomically with update of the even-numbered counter <a href="#">PMEVCNTR&lt;n&gt;</a>.</p> <p>If the increment is not atomic it must occur in limited finite time such that a read of <a href="#">PMEVCNTR&lt;n+1&gt;</a> that is ordered-after a read of <a href="#">PMEVCNTR&lt;n&gt;</a> must not observe the updated <a href="#">PMEVCNTR&lt;n&gt;</a> without also observing the updated <a href="#">PMEVCNTR&lt;n+1&gt;</a>. However, it is possible that the reads might observe the updated <a href="#">PMEVCNTR&lt;n+1&gt;</a> without observing the updated <a href="#">PMEVCNTR&lt;n&gt;</a>.</p>
S <sub>QZDDJ</sub>	If the updates are not atomic, software must take care when reading the pair of counters.
	<p>See also:</p> <ul style="list-style-type: none"> <li>• <a href="#">3.2 Freeze on overflow extension</a>.</li> </ul>

### 3.7 Event counter threshold and edge detection extensions

I<sub>FVTLP</sub>

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.

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 supported threshold conditions are:

- Less-than.
- Greater-than-or-equal-to.
- Not-equals.
- Equal-to.

The amount the counter counts by is either one or the amount the event would have counted by.

I<sub>YCRKM</sub>

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.

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 transitions between zero and nonzero. The supported edge conditions are:

- Leading-edge.
- Falling-edge.
- Either edge.

R<sub>YDRBX</sub>

The values  $V_B[n]$ ,  $C_T[n]$ ,  $V_T[n]$ ,  $C_P[n]$ ,  $C_E[n]$ ,  $V_E[n]$ , and  $V[n]$  for an event counter  $\langle n \rangle$  are defined as follows:

- $V_B[n]$  is the (base) value the counter increments by when [threshold counting](#) is [disabled](#) and [edge counting](#) is [disabled](#) for the event counter  $\langle n \rangle$ .
- If the [threshold extension](#) is implemented, then  $C_T[n]$  is the result of evaluating the [threshold condition](#).  $C_T[n]$  is always TRUE or FALSE.
- $V_T[n]$  is the value that results from applying the [threshold condition](#).  $V_T[n]$  is one of  $V_B[n]$ , one, or zero.
- $C_P[n]$  is the value  $C_T[n]$  took on the previous cycle if counting the event was allowed on the previous cycle, and FALSE otherwise.  $C_P[n]$  is always TRUE or FALSE.

$C_P[n]$  is UNKNOWN if the configuration of the PMU changes such that the event counter  $\langle n \rangle$  might have counted a different event, or not counted any event, on the previous cycle. 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  $\langle n \rangle$ .

- $C_E[n]$  is the result of evaluating the [edge condition](#).  $C_E[n]$  is always TRUE or FALSE.
- $V_E[n]$  is the value that results from applying the [edge condition](#).  $V_E[n]$  is one if  $C_E[n]$  is TRUE and zero if  $C_E[n]$  is FALSE.
- $V[n]$  is the value that results from applying both the threshold and edge conditions.  $V[n]$  is one of  $V_E[n]$ ,  $V_T[n]$ , or  $V_B[n]$ , depending on the configuration of counter  $\langle n \rangle$ .

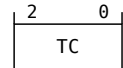
If an event counter  $\langle m \rangle$  is disabled, then  $V[m]$  is zero.

R<sub>KVVS</sub>K

When the [threshold extension](#) is implemented by an event counter <n>, the event counter implements two control fields, TC and TH:

- The *threshold condition* is controlled by TC and TH. TC also controls the amount the counter increments by when the [threshold condition](#) is met.
- 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*.

TC is a 3-bit field.

**TC, bits [2:0]**

Threshold Control. Defines the threshold condition.

TC	Description	Meaning
0b000	Not-equal.	$V_T[n]$ is $V_B[n]$ if $V_B[n]$ is not equal to TH, and zero otherwise. If TH is zero, then threshold counting is <a href="#">disabled</a> .
0b001	Not-equal, count.	$V_T[n]$ is 1 if $V_B[n]$ is not equal to TH, and zero otherwise.
0b010	Equals.	$V_T[n]$ is $V_B[n]$ if $V_B[n]$ is equal to TH, and zero otherwise.
0b011	Equals, count.	$V_T[n]$ is 1 if $V_B[n]$ is equal to TH, and zero otherwise.
0b100	Greater-than-or-equal.	$V_T[n]$ is $V_B[n]$ if $V_B[n]$ is TH or greater, and zero otherwise.
0b101	Greater-than-or-equal, count.	$V_T[n]$ is 1 if $V_B[n]$ is TH or greater, and zero otherwise.
0b110	Less-than.	$V_T[n]$ is $V_B[n]$ if $V_B[n]$ is less than TH, and zero otherwise.
0b111	Less-than, count.	$V_T[n]$ is 1 if $V_B[n]$ is less than TH, and zero otherwise.

The size of the TH field is IMPLEMENTATION DEFINED.

R<sub>QMYLF</sub>

When [threshold extension](#) is not implemented, the Effective values of {TC, TH} are {0b000, 0}.

R<sub>MHHLN</sub>

When the [edge-detect extension](#) is implemented and the [threshold extension](#) is implemented by an event counter <n>, the event counter implements an additional control field, TE:

- The *edge condition* is controlled by TE and TC.
- If TE is 0b1, [edge counting](#) for event counter <n> is [enabled](#).
- If TE is 0b0, edge counting for event counter <n> is [disabled](#).

TE is a 1-bit field.

**TE, bits [0]**

Threshold Edge. With TC[2:0], defines the edge condition.

TC[2:0]	TE	Description
X	0b0	Edge counting is <a href="#">disabled</a> .
0b000	0b1	Reserved.
0b001	0b1	Equal to not-equal (EQ → NE).
0b010	0b1	Equal to/from not-equal (EQ ↔ NE).

TC[2:0]	TE	Description
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).

R<sub>WRXCK</sub>

When the [edge-detect extension](#) is not implemented, the Effective value of TE is 0.

R<sub>MYCSJ</sub>

When the [edge-detect extension](#) is implemented and the [threshold extension](#) is not implemented by an event counter <n>, the event counter implements a control field, TE.

- The *edge condition* is controlled by TE.
- 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*.

TE is a 2-bit field.

1	0
TE	

**TE, bits [1:0]**

Threshold Edge. Defines the edge condition.

TE	Description
0b00	Edge counting is <a href="#">disabled</a> .
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).

I<sub>YDXYW</sub>

The following combination is reserved:

- {TC[1:0], TE} is {0b00, 1}.

The effect of setting {TC, TE} to a reserved value is CONSTRAINED UNPREDICTABLE.

R<sub>SGCNP</sub>

The [threshold condition](#) for event counter <n>, denoted by the value  $C_T[n]$ , is defined as follows:

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

R<sub>NJHCK</sub>

When the [threshold extension](#) is implemented, the [edge condition](#) for event counter <n>, denoted by the value  $C_E[n]$ , is defined as follows:

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

**R<sub>LWKFP</sub>** When the **threshold extension** is not implemented, the **threshold condition** for event counter <n>, denoted by the value  $C_T[n]$ , is defined as follows:

$$C_T[n] = (V_B[n] \neq 0), \quad \text{if the threshold extension is not implemented.}$$

**R<sub>CTLDZ</sub>** When the **threshold extension** is not implemented, the **edge condition** for event counter <n>, denoted by the value  $C_E[n]$ , is defined as follows:

$$\text{edge condition} = C_E[n] = \begin{cases} \text{FALSE}, & \text{if TE} = 0b00 \text{ (disabled).} \\ C_P[n] \wedge \neg C_T[n], & \text{if TE} = 0b01 \text{ (falling edge).} \\ C_P[n] \neq C_T[n], & \text{if TE} = 0b10 \text{ (both edges).} \\ \neg C_P[n] \wedge C_T[n], & \text{if TE} = 0b11 \text{ (rising edge).} \end{cases}$$

**R<sub>ZSYKS</sub>** The result of applying the **threshold condition** for event counter <n>, denoted by value  $V_T[n]$ , is defined as follows:

$$V_T[n] = \begin{cases} V_B[n], & \text{if TC}[0] = 0b0 \text{ and } C_T[n] \text{ is TRUE.} \\ 1, & \text{if TC}[0] = 0b1 \text{ and } C_T[n] \text{ is TRUE.} \\ 0, & \text{otherwise.} \end{cases}$$

**R<sub>PYXQJ</sub>** The result of applying the **edge condition** for event counter <n>, denoted by the value  $V_E[n]$ , is defined as follows:

$$V_E[n] = \begin{cases} 1, & \text{if } C_E[n] \text{ is TRUE.} \\ 0, & \text{otherwise.} \end{cases}$$

**R<sub>FTDXM</sub>** The value of  $V[n]$  for event counter <n> is defined as follows:

$$V[n] = \begin{cases} V_B[n], & \text{if TC} = 0b000 \text{ and TH} = 0 \text{ (disabled).} \\ V_T[n], & \text{if (TC} \neq 0b000 \text{ or TH} \neq 0) \text{ and TE} = 0b0 \text{ (threshold).} \\ V_E[n], & \text{if TE} = 0b1 \text{ (edge).} \end{cases}$$

**R<sub>ZZZTY</sub>** When the **threshold extension** is not implemented, the value of  $V[n]$  for event counter <n> is defined as follows:

$$V[n] = \begin{cases} V_B[n], & \text{if TE} = 0b00 \text{ (disabled).} \\ V_E[n], & \text{if TE} \neq 0b00 \text{ (edge).} \end{cases}$$

**I<sub>JZZBZ</sub>** **R<sub>ZSYKS</sub>** and **R<sub>PYXQJ</sub>** describe the following permitted combinations for an event counter <n>:

**Table 3.6: Count value when TE is 0**

TC[0]	If $C_T[n]$ is TRUE, count by	If $C_T[n]$ is FALSE, count by
0	$V_B[n]$	0
1	1	0

Table 3.7: Count value when TE is 1

TC[0]	If $C_E[n]$ is TRUE, count by	If $C_E[n]$ is FALSE, count by
1	1	0

### 3.7.1 Pseudocode

I<sub>szscq</sub>

The operation of the [threshold condition](#) and [edge condition](#) is described by CountValue():

---

```

// CountValue()
// =====
// Implements the PMU threshold function.
// Returns the value to increment an 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.
// 'TE' is zero if the edge detect extension is not implemented.

integer PMUCountValue(integer Vb, integer TH, bits(3) TC, bit TE)
    // Check if disabled. Note that this function will return the value of Vb when
    // the control register fields are all zero, even without this check.
    if TC == '000' && TH == 0 && TE == '0' then
        boolean Vb;

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

    integer Vn;
    if TE == '1' then
        // Edge condition
        constant boolean Cp = PMULastThresholdValue[n];
        boolean Ce;
        integer Ve;
        case TC<1:0> of
            when '10' Ce = (Cp != Ct); // Both edges
            when 'x1' Ce = (!Cp && Ct); // Single edge
            otherwise Unreachable();
        Ve = (if Ce then 1 else 0);
        Vn = Ve;
    else
        // Threshold condition
        integer Vt;
        if TC<0> == '0' then
            Vt = (if Ct then Vb else 0);
        else
            Vt = (if Ct then 1 else 0);
        Vn = Vt;

    PMULastThresholdValue[n] = Ct;

    return Vn;

```

---

## 3.8 Reusable event filter definitions

- I<sub>FQPMH</sub>** 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<sub>JMSQC</sub>** 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<sub>CBWPY</sub>** If non-invasive debug of an *operating state* is prohibited, then **R<sub>VHTGC</sub>** 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](#).

### 3.8.1 Security operating state filtering

- I<sub>GHXKJ</sub>** For a PMU monitoring an agent that implements multiple Security states or processes operations attributable to multiple 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<sub>JYHZZ</sub>** 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<sub>CSLXT</sub>** Each Security state or PAS corresponds to an *operating state*.
- The State match controls bit assignments are:

3	2	1	0
RL	RT	NS	S

#### RL, bit [3]

Realm state.

#### When Realm state is implemented

Configures matching Realm state.

RL	Description
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.

RT	Description
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.

NS	Description
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.

S	Description
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.

### 3.8.2 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.

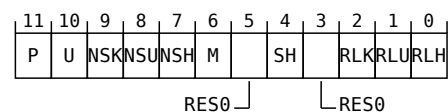
$I_{TTVKD}$

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:



#### P, bit [11]

EL1 filtering. Controls counting events attributable to EL1.

P	Description
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.

U	Description
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.

When P == 0:

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

When P == 1:

NSK	Description
0b0	Events attributable to Non-secure EL1 are not counted.
0b1	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.

When U == 0:

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

When U == 1:

NSU	Description
0b0	Events attributable to Non-secure EL0 are not counted.
0b1	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.

NSH	Description
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.

When P == 0:

M	Description
0b0	This bit has no effect on filtering of events.
0b1	Events attributable to EL3 are not counted.

When P == 1:

M	Description
0b0	Events attributable to EL3 are not counted.
0b1	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.

When NSH == 0:

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

When NSH == 1:

SH	Description
0b0	This bit has no effect on filtering of events.
0b1	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.

When P == 0:

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

When P == 1:

RLK	Description
0b0	Events attributable to Realm EL1 are not counted.
0b1	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.

When U == 0:

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

When U == 1:

RLU	Description
0b0	Events attributable to Realm EL0 are not counted.
0b1	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.

When NSH == 0:

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

When NSH == 1:

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

#### Otherwise

Reserved. This bit is RES0.

### 3.8.3 VMID filtering

I<sub>BJXRK</sub>

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<sub>GKDMV</sub>

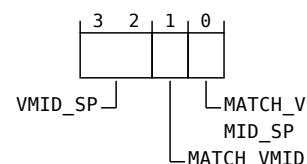
Each VMID space corresponds to an **operating state**.

R<sub>SBCLF</sub>

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:



#### VMID\_SP, bits [3:2]

Virtual Machine Identifier space. Selects the VMID space to match.

VMID_SP	Description
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.

MATCH_VMID	Description
0b0	VMID is ignored.
0b1	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.

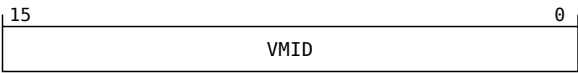
MATCH_VMID_SP	Description
0b0	VMID_SP is ignored. Monitor events with any VMID space.
0b1	Only count events or monitor characteristics attributable to a Virtual Machine ID in the VMID space specified by VMID_SP.

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:



**VMID, bits [15:0]**

Virtual Machine Identifier. Selects the VMID to match.

Ignored if MATCH\_VMID is 0b0.

**3.8.4 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](#).

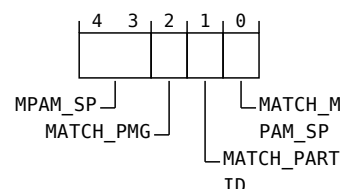


RJTGBW

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:



#### MPAM\_SP, bits [4:3]

Partition Identifier space. Selects the PARTID space to match.

MPAM_SP	Description
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.

MATCH_PMG	Description
0b0	PMG is ignored.
0b1	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.

MATCH_PARTID	Description
0b0	PARTID is ignored.
0b1	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.

MATCH_MPAM_SP	Description
0b0	MPAM_SP is ignored. Monitor events with any PARTID space.
0b1	Only count events or monitor characteristics attributable to a Partition ID in the PARTID space specified by MPAM_SP.

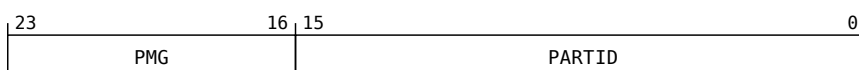
This bit is RES0 and might be not implemented if MPAM\_SP is RES0 or not implemented.

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:



#### 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 System Component Specification; Memory System Resource Partitioning and Monitoring \(MPAM\), for A-profile architecture \[2\]](#).

## 3.9 Snapshot extension

R <sub>WGXRN</sub>	When <b>FEAT_CSPMU_SS</b> is implemented, the PMU supports a snapshot function.
R <sub>PDXMS</sub>	If <b>FEAT_CSPMU_SS</b> is implemented then <b>PMCFGR.SS</b> reads as 0b1.
R <sub>DSDDX</sub>	<p>If <b>FEAT_CSPMU_SS</b> is implemented then the PMU includes the following registers:</p> <ul style="list-style-type: none"> <li>An IMPLEMENTATION DEFINED number of <i>snapshot value registers</i>, <b>PMSVR&lt;n&gt;</b>. If <b>FEAT_CSPMU_EXT64</b> is implemented, each <i>snapshot value register</i> is a 64-bit register. Otherwise, each <i>snapshot value register</i> is a 32-bit register.</li> <li>The <i>snapshot control register</i>, <b>PMSSCR</b>. <b>PMSSCR</b> is a 64-bit register.</li> <li>An IMPLEMENTATION DEFINED number of optional <i>overflow status saved value registers</i>, <b>PMOVSSR&lt;n&gt;</b>. If <b>FEAT_CSPMU_EXT64</b> is implemented, each <i>overflow status saved value register</i> is a 64-bit register. Otherwise, each <i>overflow status saved value register</i> is a 32-bit register, but pairs of <i>overflow status saved value registers</i> might be accessible as a single 64-bit register. See also <b>R<sub>ZNTBP</sub></b>.</li> <li>An optional snapshot reset register, <b>PMSSRR</b>. <b>PMSSRR</b> is a 64-bit register.</li> </ul>
R <sub>SCWKJ</sub>	<p>In some implementations, when <b>FEAT_CSPMU_EXT32</b> is implemented, part of the <i>snapshot control register</i> is replaced by a separate Snapshot Status Register, <b>PMSSSR</b>. This includes the <i>No capture</i> bit, <b>PMSSSR.NC</b>.</p> <p>When <b>PMSSSR</b> is implemented:</p> <ul style="list-style-type: none"> <li><b>PMSSSR</b> is a 32-bit register, implemented as an alias for <b>PMSVR&lt;n&gt;</b>, where <i>n</i> is IMPLEMENTATION DEFINED.</li> <li><b>PMSSCR</b> is the 32-bit <i>snapshot capture register</i>.</li> </ul>
R <sub>DNJSY</sub>	On a <i>Capture event</i> , the PMU writes snapshot values to the <i>snapshot value registers</i> , and then sets <b>PMSSCR.NC</b> to zero to indicate a successful capture.
I <sub>NNVZK</sub>	<p>The snapshot values typically include:</p> <ul style="list-style-type: none"> <li>The event monitors, <b>PMEVCNTR&lt;m&gt;</b>.</li> <li><b>PMCCNTR</b>, if implemented.</li> <li><b>PMSSSR</b>, if implemented.</li> <li>The overflow status flags, captured into <b>PMOVSSR&lt;n&gt;</b>, if implemented.</li> <li>Any additional IMPLEMENTATION DEFINED syndrome information captured by the PMU.</li> </ul> <p>Up-to 128 32-bit values or sixty-four 64-bit values can be captured.</p>
I <sub>SSNNY</sub>	<b>FEAT_CSPMU_SS</b> might not capture all the implemented monitors, due to the limited number of snapshot value registers and the need to capture <b>PMSSSR</b> (if implemented), <b>PMOVSSR&lt;n&gt;</b> (if implemented), and/or any other IMPLEMENTATION DEFINED capture information.
R <sub>CFXNM</sub>	The mapping of snapshot values to the <i>snapshot value registers</i> , is IMPLEMENTATION DEFINED.
I <sub>QZHVb</sub>	When <b>FEAT_CSPMU_EXT64</b> is implemented, <b>PMOVSSR&lt;n&gt;</b> is not an array and is referred to as <b>PMOVSSR</b> .
I <sub>FPFVC</sub>	Implementation of the <b>PMOVSSR</b> register is deprecated.
R <sub>BXVMT</sub>	<p>When <b>FEAT_CSPMU_EXT32</b> is implemented, it is IMPLEMENTATION DEFINED and recommended that all of the following apply:</p> <ul style="list-style-type: none"> <li>Any 64-bit snapshot value is mapped to a pair of 32-bit <i>snapshot value registers</i>, <b>PMSVR&lt;n&gt;</b> and <b>PMSVR&lt;n+1&gt;</b>, where <i>n</i> is even.</li> <li>This pair of registers is treated as a 64-bit register mapped to a doubleword-aligned pair of adjacent 32-bit locations.</li> </ul>
I <sub>TRJQW</sub>	When implemented, each <b>PMOVSSR&lt;m&gt;</b> is an alias for <b>PMSVR&lt;n&gt;</b> , where <i>n</i> is IMPLEMENTATION DEFINED.

Chapter 3. Extensions  
3.9. Snapshot extension

R <sub>CVTFP</sub>	When <b>PMSSRR</b> is implemented, for each monitor $n$ , if $n$ is less than 64 and <b>PMSSRR</b> [ $n$ ] is 0b1 then <b>PMEVCNTR</b> < $n$ > and <b>PMOVS</b> [ $n$ ] are reset to zero after a <b>Capture event</b> .
R <sub>NBXXGD</sub>	When <b>PMSSRR</b> is implemented, the <b>overflow status saved value registers</b> are implemented for at least the first 64 monitors.
R <sub>ZXXZYK</sub>	<p>A <i>Capture event</i> is generated by one of:</p> <ul style="list-style-type: none"><li>• Assertion of an IMPLEMENTATION DEFINED snapshot request signal, <b>PMUSNAPSHOTREQ</b>.</li><li>• Writing 0b1 to <b>PMSSCR</b>.SS or <b>PMSSCR</b>.SS.</li></ul>

## 3.10 Trace generation extension

<code>R_LBLQK</code>	<p>When <code>FEAT_CSPMU_TRO</code> is implemented, the PMU can generate a trace of events or event monitor data.</p> <p>When <code>FEAT_CSPMU_TRO</code> is implemented, the PMU includes a control, <code>PMCR.TRO</code>, to enable the tracing of events or event monitor data.</p>
<code>R_SRKVT</code>	If the PMU implements <code>FEAT_CSPMU_TRO</code> then <code>PMCFGR.TRO</code> reads as 0b1.
<code>R_ZNVML</code>	Trace is not generated for prohibited events. See <a href="#">2.6 Security</a> .
<code>R_NVRCD</code>	The details of any trace generated by the PMU are IMPLEMENTATION DEFINED.

## 3.11 Export extension

- $R_{TYMMH}$  When [FEAT\\_CSPMU\\_EX](#) is implemented, the PMU includes a control, [PMCR.X](#), for the export of events to external monitoring agents to provide triggering information.
- $R_{JSMMJ}$  If the PMU implements [FEAT\\_CSPMU\\_EX](#) then [PMCFGR.EX](#) reads as 0b1.
- $R_{YJZSC}$  Prohibited events are not exported. See [2.6 Security](#).

## 3.12 Dual-page extension

R <sub>FJVQZ</sub>	<p>When <a href="#">FEAT_CSPMU_DUALPAGE</a> is implemented, the PMU is programmed through two pages of registers.</p> <p>When <a href="#">FEAT_CSPMU_DUALPAGE</a> is not implemented, the PMU is programmed through a single page of registers.</p>
R <sub>XZWHB</sub>	<p>The following are <a href="#">Page 1</a> registers:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMEVCNTR&lt;n&gt;</a>.</li> <li>• <a href="#">PMOVSLR&lt;m&gt;</a>.</li> <li>• <a href="#">PMOVSSET&lt;m&gt;</a>.</li> </ul>
R <sub>MBLCQ</sub>	<p>If <a href="#">FEAT_CSPMU_EXT64</a> is implemented, the following is a <a href="#">Page 1</a> register:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMOVS</a>.</li> </ul>
R <sub>PVRYN</sub>	<p>If <a href="#">FEAT_CSPMU_CCNTR</a> is implemented, the following is a <a href="#">Page 1</a> register:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMCCNTR</a>.</li> </ul>
R <sub>MYXPD</sub>	<p>If <a href="#">FEAT_CSPMU_SS</a> is implemented, the following are <a href="#">Page 1</a> registers:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMSVR&lt;n&gt;</a>.</li> <li>• <a href="#">PMOVSSR&lt;n&gt;</a>.</li> <li>• <a href="#">PMSSSR</a>.</li> </ul>
R <sub>HXZHB</sub>	<p>The following registers are both <a href="#">Page 0</a> and <a href="#">Page 1</a> registers:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMIIDR</a>.</li> <li>• <a href="#">PMDEVAFF</a>.</li> <li>• <a href="#">PMDEVID</a>.</li> <li>• <a href="#">PMPIDR0</a>.</li> <li>• <a href="#">PMPIDR1</a>.</li> <li>• <a href="#">PMPIDR2</a>.</li> <li>• <a href="#">PMPIDR3</a>.</li> <li>• <a href="#">PMPIDR4</a>.</li> <li>• <a href="#">PMPIDR5</a>.</li> <li>• <a href="#">PMPIDR6</a>.</li> <li>• <a href="#">PMPIDR7</a>.</li> <li>• <a href="#">PMCIDR0</a>.</li> <li>• <a href="#">PMCIDR1</a>.</li> <li>• <a href="#">PMCIDR2</a>.</li> <li>• <a href="#">PMCIDR3</a>.</li> </ul> <p>The following registers are both <a href="#">Page 0</a> and <a href="#">Page 1</a> registers. When <a href="#">FEAT_CSPMU_DUALPAGE</a> is implemented, these registers do not have the same value in both <a href="#">Page 0</a> and <a href="#">Page 1</a> views:</p> <ul style="list-style-type: none"> <li>• <a href="#">PMCFGR</a>.</li> <li>• <a href="#">PMDEVARCH</a>.</li> <li>• <a href="#">PMDEVTYPE</a>.</li> </ul>
R <sub>FQFQP</sub>	<p>For each IMPLEMENTATION DEFINED register, it is IMPLEMENTATION DEFINED whether the register is a <a href="#">Page 0</a> register, a <a href="#">Page 1</a> register, or both.</p>
R <sub>MLQHD</sub>	<p>All PMU registers that are not <a href="#">Page 1</a> registers are <a href="#">Page 0</a> registers.</p>
R <sub>RSCXF</sub>	<p>When <a href="#">FEAT_CSPMU_DUALPAGE</a> is implemented:</p> <ul style="list-style-type: none"> <li>• <a href="#">Page 0</a> registers are memory-mapped at architecturally-defined offsets from the IMPLEMENTATION DEFINED <a href="#">Page 0</a> base addresses.</li> </ul>

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

R <sub>BVDMH</sub>	When <a href="#">FEAT_CSPMU_DUALPAGE</a> is not implemented, all <a href="#">Page 0</a> and <a href="#">Page 1</a> registers are memory-mapped at architecturally-defined offsets from an IMPLEMENTATION DEFINED base address.
R <sub>JYWDL</sub>	Base addresses must be aligned to a multiple of 4KB.
I <sub>BKZLQ</sub>	Arm recommends that base addresses are aligned to a multiple of 64KB.
S <sub>TCPZJ</sub>	<a href="#">FEAT_CSPMU_DUALPAGE</a> allows a Hypervisor to provide different accessibility to the <a href="#">Page 0</a> and <a href="#">Page 1</a> registers to a Guest operating system.  For example, the Guest can read performance monitors directly from the <a href="#">Page 1</a> registers, but writes to <a href="#">Page 1</a> and any accesses to <a href="#">Page 0</a> registers generate a stage 2 translation fault and can be emulated by the Hypervisor.
R <sub>QCGTS</sub>	If <a href="#">FEAT_CSPMU_DUALPAGE</a> is implemented then this is identified through the unique <a href="#">PMDEVARCH</a> values for the two pages.
I <sub>CQFCY</sub>	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 <sub>MZWTV</sub>	<a href="#">FEAT_CSPMU_DUALPAGE</a> does not provide any mechanism for the <a href="#">Page 1</a> 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 <a href="#">Page 1</a> view.



### 3.13 Observability and access control extension

I<sub>LSRMY</sub>

**FEAT\_CSPMU\_ACR** provides programmable [authentication controls](#) for privileged software.

R<sub>QCSPZ</sub>

When **FEAT\_CSPMU\_ACR** 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 OPTIONAL access control for Non-secure accesses to the PMU.

When register access is disabled, all PMU registers other than CoreSight management registers are RAZ/WI. For the CoreSight management registers, 0xFA8 to 0xFFC, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by **PMSCR**.

- If *message-signaled interrupts* are implemented, a control the message-signaled interrupt physical address space.

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

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

I<sub>XJBRQ</sub>

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<sub>BGGSHZ</sub>

When **FEAT\_CSPMU\_ACR** 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.
- An OPTIONAL access control for Secure, Non-secure, and Realm accesses to the PMU. When register access is disabled, all PMU registers other than CoreSight management registers are RAZ/WI. For the CoreSight management registers, 0xFA8 to 0xFFC, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by **PMROOTCR**.

When this control is implemented, the access control in **PMSCR** is not implemented. Otherwise, if the access control in **PMSCR** is implemented, then it also applies to Realm accesses.

When Realm or Non-secure access is enabled, *message-signaled interrupts* are always Non-secure.

**PMROOTCR** is only accessible to Root state.

I<sub>KGGXD</sub>

The state of the observability and access controls at reset is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing the PMU.

S<sub>XRWTZ</sub>

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<sub>HVCBH</sub>

It is IMPLEMENTATION DEFINED which of the following apply:

- [PMSCR](#) and [PMROOTCR](#) are registers in Page 0 of the PMU. [Chapter 4 Programmers' Model](#) defines 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.

S<sub>KKWKG</sub>

[FEAT\\_CSPMU\\_ACR](#) 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 [FEAT\\_CSPMU\\_ACR](#) is implemented.
- When implemented, whether the [FEAT\\_CSPMU\\_ACR](#) 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

This section is *normative*.

I<sub>BWWWJ</sub>

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.

I<sub>FRBDS</sub>

The programmers' model has two main variants, [FEAT\\_CSPMU\\_EXT64](#) and [FEAT\\_CSPMU\\_EXT32](#):

- When the *64-bit programmers' model extension*, [FEAT\\_CSPMU\\_EXT64](#), is implemented, the structure and layout of registers is suited to implementations with a native 64-bit interface.
- When [FEAT\\_CSPMU\\_EXT32](#) is implemented, the structure and layout of registers is suited to implementations that support a 32-bit Execution state.

I<sub>HVTKD</sub>

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](#).

## 4.1 Memory-mapped registers summary

R<sub>LLXMB</sub>

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** [3.4 Fixed-function cycle counter extension](#).

**CS** CoreSight registers.

**MG**

[3.5 Monitor group extension](#).

**MSI**

Message-signaled interrupts. See [R<sub>FDLKQ</sub>](#).

**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 [3.13 Observability and access control extension](#).

**SS** [3.9 Snapshot extension](#).

**TZ** Security extensions. The component incorporating the PMU implements a Secure operating state, or processes Secure operations. See [2.6 Security](#) and [3.13 Observability and access control extension](#).

I<sub>MJGKQ</sub>

The following tables show the view of the memory-mapped registers:

- When the 64-bit programmers' model extension is not implemented:
  - [Table 4.1](#).
  - [Table 4.2](#).
  - [Table 4.3](#).
- When the 64-bit programmers' model extension is implemented:
  - [Table 4.4](#).
  - [Table 4.5](#).
  - [Table 4.6](#).

## 4.2 When the 64-bit programmers' model extension is not implemented

When [FEAT\\_CSPMU\\_EXT32](#) is implemented:

- The PMU registers are a mix of 32 bit and 64 bit registers.
- If all monitor values are 32 bits or fewer, then [PMEVCNTR<n>](#) are 32-bit registers. Otherwise, [PMEVCNTR<n>](#) are 64-bit registers.
- Up to 128 64-bit or 256 32-bit monitors can be implemented. See [R\\_HKCFV](#).
- The [PMCNTEN](#), [PMINTEN](#), and [PMOVS](#) registers are not directly accessible. These registers can only be accessed through SET and CLR registers.

### 4.2.1 When the dual-page extension is not implemented

**Table 4.1: Memory-mapped register map**

Offset	Access	Version	Register	Description
$0 \times 000 + 4 \times n$	R/W	32b	<a href="#">PMEVCNTR&lt;n&gt;</a>	Event Count Register <n> (up-to 32 bits)
$0 \times 000 + 8 \times n$	R/W	64b	<a href="#">PMEVCNTR&lt;n&gt;</a> [31:0]	Event Count Register <n> (up-to 64 bits), bits[31:0]
$0 \times 004 + 8 \times n$	R/W	64b	<a href="#">PMEVCNTR&lt;n&gt;</a> [63:32]	Event Count Register <n> (up-to 64 bits), bits[63:32]
$0 \times 07C$	R/W	32b, CC	<a href="#">PMCCNTR</a>	Cycle Count Register (up-to 32 bits)
$0 \times 0F8$	R/W	64b, CC	<a href="#">PMCCNTR</a> [31:0]	Cycle Count Register (up-to 64 bits), bits[31:0]
$0 \times 0FC$	R/W	64b, CC	<a href="#">PMCCNTR</a> [63:32]	Cycle Count Register (up-to 64 bits), bits[63:32]
$0 \times 200 + 4 \times n$	R/W		<a href="#">PMIMPDEF&lt;n+32&gt;</a>	IMPLEMENTATION DEFINED Register <n+32>
$0 \times 400 + 4 \times n$	RO or R/W		<a href="#">PMEVTYPEPER&lt;n&gt;</a>	Event Type Select Register <n>
$0 \times 47C$	RO or R/W	CC	<a href="#">PMCCFILTR</a>	Cycle Counter Filter Register
$0 \times 600 + 4 \times n$	RO	SS	<a href="#">PMSVR&lt;n&gt;</a>	Saved Value Register <n>
$0 \times 600 + \text{IMPDEF}$	RO	SS	<a href="#">PMSSSR</a>	Snapshot Status Register
$0 \times 600$	RO	SS	<a href="#">PMOVSSR&lt;n&gt;</a>	Overflow Status Snapshot Register <n>
$\rightarrow + \text{IMPDEF} + 4 \times n$				
$0 \times 800 + 4 \times n$	RO or R/W		<a href="#">PMEVFILT2R&lt;n&gt;</a>	Event Filter 2 Register <n>
$0 \times A00 + 4 \times n$	RO or R/W		<a href="#">PMEVFILTR&lt;n&gt;</a>	Event Filter Register <n>
$0 \times C00 + 4 \times n$	R/W		<a href="#">PMCNTENSET&lt;m&gt;</a>	Count Enable Set Register <m>
$0 \times C20 + 4 \times n$	R/W		<a href="#">PMCNTENCLR&lt;m&gt;</a>	Count Enable Clear Register <m>
$0 \times C40 + 4 \times n$	R/W		<a href="#">PMINTENSET&lt;m&gt;</a>	Interrupt Enable Set Register <m>
$0 \times C60 + 4 \times n$	R/W		<a href="#">PMINTENCLR&lt;m&gt;</a>	Interrupt Enable Clear Register <m>
$0 \times C80 + 4 \times n$	R/W		<a href="#">PMOVSCLR&lt;m&gt;</a>	Overflow Flag Status Clear Register <m>
$0 \times CC0 + 4 \times n$	R/W		<a href="#">PMOVSSET&lt;m&gt;</a>	Overflow Flag Status Set Register <m>
$0 \times CE0 + 4 \times n$	RO	MG	<a href="#">PMCGCR&lt;n&gt;</a>	Counter Group Configuration Register <n>
$0 \times D80 + 4 \times n$	R/W		<a href="#">PMIMPDEF&lt;n&gt;</a>	IMPLEMENTATION DEFINED Register <n>
$0 \times E00$	RO		<a href="#">PMCFGR</a>	Configuration Register
$0 \times E04$	R/W		<a href="#">PMCR</a>	Control Register
$0 \times E08$	RO		<a href="#">PMIIDR</a>	Implementation Identification Register
$0 \times E30$	WO	SS	<a href="#">PMSSCR</a>	Snapshot Capture Register ( <a href="#">PMSSSR</a> implemented)
$0 \times E30$	R/W	SS	<a href="#">PMSSCR</a> [31:0]	Snapshot Control Register ( <a href="#">PMSSSR</a> not implemented), bits[31:0]

Offset	Access	Version	Register	Description
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

#### 4.2.2 Page 0, when the dual-page extension is implemented

Table 4.2: Memory-mapped register map

Offset	Access	Version	Register	Description
0x200+4×n	R/W		PMIMPDEF<n+32>	IMPLEMENTATION DEFINED Register <n+32>
0x400+4×n	RO or R/W		PMEVTYPEPER<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>

Offset	Access	Version	Register	Description
0xC00+4×n	R/W		PMCNTENSET<m>	Count Enable Set Register <m>
0xC20+4×n	R/W		PMCNTENCLR<m>	Count Enable Clear Register <m>
0xC40+4×n	R/W		PMINTENSET<m>	Interrupt Enable Set Register <m>
0xC60+4×n	R/W		PMINTENCLR<m>	Interrupt Enable Clear Register <m>
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		PMIDR	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

### 4.2.3 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]
0x200+4×n	R/W		PMIMPDEF<n+32>	IMPLEMENTATION DEFINED Register <n+32>
0x600+4×n	RO	SS	PMSVR<n>	Saved Value Register <n>
0x600+IMPDEF	RO	SS	PMSSSR	Snapshot Status Register
0x600	RO	SS	PMOVSSR<n>	Overflow Status Snapshot Register <n>
↔+IMPDEF+4×n				
0xC80+4×n	R/W		PMOVSCLR<m>	Overflow Flag Status Clear Register <m>
0xCC0+4×n	R/W		PMOVSSET<m>	Overflow Flag Status Set Register <m>
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



## 4.3 When the 64-bit programmers' model extension is implemented

R<sub>LXDDK</sub>

When FEAT\_CSPMU\_EXT64 is implemented:

- All PMU registers are 64 bits, other than the CoreSight management registers (when implemented) excluding PMDEVAFF. This includes all monitor value registers, PMEVCNTR<n>. However, it is not required that monitor values are 64 bits.
- Up to 64 monitors are implemented.
- The PMCNTEN, PMINTEN, and PMOVS registers are implemented. These provide direct read/write access to the monitor control and status bits.
- The PMCNTEN, PMINTEN, and PMOVS registers are implemented with direct read/write access to the monitor control and status bits.
- If the message-signaled interrupt registers are implemented, then PMIRQCR1 and PMIRQCR2 are accessed as a single 64-bit register, PMIRQCR12.

### 4.3.1 When the dual-page extension is not implemented

Table 4.4: Memory-mapped register map

Offset	Access	Size	Version	Register	Description
0x000+8×n	R/W	64	CC	PMEVCNTR<n>	Event Count Register <n>
0x0F8	R/W	64		PMCCNTR	Cycle Count Register
0x200+8×n	R/W	64		PMIMPDEF<n+16>	IMPLEMENTATION DEFINED Register <n+16>
0x400+8×n	RO or R/W	64		PMEVTYPEPER<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	RO	64	SS	PMOVSSR	Overflow Status Snapshot Register
→+IMPDEF					
0x800+8×n	RO or R/W	64	MG	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		PMOVSSSET	Overflow Flag Status Set Register
0xCE0+8×n	RO	64		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		PMIDR	Implementation Identification Register
0xE10	R/W	64		PMCR	Control Register
0xE30	R/W	64	SS	PMSSCR	Snapshot Control Register

Offset	Access	Size	Version	Register	Description
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

### 4.3.2 Page 0, when the dual-page extension is implemented

Table 4.5: Memory-mapped register map

Offset	Access	Size	Version	Register	Description
0x200+8× <i>n</i>	R/W	64		PMIMPDEF< <i>n</i> +16>	IMPLEMENTATION DEFINED Register < <i>n</i> +16>
0x400+8× <i>n</i>	RO or R/W	64		PMEVTYPEPER< <i>n</i> >	Event Type Select Register < <i>n</i> >
0x4F8	RO or R/W	64	CC	PMCCFILTR	Cycle Counter Filter Register
0x800+8× <i>n</i>	RO or R/W	64		PMEVFILT2R< <i>n</i> >	Event Filter 2 Register < <i>n</i> >
0xA00+8× <i>n</i>	RO or R/W	64		PMEVFILTR< <i>n</i> >	Event Filter Register < <i>n</i> >
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× <i>n</i>	RO	64	MG	PMCGCR< <i>n</i> >	Counter Group Configuration Register < <i>n</i> >
0xD80+8× <i>n</i>	R/W	64		PMIMPDEF< <i>n</i> >	IMPLEMENTATION DEFINED Register < <i>n</i> >
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

### 4.3.3 Page 1, when the dual-page extension is implemented

Table 4.6: 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
0x200+8×n	R/W	64		PMIMPDEF<n+16>	IMPLEMENTATION DEFINED Register <n+16>
0x600+8×n	RO	64	SS	PMSVR<n>	Saved Value Register <n>
0x600	RO	64	SS	PMOVSSR	Overflow Status Snapshot Register
↔+IMPDEF					
0xC80	R/W	64		PMOVSCLR	Overflow Flag Status Clear Register
0xC90	R/W	64		PMOVVS	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

Offset	Access	Size	Version	Register	Description
0xFD0	RO	32	CS	<a href="#">PMPIDR4</a>	Peripheral Identification Register 4
0xFD4	RO	32	CS	<a href="#">PMPIDR5</a>	Peripheral Identification Register 5
0xFD8	RO	32	CS	<a href="#">PMPIDR6</a>	Peripheral Identification Register 6
0xFDC	RO	32	CS	<a href="#">PMPIDR7</a>	Peripheral Identification Register 7
0xFE0	RO	32	CS	<a href="#">PMPIDR0</a>	Peripheral Identification Register 0
0xFE4	RO	32	CS	<a href="#">PMPIDR1</a>	Peripheral Identification Register 1
0xFE8	RO	32	CS	<a href="#">PMPIDR2</a>	Peripheral Identification Register 2
0xFEC	RO	32	CS	<a href="#">PMPIDR3</a>	Peripheral Identification Register 3
0xFF0	RO	32	CS	<a href="#">PMCIDR0</a>	Component Identification Register 0
0xFF4	RO	32	CS	<a href="#">PMCIDR1</a>	Component Identification Register 1
0xFF8	RO	32	CS	<a href="#">PMCIDR2</a>	Component Identification Register 2
0xFFC	RO	32	CS	<a href="#">PMCIDR3</a>	Component Identification Register 3

## 4.4 Register descriptions

This section describes the CoreSight PMU registers. [4.1 Memory-mapped registers summary](#) lists these registers in offset order.

### 4.4.1 PMAUTHSTATUS, Authentication Status Register

The PMAUTHSTATUS characteristics are:

#### Purpose

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

#### Configuration

For each debug level, if a CoreSight or IMPLEMENTATION DEFINED authentication interface is present that controls that debug level, these fields represent the debug levels currently permitted by that interface.

Otherwise, the permitted debug for each debug level is fixed and the field represents that behavior.

#### Attributes

PMAUTHSTATUS is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions

31	8	7	6	5	4	3	2	1	0
RES0								SNID	NSID

#### Bits [31:8]

Reserved, RES0.

#### SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

SNID	Meaning
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:

- When Security Extensions are not implemented and the PMU is Non-secure, access to this field is RAZ/WI.
- Otherwise, access to this field is RO.

#### SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

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

All other values are reserved.

Access to this field is RAZ/WI.

### NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

NSNID	Meaning
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:

- When Security Extensions are not implemented and the PMU is Secure, access to this field is RAZ/WI.
- Otherwise, access to this field is RO.

### NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

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

All other values are reserved.

Access to this field is RAZ/WI.

## Accessing PMAUTHSTATUS

Accesses to this register use the following encodings:

### When **FEAT\_CSPMU\_EXT** is implemented

Accessible at offset 0xFB8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.2 PMCCFILTR, Cycle Counter Filter Register

The PMCCFILTR characteristics are:

#### Purpose

Configures the cycle counter.

#### Configuration

This register is present only when [FEAT\\_CSPMU\\_CCNTR](#) is implemented. Otherwise, direct accesses to PMCCFILTR are RES0.

#### Attributes

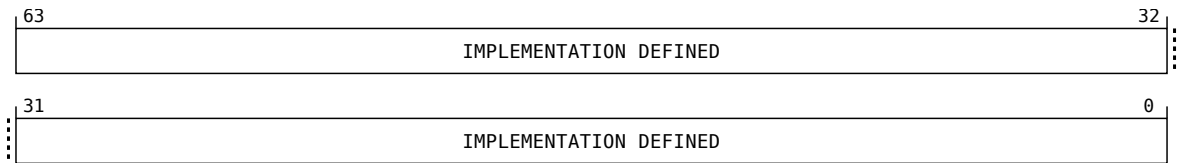
PMCCFILTR is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

#### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



#### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

#### Otherwise:



#### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

#### Accessing PMCCFILTR

If PMCCFILTR is read/write and [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMCCFILTR is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

When [FEAT\\_CSPMU\\_CCNTR](#) is implemented and [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset 0x47C from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.



- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented and [FEAT\\_CSPMU\\_EXT64](#) is implemented**

Accessible at offset 0x4F8 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

### 4.4.3 PMCCNTR, Cycle Count Register

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](#).

#### Configuration

This register is present only when [FEAT\\_CSPMU\\_CCNTR](#) is implemented. Otherwise, direct accesses to PMCCNTR are RES0.

#### Attributes

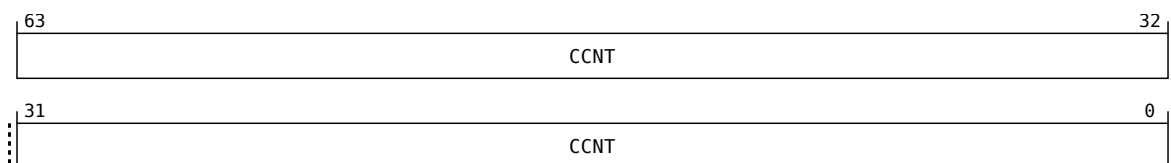
PMCCNTR is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

#### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

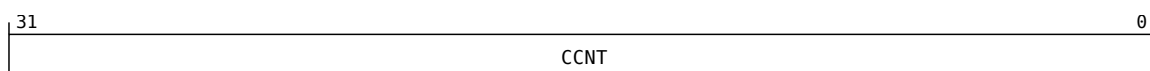


#### CCNT, bits [63:0]

Cycle Counter.

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

#### Otherwise:



#### CCNT, bits [31:0]

Cycle Counter.

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

#### Accessing PMCCNTR

If [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMCCNTR is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

When [MAX\\_COUNTER\\_SIZE](#) <= 32, [FEAT\\_CSPMU\\_CCNTR](#) is implemented, and [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset 0x07C from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

When **MAX\_COUNTER\_SIZE > 32**, **FEAT\_CSPMU\_CCNTR** is implemented, and **FEAT\_CSPMU\_EXT32** is implemented

Accessible at offset 0x0F8 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

When **FEAT\_CSPMU\_CCNTR** is implemented and **FEAT\_CSPMU\_EXT64** is implemented

Accessible at offset 0x0F8 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.4 PMCFGR, Configuration Register

The PMCFGR characteristics are:

##### Purpose

Describes the performance monitor.

##### Configuration

There are no configuration notes.

##### Attributes

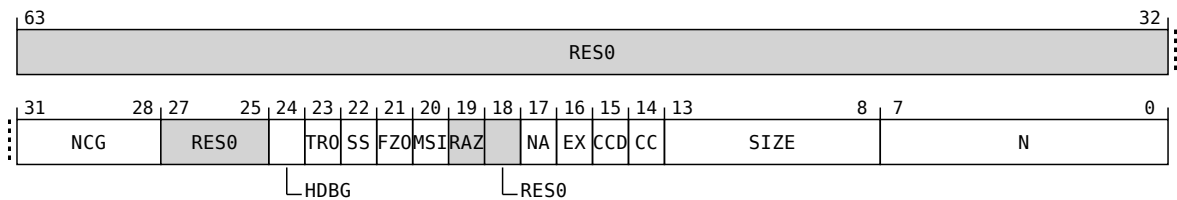
PMCFGR is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### Bits [63:32]

Reserved, RES0.

##### 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:

- [PMEVCNTR<m×32>](#), meaning there are at most 32 monitors per group, if there are 2 monitor groups.
- [PMEVCNTR<m×16>](#), meaning there are at most 16 monitors per group, if there are 3 or 4 monitor groups.
- [PMEVCNTR<m×8>](#), meaning there are at most 8 monitors per group, if there are between 5 and 8 monitor groups.
- [PMEVCNTR<m×4>](#), meaning there are at most 4 monitors per group, if there are more than 8 monitor groups.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### Bits [27:25]

Reserved, RES0.

#### HDBG, bit [24]

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

Halt-on-debug feature supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

Access to this field is RO.

Otherwise:

Reserved, RES0.

#### TRO, bit [23]

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

Trace features supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

The nature of any supported trace features are IMPLEMENTATION DEFINED.

Access to this field is RO.

Otherwise:

Reserved, RES0.

#### SS, bit [22]

Snapshot supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SS	Meaning
0b0	Snapshot mechanism not supported. The locations 0x600-0x7FC and 0xE30-0xE3C are IMPLEMENTATION DEFINED.
0b1	Snapshot mechanism supported. <b>PMSVR&lt;n&gt;</b> and <b>PMSSCR</b> 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.

Access to this field is RO.

#### FZO, bit [21]

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

Freeze-on-overflow supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FZO	Meaning
0b0	Freeze-on-overflow mechanism not supported. <b>PMCR.FZO</b> is RES0.
0b1	Freeze-on-overflow mechanism supported. <b>PMCR.FZO</b> is RW.

Access to this field is RO.

Otherwise:

Reserved, RES0.

**MSI, bit [20]**

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

Message-signaled interrupts (MSI) supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MSI	Meaning
0b0	MSI not supported. <b>PMIRQCR0</b> , <b>PMIRQCR12</b> , and <b>PMIRQSR</b> are reserved.
0b1	MSI supported. <b>PMIRQCR0</b> and <b>PMIRQCR12</b> are used to configure the MSI, and <b>PMIRQSR</b> 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.

Access to this field is RO.

Otherwise:

Reserved, RES0.

**Bit [19]**

Reserved, RAZ.

**Bit [18]**

Reserved, RES0.

**NA, bit [17]**

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

No write access when running.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NA	Meaning
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 <b>STOP</b> state.

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

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**EX, bit [16]**

When **FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0:

Export supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EX	Meaning
0b0	Export is not supported. <b>PMCR.X</b> is RES0.
0b1	Export is supported. <b>PMCR.X</b> is read/write.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**CCD, bit [15]**

When **FEAT\_CSPMU\_CCNTR** is implemented and (**FEAT\_CSPMU\_DUALPAGE** is not implemented or accessed in Page 0):

Cycle counter has pre-scale.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

Access to this field is RO.

**Otherwise:**

Reserved, RAZ.

**CC, bit [14]**

Dedicated Cycle counter implemented as **PMEVCNTR31**.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

Access to this field is RO.

#### SIZE, bits [13:8]

Monitor size. The size of the largest implemented monitor.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIZE	Meaning
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.

Access to this field is RO.

#### N, bits [7:0]

Number of monitors, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

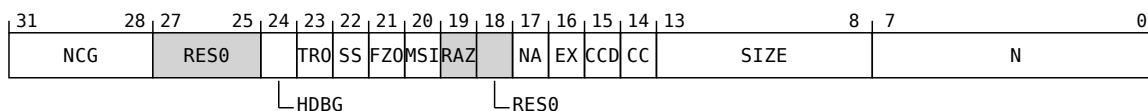
N	Meaning
0x00	1 monitor.
0x01	2 monitors.
0x02..0x3F	Number of monitors, minus one.

If PMCFGR.CC == 1, PMEVCNTR31, the cycle counter, is one of the (N+1) monitors. For example, if PMCFGR.N == 0x00 and PMCFGR.CC == 1, 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.

Access to this field is RO.

#### Otherwise:





#### 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:

- `PMEVCNTR<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.
- `PMEVCNTR<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.
- `PMEVCNTR<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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

#### Bits [27:25]

Reserved, RES0.

#### HDBG, bit [24]

When `FEAT_CSPMU_DUALPAGE` is not implemented or accessed in Page 0:

Halt-on-debug feature supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

Access to this field is RO.

Otherwise:

Reserved, RES0.

#### TRO, bit [23]

When `FEAT_CSPMU_DUALPAGE` is not implemented or accessed in Page 0:

Trace features supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

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

The nature of any supported trace features are IMPLEMENTATION DEFINED.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**SS, bit [22]**

Snapshot supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SS	Meaning
0b0	Snapshot mechanism not supported. The locations 0x600-0x7FC and 0xE30-0xE3C are IMPLEMENTATION DEFINED.
0b1	Snapshot mechanism supported. <a href="#">PMSVR&lt;n&gt;</a> and <a href="#">PMSSCR</a> 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.

Access to this field is RO.

**FZO, bit [21]**

When [FEAT\\_CSPMU\\_DUALPAGE](#) is not implemented or accessed in Page 0:

Freeze-on-overflow supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FZO	Meaning
0b0	Freeze-on-overflow mechanism not supported. <a href="#">PMCR.FZO</a> is RES0.
0b1	Freeze-on-overflow mechanism supported. <a href="#">PMCR.FZO</a> is RW.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**MSI, bit [20]**

When [FEAT\\_CSPMU\\_DUALPAGE](#) is not implemented or accessed in Page 0:

Message-signaled interrupts (MSI) supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MSI	Meaning
0b0	MSI not supported. <a href="#">PMIRQCR0</a> , <a href="#">PMIRQCR1</a> , <a href="#">PMIRQCR2</a> , and <a href="#">PMIRQSR</a> are reserved.
0b1	MSI supported. <a href="#">PMIRQCR0</a> , <a href="#">PMIRQCR1</a> , and <a href="#">PMIRQCR2</a> are used to configure the MSI, and <a href="#">PMIRQSR</a> 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.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Bit [19]**

Reserved, RAZ.

**Bit [18]**

Reserved, RES0.

**NA, bit [17]**

**When [FEAT\\_CSPMU\\_DUALPAGE](#) is not implemented or accessed in Page 0:**

No write access when running.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NA	Meaning
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 <a href="#">STOP</a> state.

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

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**EX, bit [16]**

**When [FEAT\\_CSPMU\\_DUALPAGE](#) is not implemented or accessed in Page 0:**

Export supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EX	Meaning
0b0	Export is not supported. <a href="#">PMCR.X</a> is RES0.
0b1	Export is supported. <a href="#">PMCR.X</a> is read/write.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**CCD, bit [15]**

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented and ([FEAT\\_CSPMU\\_DUALPAGE](#) is not implemented or accessed in Page 0):**

Cycle counter has pre-scale.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCD	Meaning
0b0	Cycle counter only ever counts every cycle. <a href="#">PMCR.D</a> is RES0.
0b1	Cycle counter can count every 64th cycle. <a href="#">PMCR.D</a> is read/write.

Access to this field is RO.

**Otherwise:**

Reserved, RAZ.

**CC, bit [14]**

Dedicated Cycle counter implemented as PMEVCNTR31.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CC	Meaning
0b0	If PMEVCNTR31 is implemented, it is a normal monitor. <a href="#">PMCR.C</a> , <a href="#">PMCR.D</a> and <a href="#">PMCFGR.CCD</a> are RES0.
0b1	PMEVCNTR31 is implemented and is a dedicated cycle counter. <a href="#">PMCR.C</a> is write-only.

Access to this field is RO.

**SIZE, bits [13:8]**

Monitor size. The size of the largest implemented monitor.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIZE	Meaning
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.

Access to this field is RO.

### N, bits [7:0]

Number of monitors, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

N	Meaning	Applies when
0x00	1 monitor.	
0x01	2 monitors.	
0x02..0x7F	Number of monitors, minus one.	
0x80..0xFF	Number of monitors, minus one.	<a href="#">MAX_COUNTER_SIZE</a> <= 32

If PMCFGR.CC == 1, PMEVCNTR31, the cycle counter, is one of the (N+1) monitors. For example, if PMCFGR.N == 0x00 and PMCFGR.CC == 1, 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.

Access to this field is RO.

## Accessing PMCFGR

Accesses to this register use the following encodings:

### When [FEAT\\_CSPMU\\_EXT](#) is implemented

Accessible at offset 0xE00 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.5 PMCGCR<n>, Counter Group Configuration Registers, n = 0 - 3

The PMCGCR<n> characteristics are:

##### Purpose

Describes the performance monitor.

##### Configuration

This register is present only when  $\text{UInt}(\text{CS\_PMU.PMCFGR.NCG}) \neq 0$ , monitor groups are implemented. Otherwise, direct accesses to PMCGCR<n> are RES0.

##### Attributes

PMCGCR<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	56	55	48	47	40	39	32
CG7NC	CG6NC	CG5NC	CG4NC				
31	24	23	16	15	8	7	0
CG3NC	CG2NC	CG1NC	CG0NC				

##### CG<m>NC, bits [8m+7:8m], for m = 7 to 0

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](#).

Otherwise:

31	24	23	16	15	8	7	0
CG3NC	CG2NC	CG1NC	CG0NC				

##### CG<m>NC, bits [8m+7:8m], for m = 3 to 0

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](#).

##### Accessing PMCGCR<n>

Accesses to this register use the following encodings:

When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0 \times \text{CE0} + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.

- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

**When [FEAT\\_CSPMU\\_EXT64](#) is implemented**

Accessible at offset  $0 \times \text{CE0} + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.6 PMCIDR0, Component Identification Register 0

The PMCIDR0 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

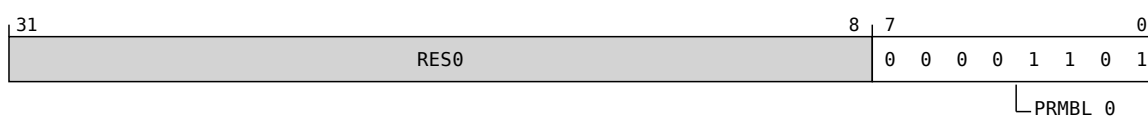
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMCIDR0 are RES0.

#### Attributes

PMCIDR0 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions



#### Bits [31:8]

Reserved, RES0.

#### PRMBL\_0, bits [7:0]

Component identification preamble, segment 0.

Reads as 0x0D

Access to this field is RO.

#### Accessing PMCIDR0

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFF0 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.



### 4.4.7 PMCIDR1, Component Identification Register 1

The PMCIDR1 characteristics are:

**Purpose**

Provides discovery information about the component.

**Configuration**

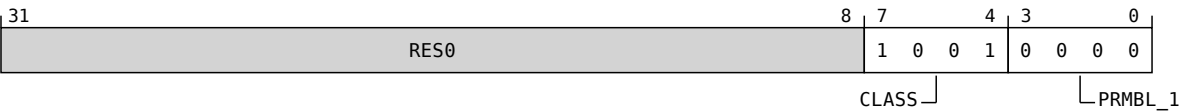
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMCIDR1 are RES0.

**Attributes**

PMCIDR1 is a 32-bit register.

This register is part of the CS\_PMU block.

**Field descriptions**



**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class.

CLASS	Meaning
0b1001	CoreSight peripheral.

Other values are defined by the CoreSight Architecture.

Access to this field is RO.

**PRMBL\_1, bits [3:0]**

Component identification preamble, segment 1.

Reads as 0b0000

Access to this field is RO.

**Accessing PMCIDR1**

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFF4 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.

- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.8 PMCIDR2, Component Identification Register 2

The PMCIDR2 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

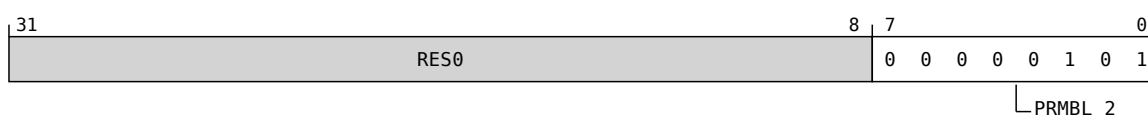
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMCIDR2 are RES0.

#### Attributes

PMCIDR2 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions



#### Bits [31:8]

Reserved, RES0.

#### PRMBL\_2, bits [7:0]

Component identification preamble, segment 2.

Reads as 0x05

Access to this field is RO.

#### Accessing PMCIDR2

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFF8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.9 PMCIDR3, Component Identification Register 3

The PMCIDR3 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMCIDR3 are RES0.

#### Attributes

PMCIDR3 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions



#### Bits [31:8]

Reserved, RES0.

#### PRMBL\_3, bits [7:0]

Component identification preamble, segment 3.

Reads as 0xB1

Access to this field is RO.

#### Accessing PMCIDR3

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFFC from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.10 PMCNTEN, Count Enable Set Register

The PMCNTEN characteristics are:

##### Purpose

Enable monitors.

##### Configuration

This register is present only when [FEAT\\_CSPMU\\_EXT64](#) is implemented. Otherwise, direct accesses to PMCNTEN are RES0.

##### Attributes

PMCNTEN is a 64-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<n> , bits [n], for n = 63 to 0**

Event counter enable for [PMEVCNTR<n>](#).

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> is disabled.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> is enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMCNTEN.P31 is the event counter enable for [PMCCNTR](#).

##### Accessing PMCNTEN

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset 0xC10 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.11 PMCNTENCLR<m>, Count Enable Clear Registers, m = 0 - 7

The PMCNTENCLR<m> characteristics are:

##### Purpose

Disable monitors.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMCNTENCLR not PMCNTENCLR0.

##### Configuration

There are no configuration notes.

##### Attributes

PMCNTENCLR<m> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	:
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	:
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	:
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	:

**P<n> , bits [n], for n = 63 to 0**

[PMEVCNTR<n>](#) disable. On writes, allows software to disable [PMEVCNTR<n>](#). On reads, returns the [PMEVCNTR<n>](#) enable status.

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> disabled.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMCNTENCLR.P31 allows software to disable [PMCCNTR](#) and query the [PMCCNTR](#) enable status.

Access to this field is WIC.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<p> , bits [p], for p = 31 to 0**

[PMEVCNTR<n>](#) disable, where  $n == p + 32m$ . On writes, allows software to disable [PMEVCNTR<n>](#). On reads, returns the [PMEVCNTR<n>](#) enable status.

P<p>	Meaning
0b0	PMEVCNTR<n> disabled.
0b1	PMEVCNTR<n> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMCNTENCLR0.P31 allows software to disable [PMCCNTR](#) and query the [PMCCNTR](#) enable status.

Access to this field is WIC.

### Accessing PMCNTENCLR<m>

Accesses to this register use the following encodings:

#### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset 0xC20 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset 0xC20 + (4 \* m) from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.12 PMCNTESET<m>, Count Enable Set Registers, m = 0 - 7

The PMCNTESET<m> characteristics are:

##### Purpose

Enable monitors.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMCNTESET not PMCNTESET0.

##### Configuration

There are no configuration notes.

##### Attributes

PMCNTESET<m> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	⋮
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	

**P<n> , bits [n], for n = 63 to 0**

[PMEVCNTR<n>](#) enable. On writes, allows software to enable [PMEVCNTR<n>](#). On reads, returns the [PMEVCNTR<n>](#) enable status.

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> disabled.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMCNTESET.P31 allows software to enable [PMCCNTR](#) and query the [PMCCNTR](#) enable status.

Access to this field is W1S.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<p> , bits [p], for p = 31 to 0**

[PMEVCNTR<n>](#) enable, where  $n == p + 32m$ . On writes, allows software to enable [PMEVCNTR<n>](#). On reads, returns the [PMEVCNTR<n>](#) enable status.



P<p>	Meaning
0b0	PMEVCNTR<n> disabled.
0b1	PMEVCNTR<n> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMCNTENSET0.P31 allows software to enable [PMCCNTR](#) and query the [PMCCNTR](#) enable status.

Access to this field is WIS.

### Accessing PMCNTENSET<m>

Accesses to this register use the following encodings:

#### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0xC00 + (4 * m)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset  $0xC00$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

4.4.13 PMCR, Control Register

The PMCR characteristics are:

Purpose

Main control register for the performance monitors.

Configuration

There are no configuration notes.

Attributes

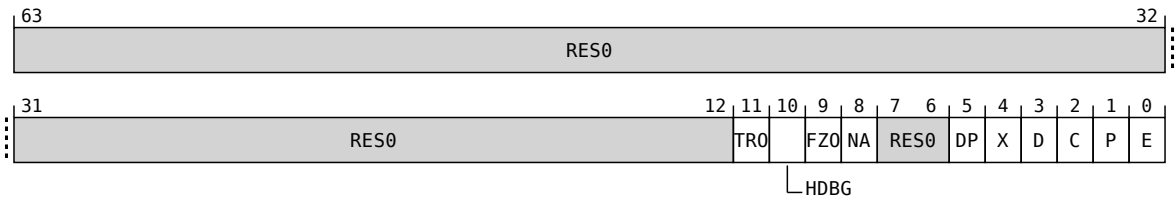
PMCR is a:

- 64-bit register when FEAT\_CSPMU\_EXT64 is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

Field descriptions

When FEAT\_CSPMU\_EXT64 is implemented:



Bits [63:12]

Reserved, RES0.

TRO, bit [11]

When FEAT\_CSPMU\_TRO is implemented:

Trace enable. Enable trace features.

TRO	Meaning
0b0	Trace disabled.
0b1	Trace enabled.

Otherwise:

Reserved, RES0.

HDBG, bit [10]

When FEAT\_CSPMU\_HDBG is implemented:

Halt-on-debug. Stops events being counted when the affine PE or agent is in a halted state such as Debug state.

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

**Otherwise:**

Reserved, RES0.

**FZO, bit [9]**

**When [FEAT\\_CSPMU\\_FZO](#) is implemented:**

Freeze-on-overflow. Stop monitors on overflow.

FZO	Meaning
0b0	Do not freeze on overflow.
0b1	Monitors do not count when <a href="#">PMOVS</a> is nonzero.

**Otherwise:**

Reserved, RES0.

**NA, bit [8]**

**When `CS_PMU.PMCFGR.NA == 1`:**

Not accessible. Indicates the monitors are read-only.

NA	Meaning
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.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Bits [7:6]**

Reserved, RES0.

**DP, bit [5]**

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented:**

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by <a href="#">PMCCNTR</a> is not affected by this mechanism.
0b1	Cycle counting by <a href="#">PMCCNTR</a> is disabled in prohibited regions. Prohibited regions are IMPLEMENTATION DEFINED regions where event counting is prohibited.

**Otherwise:**

Reserved, RES0.

**X, bit [4]**

**When `FEAT_CSPMU_EX` is implemented:**

Export enable. Permit events to be exported to another debug device, such as a trace unit, over an event bus.

<b>X</b>	<b>Meaning</b>
0b0	Export of events is disabled.
0b1	Export of events is enabled.

This field 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, RES0.

**D, bit [3]**

**When `FEAT_CSPMU_CCNTR` is implemented:**

Cycle counter divider.

<b>D</b>	<b>Meaning</b>
0b0	When enabled, the cycle counter counts every clock cycle.
0b1	When enabled, the cycle counter counts once every 64 clock cycles.

**Otherwise:**

Reserved, RES0.

**C, bit [2]**

**When `FEAT_CSPMU_CCNTR` is implemented:**

Cycle counter reset.

<b>C</b>	<b>Meaning</b>
0b0	Write is ignored.
0b1	Reset the cycle counter to zero.

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

---

**Note**

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

---

**Otherwise:**

Reserved, RES0.

**P, bit [1]**

**When the PMU is not an *Activity Monitor Unit* (AMU):**

Monitor reset.

P	Meaning
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 field is write-only and reads-as-zero.

Otherwise:

Reserved, RAZ/WI.

E, bit [0]

When the PMU is not an AMU:

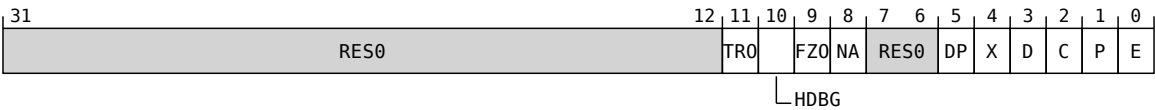
Enable.

E	Meaning
0b0	All monitors are disabled.
0b1	All monitors are enabled by <a href="#">PMCNTENSET</a> .

Otherwise:

Reserved, RAZ/WI.

Otherwise:



Bits [31:12]

Reserved, RES0.

TRO, bit [11]

When [FEAT\\_CSPMU\\_TRO](#) is implemented:

Trace enable. Enable trace features.

TRO	Meaning
0b0	Trace disabled.
0b1	Trace enabled.

Otherwise:

Reserved, RES0.

HDBG, bit [10]

**When [FEAT\\_CSPMU\\_HDBG](#) is implemented:**

Halt-on-debug. Stops events being counted when the affine PE or agent is in a halted state such as Debug state.

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

**Otherwise:**

Reserved, RES0.

**FZO, bit [9]**

**When [FEAT\\_CSPMU\\_FZO](#) is implemented:**

Freeze-on-overflow. Stop monitors on overflow.

FZO	Meaning
0b0	Do not freeze on overflow.
0b1	Monitors do not count when <a href="#">PMOVS</a> is nonzero.

**Otherwise:**

Reserved, RES0.

**NA, bit [8]**

**When [CS\\_PMU.PMCFGR.NA == 1](#):**

Not accessible. Indicates the monitors are read-only.

NA	Meaning
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.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Bits [7:6]**

Reserved, RES0.

**DP, bit [5]**

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented:**

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by <a href="#">PMCCNTR</a> is not affected by this mechanism.

DP	Meaning
0b1	Cycle counting by <a href="#">PMCCNTR</a> is disabled in prohibited regions. Prohibited regions are IMPLEMENTATION DEFINED regions where event counting is prohibited.

**Otherwise:**

Reserved, RES0.

**X, bit [4]**

**When [FEAT\\_CSPMU\\_EX](#) is implemented:**

Export enable. Permit events to be exported to another debug device, such as a trace unit, over an event bus.

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

This field 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, RES0.

**D, bit [3]**

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented:**

Cycle counter divider.

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

**Otherwise:**

Reserved, RES0.

**C, bit [2]**

**When [FEAT\\_CSPMU\\_CCNTR](#) is implemented:**

Cycle counter reset.

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

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

**Note**

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

**Otherwise:**

Reserved, RES0.

**P, bit [1]**

**When the PMU is not an AMU:**

Monitor reset.

P	Meaning
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 field is write-only and reads-as-zero.

**Otherwise:**

Reserved, RAZ/WI.

**E, bit [0]**

**When the PMU is not an AMU:**

Enable.

E	Meaning
0b0	All monitors are disabled.
0b1	All monitors are enabled by <a href="#">PMCNTENSET</a> .

**Otherwise:**

Reserved, RAZ/WI.

## Accessing PMCR

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_EXT32](#) is implemented**

Accessible at offset 0xE04 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.



**When FEAT\_CSPMU\_EXT64 is implemented**

Accessible at offset 0xE10 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.14 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, then the affinity level is 0 and 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, then the affinity level is 1, 2, or 3, 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.

##### Configuration

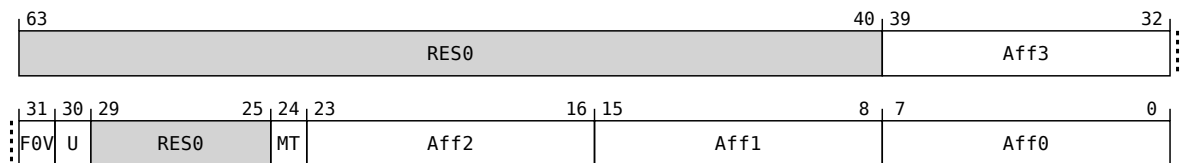
This register is present only when the monitor has affinity with PE. Otherwise, direct accesses to PMDEVAFF are RES0.

##### Attributes

PMDEVAFF is a 64-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### Bits [63:40]

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### F0V, bit [31]

Indicates that the PMDEVAFF.Aff0 field is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F0V	Meaning
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.

Access to this field is RO.

#### U, bit [30]

##### When CS\_PMU.PMDEVAFF.F0V == 1:

Uniprocessor. The MPIDR\_EL1.U field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### Otherwise:

Reserved, UNKNOWN.

#### Bits [29:25]

Reserved, RES0.

#### MT, bit [24]

##### When CS\_PMU.PMDEVAFF.F0V == 1:

Multithreaded. The MPIDR\_EL1.MT field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### Otherwise:

Reserved, UNKNOWN.

#### Aff2, bits [23:16]

##### When affine with a PE at affinity level 2 or with a PE or PEs below affinity level 2:

PE affinity level 2. The MPIDR\_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### When affine with a subset of PEs at affinity level 2 or with a PE or PEs above affinity level 2:

PE affinity level 2. Defines part of the MPIDR\_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff2	Meaning
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.
0bxxxxxx100	PMDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.

Aff2	Meaning
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.
0x80	PE affinity is at level 3.

All other values are reserved.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Aff1, bits [15:8]**

**When affine with a PE at affinity level 1 or with a PE or PEs below affinity level 1:**

PE affinity level 1. The MPIDR\_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**When affine with a subset of PEs at affinity level 1 or with a PE or PEs above affinity level 1:**

PE affinity level 1. Defines part of the MPIDR\_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0x00	PE affinity is above level 2 or a subset of level 2.
0bxxxxxxxx1	PMDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxxxx10	PMDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxxxxx100	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.
0x80	PE affinity is at level 2.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Aff0, bits [7:0]**

**When affine with a PE at affinity level 0:**

PE affinity level 0. The MPIDR\_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**When affine with a subset of PEs at affinity level 0 or with a PE or PEs above affinity level 0:**

PE affinity level 0. Defines part of the MPIDR\_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0x00	PE affinity is above level 1 or a subset of level 1.
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.
0x80	PE affinity is at level 1.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**Accessing PMDEVAFF**

Accesses to this register use the following encodings:

**When the monitor has affinity with PE and FEAT\_CSPMU\_EXT is implemented**

Accessible at offset 0xFA8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.

- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.15 PMDEVARCH, Device Architecture Register

The PMDEVARCH characteristics are:

#### Purpose

Provides discovery information for the component.

#### Configuration

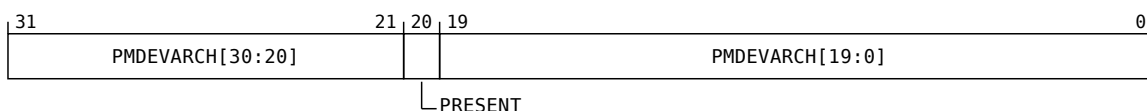
There are no configuration notes.

#### Attributes

PMDEVARCH is a 32-bit register.

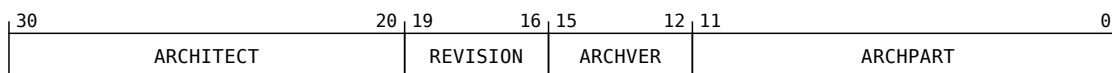
This register is part of the CS\_PMU block.

#### Field descriptions



#### PMDEVARCH, bits [31:21, 19:0]

When PMDEVARCH is implemented:



#### ARCHITECT, bits [30:20]

When Generic CoreSight PMU:

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. Defined values are:

ARCHITECT	Meaning
0b01000111011	JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.

This field reads as 0x23B.

When Generic AMU:

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. Defined values are:

ARCHITECT	Meaning
0b01000111011	JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.

This field reads as 0x23B.

**Otherwise:**

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.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**REVISION, bits [19:16]**

**When Generic CoreSight PMU:**

Revision. Defines the architecture revision of the component.

REVISION	Meaning
0b0000	Revision 0.

All other values are reserved.

Access to this field is RO.

**When Generic AMU:**

Revision. Defines the architecture revision of the component.

REVISION	Meaning
0b0000	Revision 0.

All other values are reserved.

Access to this field is RO.

**Otherwise:**

Defines the architecture revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**ARCHVER, bits [15:12]**

**When Generic CoreSight PMU:**

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0000	Revision 0.

All other values are reserved.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

Access to this field is RO.



#### When Generic AMU:

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0000	Revision 0.

All other values are reserved.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

Access to this field is RO.

#### Otherwise:

Defines the architecture version of the component.

This field has an IMPLEMENTATION DEFINED value.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

Access to this field is RO.

#### ARCHPART, bits [11:0]

#### When Generic CoreSight PMU:

Architecture Part. Defines the architecture of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHPART	Meaning
0xA66	Generic AMU, 64-bit programmers' model extension not implemented.
0xA67	Generic AMU, 64-bit programmers' model extension 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.
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.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

Access to this field is RO.

#### Otherwise:

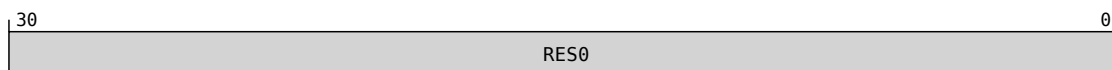
Defines the architecture of the component.

This field has an IMPLEMENTATION DEFINED value.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

Access to this field is RO.

**When PMDEVARCH is not implemented:**



**Bits [30:0]**

Reserved, RES0.

**PRESENT, bit [20]**

DEVARCH present. Defines that PMDEVARCH register is present. Defined values are:

PRESENT	Meaning
0b0	Device Architecture information not present.
0b1	Device Architecture information present.

If PMDEVARCH is not present, the register is RES0.

**Accessing PMDEVARCH**

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFBC from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.16 PMDEVID, Device Configuration Register

The PMDEVID characteristics are:

##### Purpose

Provides discovery information for the component.

##### Configuration

There are no configuration notes.

##### Attributes

PMDEVID is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions

31	IMPLEMENTATION DEFINED	0
----	------------------------	---

##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMDEVID

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT](#) is implemented

Accessible at offset 0xFC8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

4.4.17 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.

Configuration

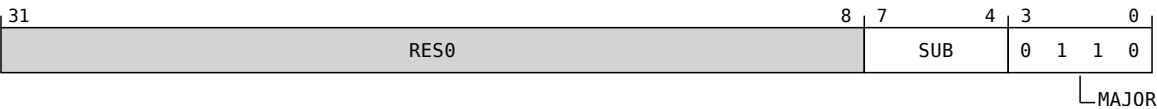
There are no configuration notes.

Attributes

PMDEVTYPE is a 32-bit register.

This register is part of the CS\_PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Component sub-type.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUB	Meaning
0b0000	Other.
0b0001	Associated with a PE.
0b0010	Associated with a DSP.
0b0011	Associated with a Data Engine or coprocessor.
0b0100	Associated with a bus or stimulus derived from bus activity.
0b0101	Associated with a memory management unit conforming to the Arm System MMU architecture.
0b0111	Derived from generic signals.

Access to this field is RO.

MAJOR, bits [3:0]

Component major type.

MAJOR	Meaning
0b0110	Performance monitor.

Other values are defined by the CoreSight Architecture.

Access to this field is RO.

## Accessing PMDEVTYPE

Accesses to this register use the following encodings:

### When **FEAT\_CSPMU\_EXT** is implemented

Accessible at offset 0xFCC from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.18 PMEVCNTR<n>, Event Count Registers, n = 0 - 127

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>](#).

##### Configuration

There are no configuration notes.

##### Attributes

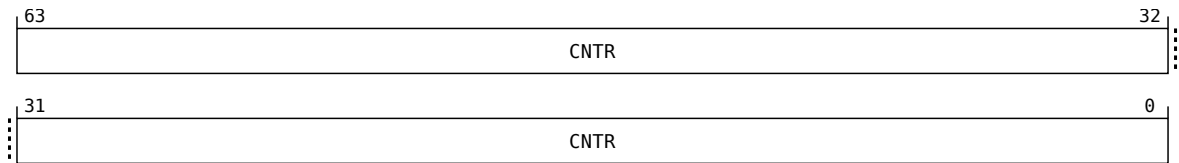
PMEVCNTR<n> is a:

- 64-bit register when [MAX\\_COUNTER\\_SIZE](#) > 32 or [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [MAX\\_COUNTER\\_SIZE](#) > 32 or [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### CNTR, bits [63:0]

Monitor value.

The number of implemented bits for PMEVCNTR<n> is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

##### Otherwise:



##### CNTR, bits [31:0]

Monitor value.

The number of implemented bits for PMEVCNTR<n> is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

##### Accessing PMEVCNTR<n>

If [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMEVCNTR<n> is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

When [MAX\\_COUNTER\\_SIZE](#) <= 32 and [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0 \times 000 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.

- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

**When `MAX_COUNTER_SIZE` > 32 and `FEAT_CSPMU_EXT32` is implemented**

Accessible at offset  $0 \times 000 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

**When `FEAT_CSPMU_EXT64` is implemented**

Accessible at offset  $0 \times 000 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.19 PMEVFILT2R<n>, Event Filter 2 Registers, n = 0 - 127

The PMEVFILT2R<n> characteristics are:

##### Purpose

For performance monitors requiring event selection controls in addition to the [PMEVTYPER<n>](#) and [PMEVFILTR<n>](#) registers.

##### Configuration

There are no configuration notes.

##### Attributes

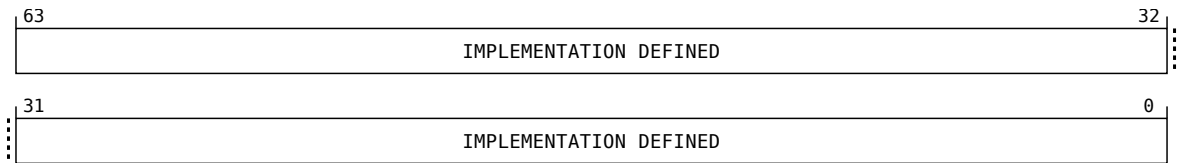
PMEVFILT2R<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

##### Otherwise:



##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMEVFILT2R<n>

If PMEVFILT2R<n> is read/write and [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMEVFILT2R<n> is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0 \times 800 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.



- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

**When [FEAT\\_CSPMU\\_EXT64](#) is implemented**

Accessible at offset  $0 \times 800 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

#### 4.4.20 PMEVFILTR<n>, Event Filter Registers, n = 0 - 127

The PMEVFILTR<n> characteristics are:

##### Purpose

For performance monitors requiring event selection controls in addition to the [PMEVTYPER<n>](#) register.

##### Configuration

There are no configuration notes.

##### Attributes

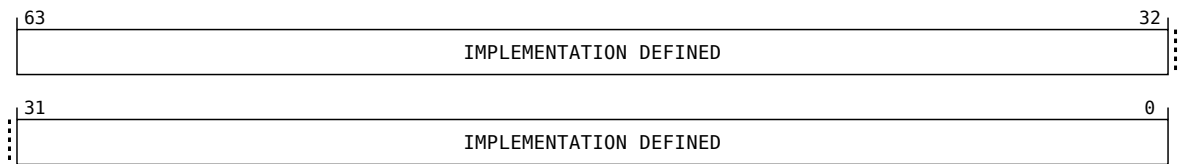
PMEVFILTR<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

##### Otherwise:



##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMEVFILTR<n>

If PMEVFILTR<n> is read/write and [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMEVFILTR<n> is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0xA00 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.

- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

**When FEAT\_CSPMU\_EXT64 is implemented**

Accessible at offset  $0xA00 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

#### 4.4.21 PMEVTYPEPER<n>, Event Type Select Registers, n = 0 - 127

The PMEVTYPEPER<n> characteristics are:

##### Purpose

Configures monitor <n>.

##### Configuration

There are no configuration notes.

##### Attributes

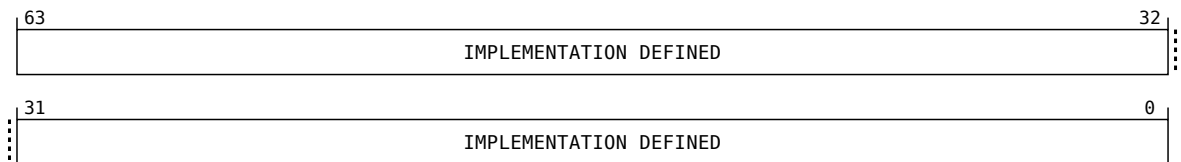
PMEVTYPEPER<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

##### Otherwise:



##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMEVTYPEPER<n>

If PMEVTYPEPER<n> is read/write and [PMCFGR.NA](#) is 1, then it is CONSTRAINED UNPREDICTABLE whether a permitted write to PMEVTYPEPER<n> is ignored when [PMCR.NA](#) == 1.

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0x400 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.

- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

**When FEAT\_CSPMU\_EXT64 is implemented**

Accessible at offset  $0 \times 400 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When Register is read-only, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

#### 4.4.22 PMIIDR, Implementation Identification Register

The PMIIDR characteristics are:

## Purpose

Provides discovery information for the component.

## Configuration

This register is present only when FEAT\_CSPMU\_EXT64 is implemented or (FEAT\_CSPMU\_EXT32 is implemented and an implementation implements PMIIDR). Otherwise, direct accesses to PMIIDR are RES0.

## Attributes

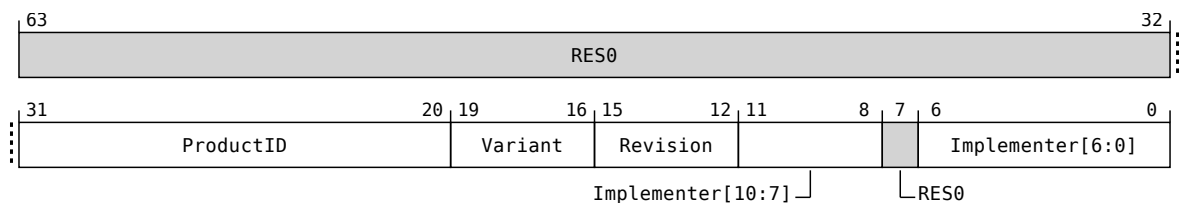
PMIIDR is a:

- 64-bit register when `FEAT_CSPMU_EXT64` is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

## Field descriptions

**When FEAT\_CSPMU\_EXT64 is implemented:**

**Bits [63:32]**

Reserved, RES0.

**ProductID, bits [31:20]**

Part number, bits [11:0]. The part number is selected by the designer of the component.

This field has an IMPLEMENTATION DEFINED value.

Matches the {PMPIDR1.PART\_1, PMPIDR0.PART\_0} fields, if PMPIDR0 and PMPIDR1 are also present.

Access to this field is RO.

**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, `PMIHDR.Variant` and `PMIHDR.Revision` together form the revision number of the component, with `PMIHDR.Variant` being the most significant part and `PMIHDR.Revision` the least significant part. When a component is changed, `PMIHDR.Variant` or `PMIHDR.Revision` is increased to ensure that software can differentiate the different revisions of the component. If `PMIHDR.Variant` is increased then `PMIHDR.Revision` should be set to `0b0000`.

This field has an IMPLEMENTATION DEFINED value.

Matches the **PMPIDR2**.REVISION field, if **PMPIDR2** is also present.

Access to this field is RO.

### 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.

This field has an IMPLEMENTATION DEFINED value.

Matches the [PMPIDR3.REVAND](#) field, if [PMPIDR3](#) is also present.

Access to this field is RO.

### 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 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.

This field has an IMPLEMENTATION DEFINED value.

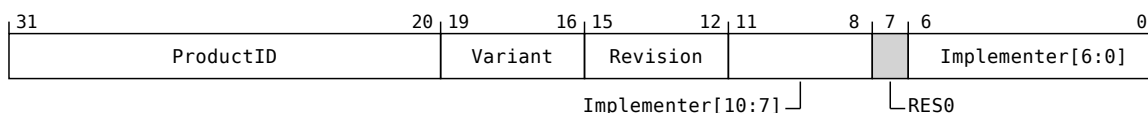
PMIIDR[11:8] matches [PMPIDR4.DES\\_2](#) and PMIIDR[6:0] match the {[PMPIDR2.DES\\_1](#), [PMPIDR1.DES\\_0](#)} fields, if [PMPIDR1](#), [PMPIDR2](#), and [PMPIDR4](#) are also present.

Access to this field is RO.

### Bit [7]

Reserved, RES0.

### Otherwise:



### ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

This field has an IMPLEMENTATION DEFINED value.

Matches the {[PMPIDR1.PART\\_1](#), [PMPIDR0.PART\\_0](#)} fields, if [PMPIDR0](#) and [PMPIDR1](#) are also present.

Access to this field is RO.

#### Variant, bits [19:16]

Component major revision.

Defines either a variant of the component defined by `PMIHDR.ProductID`, or the major revision of the component.

When defining a major revision, `PMIHDR.Variant` and `PMIHDR.Revision` together form the revision number of the component, with `PMIHDR.Variant` being the most significant part and `PMIHDR.Revision` the least significant part. When a component is changed, `PMIHDR.Variant` or `PMIHDR.Revision` is increased to ensure that software can differentiate the different revisions of the component. If `PMIHDR.Variant` is increased then `PMIHDR.Revision` should be set to `0b0000`.

This field has an IMPLEMENTATION DEFINED value.

Matches the `PMPIDR2.REVISION` field, if `PMPIDR2` is also present.

Access to this field is RO.

#### Revision, bits [15:12]

Component minor revision.

When a component is changed:

- If `PMIHDR.Variant` and `PMIHDR.Revision` together form the revision number of the component then:
  - `PMIHDR.Variant` or `PMIHDR.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, `PMIHDR.Revision` is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Matches the `PMPIDR3.REVAND` field, if `PMPIDR3` is also present.

Access to this field is RO.

#### Implementer, bits [11:8, 6:0]

JEDEC-assigned JEP106 identification code of the designer of the component.

`PMIHDR[11:8]` is the JEP106 bank identifier minus 1 and `PMIHDR[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 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 `PMIHDR[11:0]` has the value `0x43B`.

---

Zero is not a valid JEP106 identification code, meaning a value of zero for `PMIHDR` indicates this register is not implemented.

This field has an IMPLEMENTATION DEFINED value.

`PMIHDR[11:8]` matches `PMPIDR4.DES_2` and `PMIHDR[6:0]` match the {`PMPIDR2.DES_1`, `PMPIDR1.DES_0`} fields, if `PMPIDR1`, `PMPIDR2`, and `PMPIDR4` are also present.

Access to this field is RO.



**Bit [7]**

Reserved, RES0.

**Accessing PMIIDR**

Accesses to this register use the following encodings:

**When `FEAT_CSPMU_EXT` is implemented**

Accessible at offset `0xE08` from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.23 PMIMPDEF<n>, IMPLEMENTATION DEFINED Registers, n = 0 - 63

The PMIMPDEF<n> characteristics are:

##### Purpose

IMPLEMENTATION DEFINED extensions.

##### Configuration

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

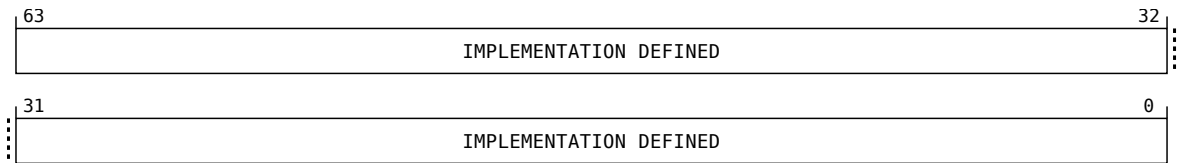
PMIMPDEF<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

##### Otherwise:



##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMIMPDEF<n>

Accesses to this register use the following encodings:

When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0xD80 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

**When `FEAT_CSPMU_EXT64` is implemented**

Accessible at offset  $0xD80 + (8 * n)$  from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

**When `NUM_IMPDEF_REGISTERS` > 32 and `FEAT_CSPMU_EXT32` is implemented**

Accessible at offset  $0x200 + (4 * (n - 32))$  from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

**When `NUM_IMPDEF_REGISTERS` > 16 and `FEAT_CSPMU_EXT64` is implemented**

Accessible at offset  $0x200 + (8 * (n - 16))$  from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.24 PMINTEN, Interrupt Enable Set Register

The PMINTEN characteristics are:

##### Purpose

Enable interrupt on monitor overflow status.

##### Configuration

This register is present only when [FEAT\\_CSPMU\\_EXT64](#) is implemented. Otherwise, direct accesses to PMINTEN are RES0.

##### Attributes

PMINTEN is a 64-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	...
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	...
...	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
...	P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

##### P<n>, bits [n], for n = 63 to 0

Interrupt on overflow status of [PMEVCNTR<n>](#) enable.

P<n>	Meaning
0b0	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> disabled.
0b1	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMINTEN.P31 is the overflow interrupt enable for [PMCCNTR](#).

Access to this field is WIS.

##### Accessing PMINTEN

Accesses to this register use the following encodings:

##### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset 0xC50 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.25 PMINTENCLR<m>, Interrupt Enable Clear Registers, m = 0 - 7

The PMINTENCLR<m> characteristics are:

##### Purpose

Disable interrupt on monitor overflow status.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMINTENCLR not PMINTENCLR0.

##### Configuration

There are no configuration notes.

##### Attributes

PMINTENCLR<m> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	⋮
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	⋮
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	⋮
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	⋮

##### P<n> , bits [n], for n = 63 to 0

Interrupt on overflow status of [PMEVCNTR<n>](#) disable. On writes, allows software to disable the interrupt on overflow status of [PMEVCNTR<n>](#). On reads, returns the interrupt on overflow status of [PMEVCNTR<n>](#) enable status.

P<n>	Meaning
0b0	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> disabled.
0b1	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMINTENCLR.P31 allows software to disable the interrupt on overflow status of [PMCCNTR](#) and query the interrupt on overflow status of [PMCCNTR](#) enable status.

Access to this field is WIC.

##### Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

##### P<p> , bits [p], for p = 31 to 0

Interrupt on overflow status of [PMEVCNTR<n>](#) disable, where  $n == p + 32m$ . On writes, allows software to disable the interrupt on overflow status of [PMEVCNTR<n>](#). On reads, returns the interrupt on overflow status of [PMEVCNTR<n>](#) enable status.

P<p>	Meaning
0b0	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> disabled.
0b1	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;n&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMINTENCLR0.P31 allows software to disable the interrupt on overflow status of [PMCCNTR](#) and query the interrupt on overflow status of [PMCCNTR](#) enable status.

Access to this field is WIC.

### Accessing PMINTENCLR<m>

Accesses to this register use the following encodings:

#### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset  $0xC60$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0xC60 + (4 * m)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.26 PMINTENSET<m>, Interrupt Enable Set Registers, m = 0 - 7

The PMINTENSET<m> characteristics are:

## Purpose

Enable interrupt on monitor overflow status.

If **FEAT\_CSPMU\_EXT64** is implemented, this register is not an array and is referred to as **PMINTENSET** not **PMINTENSET0**.

## Configuration

There are no configuration notes.

## Attributes

PMINTENSET<m> is a:

- 64-bit register when `FEAT_CSPMU_EXT64` is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

## Field descriptions

**When FEAT\_CSPMU\_EXT64 is implemented:**

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<n> , bits [n], for n = 63 to 0**

Interrupt on overflow status of **PMEVCNTR<n>** enable. On writes, allows software to enable the interrupt on overflow status of **PMEVCNTR<n>**. On reads, returns the interrupt on overflow status of **PMEVCNTR<n>** enable status.

<b>P&lt;n&gt;</b>	<b>Meaning</b>
0b0	Interrupt on overflow status of <b>PMEVCNTR&lt;n&gt;</b> disabled.
0b1	Interrupt on overflow status of <b>PMEVCNTR&lt;n&gt;</b> enabled.

If **FEAT\_CSPMU\_CCNTR** is implemented, then **PMINTENSET.P31** allows software to enable the interrupt on overflow status of **PMCCNTR** and query the interrupt on overflow status of **PMCCNTR** enable status.

Access to this field is W1S.

**Otherwise:**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<p> , bits [p], for p = 31 to 0**

Interrupt on overflow status of **PMEVCNTR<n>** enable, where  $n == p + 32m$ . On writes, allows software to enable the interrupt on overflow status of **PMEVCNTR<p>**. On reads, returns the interrupt on overflow status of **PMEVCNTR<p>** enable status.

P<p>	Meaning
0b0	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;p&gt;</a> disabled.
0b1	Interrupt on overflow status of <a href="#">PMEVCNTR&lt;p&gt;</a> enabled.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMINTENSET0.P31 allows software to enable the interrupt on overflow status of [PMCCNTR](#) and query the interrupt on overflow status of [PMCCNTR](#) enable status.

Access to this field is WIS.

### Accessing PMINTENSET<m>

Accesses to this register use the following encodings:

#### When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0xC40 + (4 * m)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset 0xC40 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.



#### 4.4.27 PMIRQCR0, Interrupt Configuration Register 0

The PMIRQCR0 characteristics are:

##### Purpose

Interrupt configuration register.

##### Configuration

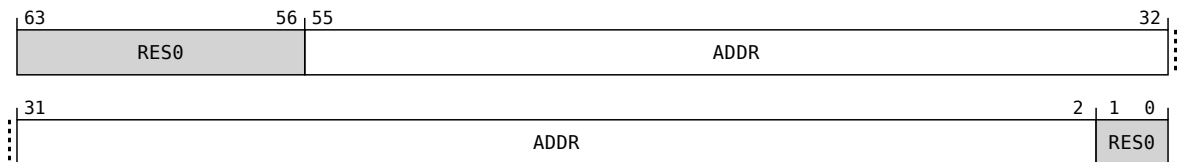
This register is present only when [FEAT\\_CSPMU\\_MSI](#) is implemented. Otherwise, direct accesses to PMIRQCR0 are RES0.

##### Attributes

PMIRQCR0 is a 64-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### Bits [63:56]

Reserved, RES0.

##### ADDR, bits [55:2]

Message Signaled Interrupt address. (PMIRQCR0.ADDR  $\ll$  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.

##### Bits [1:0]

Reserved, RES0.

#### Accessing PMIRQCR0

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xE80 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When ((PMSCR is not implemented && PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure) || (PMSCR is implemented && PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure)) && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)), accesses to this register are RO.
- Otherwise, accesses to this register are RW.

#### 4.4.28 PMIRQCR1, Interrupt Configuration Register 1

The PMIRQCR1 characteristics are:

##### Purpose

Interrupt configuration register.

##### Configuration

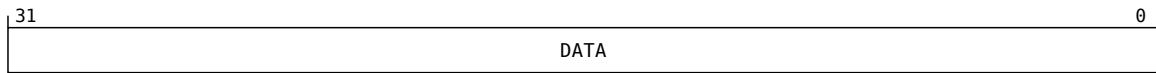
This register is present only when [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT32](#) is implemented. Otherwise, direct accesses to PMIRQCR1 are RES0.

##### Attributes

PMIRQCR1 is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### DATA, bits [31:0]

Payload for the message signaled interrupt.

##### Accessing PMIRQCR1

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT32](#) is implemented**

Accessible at offset 0xE88 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When ((PMSCR is not implemented && PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure) || (PMSCR is implemented && PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure)) && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)), accesses to this register are RO.
- Otherwise, accesses to this register are RW.

## 4.4.29 PMIRQCR2, Interrupt Configuration Register 2

The PMIRQCR2 characteristics are:

### Purpose

Interrupt control and configuration register.

### Configuration

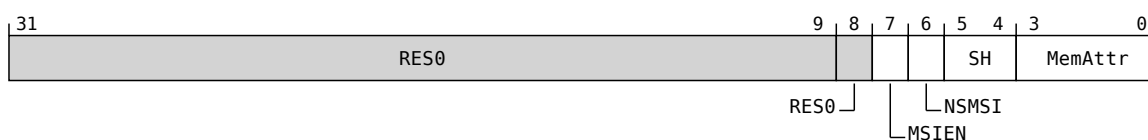
This register is present only when [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT32](#) is implemented. Otherwise, direct accesses to PMIRQCR2 are RES0.

### Attributes

PMIRQCR2 is a 32-bit register.

This register is part of the CS\_PMU block.

### Field descriptions



### Bits [31:9]

Reserved, RES0.

### Bit [8]

**When PMSCR is implemented or PMIRQCR2.NSMSI configures MSIs as Non-secure MSIs:**

Reserved, RES0.

**When the PMU supports *Memory System Resource Partitioning and Monitoring (MPAM)* on message signaled interrupts:**

MPAM Non-secure signaling for Secure MSIs. Defines the MPAM\_NS setting used for Secure message signaled interrupts.

MSI_MPAM_NS	Meaning
0b0	Secure MSIs are issued with MPAM_NS set to 0.
0b1	Secure MSIs are issued with MPAM_NS set to 1.

**Otherwise:**

Reserved, RES0.

### MSIEN, bit [7]

**When the PMU supports disabling message signaled interrupts:**

Message signaled interrupt enable. Enables generation of message signaled interrupts.

MSIEN	Meaning
0b0	Disabled.
0b1	Enabled.

**Otherwise:**

Message signaled interrupts are always enabled.

Reserved, RES0.

**NSMSI, bit [6]**

**When PMSCR is not implemented and the PMU supports configuring the physical address space for message signaled interrupts:**

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

Accessing this field has the following behavior:

- Access to this field is RO if any of the following are true:
  - IsAccessNonSecure(addrdesc)
  - IsAccessRealm(addrdesc)
- Otherwise, access to this field is RW.

**When PMSCR is not implemented:**

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

Reserved, RES0.

**Otherwise:**

Reserved, RES0.

**SH, bits [5:4]**

**When the PMU supports configuring the Shareability domain for message signaled interrupts:**

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
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.

**Otherwise:**

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

Reserved, RES0.

### MemAttr, bits [3:0]

**When the PMU supports configuring the memory type for message signaled interrupts:**

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
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.

#### 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.

Reserved, RES0.

### Accessing PMIRQCR2

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT32](#) is implemented**

Accessible at offset 0xE8C from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When ((PMSCR is not implemented && PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure) || (PMSCR is implemented && PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure)) && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)), accesses to this register are RO.
- Otherwise, accesses to this register are RW.

4.4.30 PMIRQCR12, Interrupt Configuration Register 12

The PMIRQCR12 characteristics are:

Purpose

Interrupt control and configuration register.

Configuration

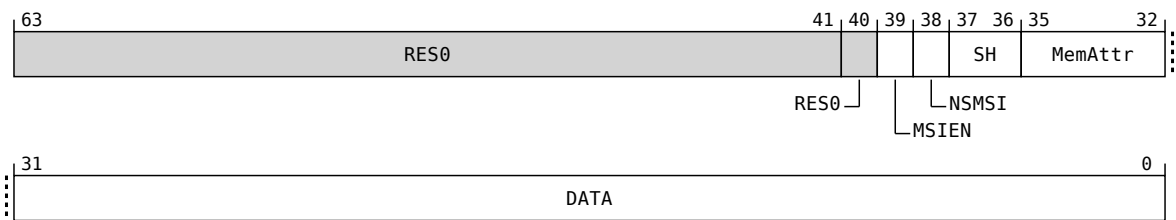
This register is present only when [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT64](#) is implemented. Otherwise, direct accesses to PMIRQCR12 are RES0.

Attributes

PMIRQCR12 is a 64-bit register.

This register is part of the CS\_PMU block.

Field descriptions



Bits [63:41]

Reserved, RES0.

Bit [40]

When PMSCR is implemented or PMIRQCR12.NSMSI configures MSIs as Non-secure MSIs:

Reserved, RES0.

When the PMU supports MPAM on message signaled interrupts:

MPAM Non-secure signaling for Secure MSIs. Defines the MPAM\_NS setting used for Secure message signaled interrupts.

MSI_MPAM_NS	Meaning
0b0	Secure MSIs are issued with MPAM_NS set to 0.
0b1	Secure MSIs are issued with MPAM_NS set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

MSIEN, bit [39]

**When the PMU supports disabling message signaled interrupts:**

Message signaled interrupt enable. Enables generation of message signaled interrupts.

MSIEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Message signaled interrupts are always enabled.

Reserved, RES0.

**NSMSI, bit [38]**

**When PMSCR is not implemented and the PMU supports configuring the physical address space for message signaled interrupts:**

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- Access to this field is RO if any of the following are true:
  - the access is Non-secure
  - the access is Realm
- Otherwise, access to this field is RW.

**When PMSCR is not implemented:**

Non-secure message signaled interrupt.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

Reserved, RES0.

**Otherwise:**

Reserved, RES0.

**SH, bits [37:36]**

**When the PMU supports configuring the Shareability domain for message signaled interrupts:**

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

#### Otherwise:

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

Reserved, RES0.

#### MemAttr, bits [35:32]

##### When the PMU supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
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.

#### Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.



**Otherwise:**

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

Reserved, RES0.

**DATA, bits [31:0]**

Payload for the message signaled interrupt.

**Accessing PMIRQCR12**

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_MSI](#) is implemented and [FEAT\\_CSPMU\\_EXT64](#) is implemented**

Accessible at offset 0xE88 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When ((PMSCR is not implemented && PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure) || (PMSCR is implemented && PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure)) && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)), accesses to this register are RO.
- Otherwise, accesses to this register are RW.

4.4.31 PMIRQSR, Interrupt Status Register

The PMIRQSR characteristics are:

Purpose

Interrupt status register.

Configuration

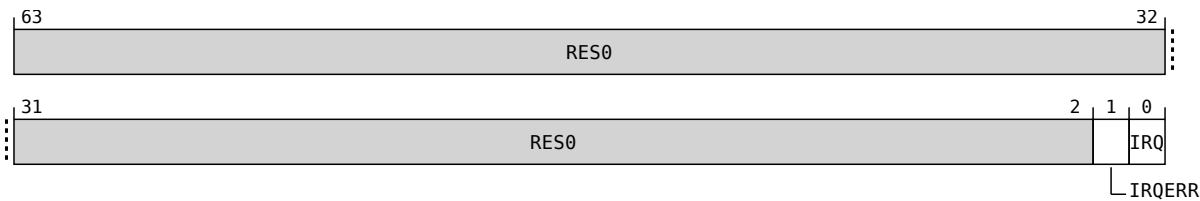
This register is present only when FEAT\_CSPMU\_MSI is implemented. Otherwise, direct accesses to PMIRQSR are RES0.

Attributes

PMIRQSR is a 64-bit register.

This register is part of the CS\_PMU block.

Field descriptions



Bits [63:2]

Reserved, RES0.

IRQERR, bit [1]

Interrupt Error. The possible values of this field are:

IRQERR	Meaning
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.

Access to this field is W1C.

IRQ, bit [0]

PMU Overflow Interrupt write in progress. The defined values of this field are:

IRQ	Meaning
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.

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.

---

Access to this field is RO.

## Accessing PMIRQSR

Accesses to this register use the following encodings:

**When FEAT\_CSPMU\_MSI is implemented and FEAT\_CSPMU\_EXT is implemented**

Accessible at offset 0xEF8 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- When ((PMSCR is not implemented && PMIRQCR2.NSMSI configures the physical address space for a message signaled interrupt as Secure) || (PMSCR is implemented && PMSCR.NSMSI configures the physical address space for a message signaled interrupt as Secure)) && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)), accesses to this register are RO.
- Otherwise, accesses to this register are RW.

### 4.4.32 PMOVS, Overflow Status Snapshot Register

The PMOVS characteristics are:

#### Purpose

PMU monitor overflow status flags.

#### Configuration

This register is present only when [FEAT\\_CSPMU\\_EXT64](#) is implemented. Otherwise, direct accesses to PMOVS are RES0.

#### Attributes

PMOVS is a 64-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

**P<n> , bits [n], for n = 63 to 0**

Overflow status flag for [PMEVCNTR<n>](#).

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> has not overflowed.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> has overflowed.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMOVS.P31 is the cycle counter overflow status flag.

#### Accessing PMOVS

Accesses to this register use the following encodings:

#### When [FEAT\\_CSPMU\\_EXT64](#) is implemented

Accessible at offset 0xC90 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

### 4.4.33 PMOVSCLR<m>, Overflow Flag Status Clear Registers, m = 0 - 7

The PMOVSCLR<m> characteristics are:

#### Purpose

Clear PMU monitor overflow status flags.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMOVSCLR not PMOVSCLR0.

#### Configuration

There are no configuration notes.

#### Attributes

PMOVSCLR<m> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

#### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	⋮
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	⋮
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	⋮
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	⋮

#### P<n> , bits [n], for n = 63 to 0

Overflow status flag for [PMEVCNTR<n>](#) clear. On writes, allows software to clear the overflow status flag for [PMEVCNTR<n>](#) to 0. On reads, returns the overflow status flag for [PMEVCNTR<n>](#) overflow status.

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> has not overflowed.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> has overflowed.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMOVSCLR.P31 allows software to clear the overflow status flag for [PMCCNTR](#) to 0 and query the overflow status flag for [PMCCNTR](#) overflow status.

Access to this field is W1C.

#### Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

#### P<p> , bits [p], for p = 31 to 0

Overflow status flag for [PMEVCNTR<n>](#) clear, where  $n == p + 32m$ . On writes, allows software to clear the overflow status flag for [PMEVCNTR<n>](#) to 0. On reads, returns the overflow status flag for [PMEVCNTR<n>](#) overflow status.

P<p>	Meaning
0b0	PMEVCNTR<n> has not overflowed.
0b1	PMEVCNTR<n> has overflowed.

If FEAT\_CSPMU\_CCNTR is implemented, then PMOVSLR0.P31 allows software to clear the overflow status flag for PMCCNTR to 0 and query the overflow status flag for PMCCNTR overflow status.

Access to this field is WIC.

### Accessing PMOVSLR<m>

Accesses to this register use the following encodings:

#### When FEAT\_CSPMU\_EXT32 is implemented

Accessible at offset  $0xC80 + (4 * m)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When FEAT\_CSPMU\_EXT64 is implemented

Accessible at offset 0xC80 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.34 PMOVSSET<m>, Overflow Flag Status Set Registers, m = 0 - 7

The PMOVSSET<m> characteristics are:

##### Purpose

Set PMU monitor overflow status flags.

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMOVSSET not PMOVSSET0.

##### Configuration

There are no configuration notes.

##### Attributes

PMOVSSET<m> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	...
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32	...
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	...
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	...

##### P<n> , bits [n], for n = 63 to 0

Overflow status flag for [PMEVCNTR<n>](#) set. On writes, allows software to set the overflow status flag for [PMEVCNTR<n>](#) to 1. On reads, returns the overflow status flag for [PMEVCNTR<n>](#) overflow status.

P<n>	Meaning
0b0	<a href="#">PMEVCNTR&lt;n&gt;</a> has not overflowed.
0b1	<a href="#">PMEVCNTR&lt;n&gt;</a> has overflowed.

If [FEAT\\_CSPMU\\_CCNTR](#) is implemented, then PMOVSSET.P31 allows software to set the overflow status flag for [PMCCNTR](#) to 1 and query the overflow status flag for [PMCCNTR](#) overflow status.

Access to this field is W1S.

##### Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

##### P<p> , bits [p], for p = 31 to 0

Overflow status flag for [PMEVCNTR<n>](#) set, where  $n == p + 32m$ . On writes, allows software to set the overflow status flag for [PMEVCNTR<n>](#) to 1. On reads, returns the overflow status flag for [PMEVCNTR<n>](#) overflow status.

P<p>	Meaning
0b0	PMEVCNTR<n> has not overflowed.
0b1	PMEVCNTR<n> has overflowed.

If FEAT\_CSPMU\_CCNTR is implemented, then PMOVSSET0.P31 allows software to set the overflow status flag for PMCCNTR to 1 and query the overflow status flag for PMCCNTR overflow status.

Access to this field is WIS.

### Accessing PMOVSSET<m>

Accesses to this register use the following encodings:

#### When FEAT\_CSPMU\_EXT64 is implemented

Accessible at offset 0xCC0 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### When FEAT\_CSPMU\_EXT32 is implemented

Accessible at offset 0xCC0 + (4 \* m) from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.



#### 4.4.35 PMOVSSR<n>, Overflow Status Snapshot Registers, n = 0 - 7

The PMOVSSR<n> characteristics are:

##### Purpose

Captured copy of [PMOVS](#). Once captured, the value in PMOVSSR is unaffected by writes to [PMOVSSET](#) and [PMOVSCLR](#).

If [FEAT\\_CSPMU\\_EXT64](#) is implemented, this register is not an array and is referred to as PMOVSSR.

##### Configuration

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](#).

This register is present only when [FEAT\\_CSPMU\\_SS](#) is implemented and an implementation implements PMOVSSRn. Otherwise, direct accesses to PMOVSSR<n> are RES0.

##### Attributes

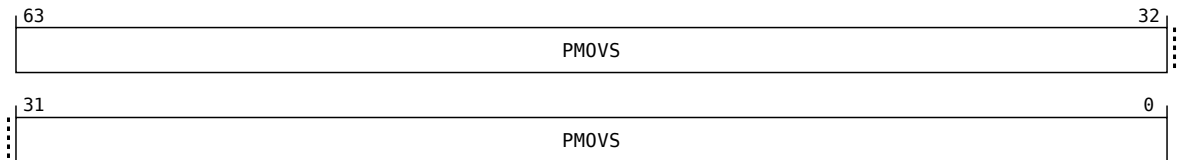
PMOVSSR<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register when [FEAT\\_CSPMU\\_EXT32](#) is implemented.

This register is part of the CS\_PMU block.

##### Field descriptions

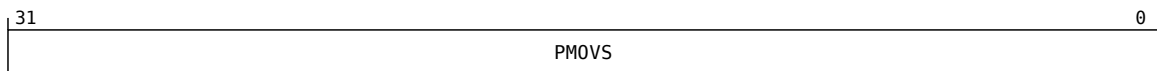
When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### PMOVS, bits [63:0]

The captured copy of [PMOVS](#).

When [FEAT\\_CSPMU\\_EXT32](#) is implemented:



##### PMOVS, bits [31:0]

The captured copy of [PMOVS<n>](#).

##### Accessing PMOVSSR<n>

PMOVSSR<n> is one of the [PMSVR<n>](#) registers. The index <n> of PMOVSSR0 within the [PMSVR<n>](#) registers is an IMPLEMENTATION DEFINED value, IMPDEF\_PMOVSSR\_INDEX.

Accesses to this register use the following encodings:

**When `FEAT_CSPMU_SS` is implemented, `FEAT_CSPMU_EXT32` is implemented, and an implementation implements `PMOVSSRn`**

Accessible at offset  $0 \times 600 + (4 * (\text{IMPDEF\_PMOVSSR\_INDEX} + n))$  from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

**When `FEAT_CSPMU_SS` is implemented, `FEAT_CSPMU_EXT64` is implemented, and an implementation implements `PMOVSSRn`**

Accessible at offset  $0 \times 600 + (8 * \text{IMPDEF\_PMOVSSR\_INDEX})$  from `CS_PMU`

- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc)` || `IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.36 PMPIDR0, Peripheral Identification Register 0

The PMPIDR0 characteristics are:

##### Purpose

Provides discovery information about the component.

##### Configuration

This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR0 are RES0.

##### Attributes

PMPIDR0 is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### Bits [31:8]

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

##### Accessing PMPIDR0

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFE0 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.37 PMPIDR1, Peripheral Identification Register 1

The PMPIDR1 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR1 are RES0.

#### Attributes

PMPIDR1 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions

31	8	7	4	3	0
RES0				DES_0	PART_1

#### Bits [31:8]

Reserved, 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 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 has an IMPLEMENTATION DEFINED value.

---

#### Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

---

Access to this field is RO.

#### 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

## Accessing PMPIDR1

Accesses to this register use the following encodings:

**When `FEAT_PERIPHERAL_ID` is implemented and `FEAT_CSPMU_EXT` is implemented**

Accessible at offset `0xFE4` from `CS_PMU`

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessRealm(addrdesc)`, and `PMROOTCR.RA` IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessNonSecure(addrdesc)`, and `PMROOTCR.RA` != 0b011, accesses to this register are RAZ/WI.
- When (`PMSCR.NSRA` is implemented && (`IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc)`)) && (`PMSCR.NSRA` == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.38 PMPIDR2, Peripheral Identification Register 2

The PMPIDR2 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR2 are RES0.

#### Attributes

PMPIDR2 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions

When the component uses a 12-bit part number:



#### Bits [31:8]

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

#### JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as 0b1

Access to this field is RO.

#### 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 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 has an IMPLEMENTATION DEFINED value.

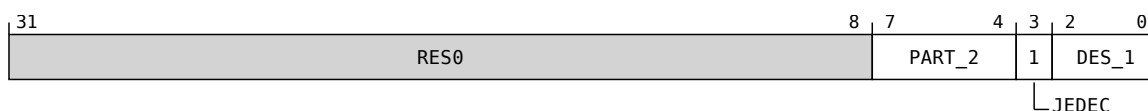
---

#### Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

Access to this field is RO.

**When the component uses a 16-bit part number:**



#### Bits [31:8]

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

#### JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as 0b1

Access to this field is RO.

#### 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 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 has an IMPLEMENTATION DEFINED value.

#### Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

Access to this field is RO.

### Accessing PMPIDR2

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFE8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.



#### 4.4.39 PMPIDR3, Peripheral Identification Register 3

The PMPIDR3 characteristics are:

## Purpose

Provides discovery information about the component.

## Configuration

This register is present only when **FEAT\_PERIPHERAL\_ID** is implemented. Otherwise, direct accesses to PMPIDR3 are RES0.

## Attributes

PMPIDR3 is a 32-bit register.

This register is part of the CS\_PMU block.

## Field descriptions

**When the component uses a 12-bit part number:**

31	8	7	4	3	0
RES0		REVAND		CMOD	

**Bits [31:8]**

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**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.

This field has an IMPLEMENTATION DEFINED value.

For any two components with the same Unique Component Identifier:

- If PMPIDR3.CMOD is zero in both components, then the components are identical.
- If PMPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If PMPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

Access to this field is RO.

**When the component uses a 16-bit part number:**

31	8	7	4	3	0
RES0		REVISION		CMOD	

#### Bits [31:8]

Reserved, 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 has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

#### 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.

This field has an IMPLEMENTATION DEFINED value.

For any two components with the same Unique Component Identifier:

- If PMPIDR3.CMOD is zero in both components, then the components are identical.
- If PMPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If PMPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

Access to this field is RO.

### Accessing PMPIDR3

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFEC from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.40 PMPIDR4, Peripheral Identification Register 4

The PMPIDR4 characteristics are:

##### Purpose

Provides discovery information about the component.

##### Configuration

This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR4 are RES0.

##### Attributes

PMPIDR4 is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions

31	8	7	4	3	0	
RES0					SIZE	DES_2

##### Bits [31:8]

Reserved, 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  $2^{\text{PMPIDR4.SIZE}}$  4KB blocks.

This field has an IMPLEMENTATION DEFINED value.

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.

Access to this field is RO.

##### 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 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 has an IMPLEMENTATION DEFINED value.

---

##### Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4.

---

Access to this field is RO.

### Accessing PMPIDR4

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFD0 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.41 PMPIDR5, Peripheral Identification Register 5

The PMPIDR5 characteristics are:

##### Purpose

Provides discovery information about the component.

##### Configuration

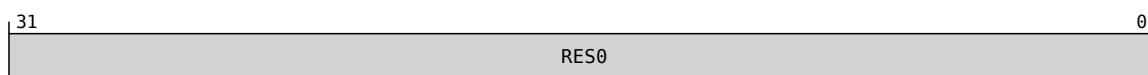
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR5 are RES0.

##### Attributes

PMPIDR5 is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### Bits [31:0]

Reserved, RES0.

##### Accessing PMPIDR5

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFD4 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.42 PMPIDR6, Peripheral Identification Register 6

The PMPIDR6 characteristics are:

##### Purpose

Provides discovery information about the component.

##### Configuration

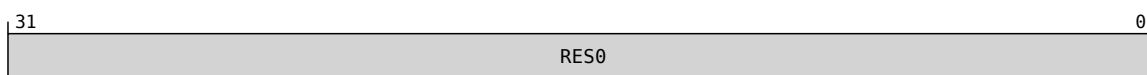
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR6 are RES0.

##### Attributes

PMPIDR6 is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



##### Bits [31:0]

Reserved, RES0.

##### Accessing PMPIDR6

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFD8 from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 4.4.43 PMPIDR7, Peripheral Identification Register 7

The PMPIDR7 characteristics are:

#### Purpose

Provides discovery information about the component.

#### Configuration

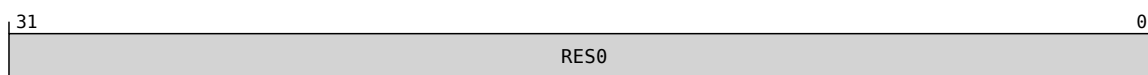
This register is present only when [FEAT\\_PERIPHERAL\\_ID](#) is implemented. Otherwise, direct accesses to PMPIDR7 are RES0.

#### Attributes

PMPIDR7 is a 32-bit register.

This register is part of the CS\_PMU block.

#### Field descriptions



#### Bits [31:0]

Reserved, RES0.

#### Accessing PMPIDR7

Accesses to this register use the following encodings:

**When [FEAT\\_PERIPHERAL\\_ID](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xFDC from CS\_PMU

- When CoreSight management registers ignore access controls, accesses to this register are RO.
- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

4.4.44 PMROOTCR, Root and Realm Control Register

The PMROOTCR characteristics are:

Purpose

Controls observability of Root and Realm events by the performance monitor.

Configuration

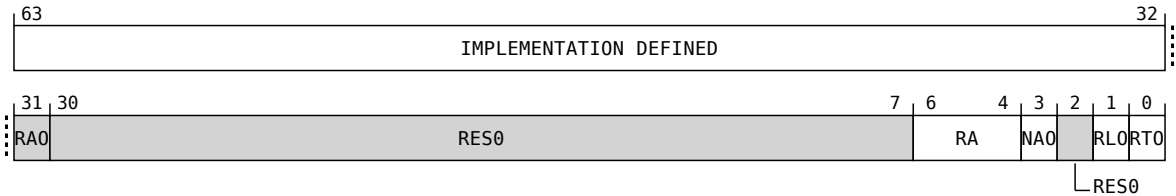
This register is present only when Root and Realm states are implemented, an implementation implements PMROOTCR, and FEAT\_CSPMU\_EXT64 is implemented. Otherwise, direct accesses to PMROOTCR are RES0.

Attributes

PMROOTCR is a 64-bit register.

This register is part of the CS\_PMU block.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events.

Bit [31]

Indicates PMROOTCR is present.

This field reads-as-one.

Reserved, RAO.

Bits [30:7]

Reserved, RES0.

RA, bits [6:4]

When the PMU allows configuration of Secure, Non-secure, and Realm Register Accesses:

Register Access.

RA	Meaning
0b000	Root register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI.
0b001	Root and Realm register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI.
0b010	Root and Secure register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI.
0b011	Root, Secure, Non-secure, and Realm register access is enabled.



Other values are reserved.

For the CoreSight management registers, 0xFA8 to 0xFFC, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by this field.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.

**NAO, bit [3]**

**When the PMU can count or monitor non-attributable events:**

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored.

NAO	Meaning
0b0	Events not attributable with any event source are not counted.
0b1	Counting non-attributable events is not prevented by this field.

When both PMROOTCR and PMSCR are implemented, non-attributable events are counted only if both PMROOTCR.NAO is 1 and PMSCR.{NAO, SO} is nonzero.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.

**Bit [2]**

Reserved, RES0.

**RLO, bit [1]**

Realm Observation. Controls whether events or monitorable characteristics attributable to a Realm event source can be monitored.

RLO	Meaning
0b0	Events attributable to a Realm event source are not counted.
0b1	Events attributable to a Realm event source are counted.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

**RTO, bit [0]**

Root Observation. Controls whether events or monitorable characteristics attributable to a Root event source can be monitored.

RTO	Meaning
0b0	Events attributable to a Root event source are not counted.
0b1	Events attributable to a Root event source are counted.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

### Accessing PMROOTCR

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_ACR](#) is implemented and [FEAT\\_CSPMU\\_EXT](#) is implemented**

Accessible at offset 0xE48 from CS\_PMU

- When !IsAccessRoot(addrdesc), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

4.4.45 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.

Configuration

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.

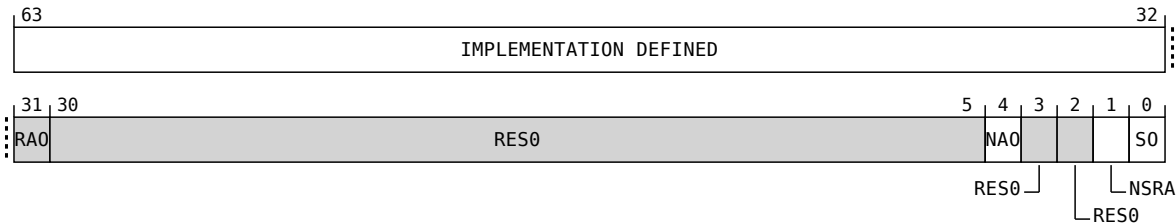
This register is present only when Secure state is implemented and an implementation implements PMSCR. Otherwise, direct accesses to PMSCR are RES0.

Attributes

PMSCR is a 64-bit register.

This register is part of the CS\_PMU block.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events.

Bit [31]

Indicates PMSCR is present.

This field reads-as-one.

Reserved, RAO.

Bits [30:5]

Reserved, RES0.

NAO, bit [4]

When the PMU can count or monitor non-attributable events:

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored.

NAO	Meaning
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 field.

When both **PMROOTCR** and **PMSCR** are implemented, non-attributable events are counted only if both **PMROOTCR.NAO** is 1 and **PMSCR.{NAO, SO}** is nonzero.

This field is optional if Root and Realm states are not implemented. When this field is not implemented, the PMU behaves as if **PMSCR.NAO** is 0, and whether events or monitorable characteristics not attributable with any source can be monitored is controlled by **PMSCR.SO**.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.

**Bit [3]**

**When **FEAT\_CSPMU\_MSI** is not implemented or **CS\_PMU.PMSR.NSMSI** configures MSIs as Non-secure MSIs:**

Reserved, RES0.

**When the PMU supports MPAM on message signaled interrupts:**

MPAM Non-secure signaling for Secure MSIs. Defines the **MPAM\_NS** setting used for Secure message signaled interrupts.

MSI_MPAM_NS	Meaning
0b0	Secure MSIs are issued with <b>MPAM_NS</b> set to 0.
0b1	Secure MSIs are issued with <b>MPAM_NS</b> set to 1.

**Otherwise:**

Reserved, RES0.

**Bit [2]**

**When **FEAT\_CSPMU\_MSI** is not implemented:**

Reserved, RES0.

**When **IsFeatureImplemented(FEAT\_CSPMU\_MSI) && ((PMSCR.NSRA is implemented && (CS\_PMU.PMSR.NSRA == '1')) || (PMROOTCR.RA is implemented && (CS\_PMU.PMROOTCR.RA IN {'0x1'})))**:**

The physical address space used for message signaled interrupts is Non-secure.

Reserved, RES0.

**Otherwise:**

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

If the PMU allows configuration of Non-secure and Realm Register Accesses, then this field resets to the same reset value as PMSCR.NSRA.

Otherwise, the reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

#### NSRA, bit [1]

**When PMROOTCR.RA is not implemented and the PMU allows configuration of Non-secure and Realm Register Accesses:**

Non-secure and Realm register access.

NSRA	Meaning
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.

For the CoreSight management registers, 0xFA8 to 0xFFC, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by PMSCR.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.

#### SO, bit [0]

Secure Observation. Controls whether events or monitorable characteristics attributable to a Secure event source can be monitored.

SO	Meaning
0b0	Events attributable to a Secure event source are not counted.
0b1	Events attributable to a Secure event source are counted.

This field also controls whether events or monitorable characteristics not attributable with any source can be monitored. See PMSCR.NAO.

The reset value of this field depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

### Accessing PMSCR

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.

Accesses to this register use the following encodings:

**When `FEAT_CSPMU_ACR` is implemented and `FEAT_CSPMU_EXT` is implemented**

Accessible at offset `0xE40` from `CS_PMU`

- When `!IsAccessRoot(addrdesc)` and `!IsAccessSecure(addrdesc)`, accesses to this register are RAZ/WI.
- When `PMROOTCR.RA` is implemented, `IsAccessSecure(addrdesc)`, and `PMROOTCR.RA` IN `{0b001, 0b000}`, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.46 PMSSCR, Snapshot Control Register

The PMSSCR characteristics are:

##### Purpose

Provides a mechanism for software to initiate a sample.

If [PMSSSR](#) is not implemented, PMSSCR additionally holds status information about the captured monitors.

##### Configuration

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.

In some Arm PE implementations where the snapshot feature is an IMPLEMENTATION DEFINED extension, PMSSCR is located at offset 0x6F0.

This register is present only when [FEAT\\_CSPMU\\_SS](#) is implemented. Otherwise, direct accesses to PMSSCR are RES0.

##### Attributes

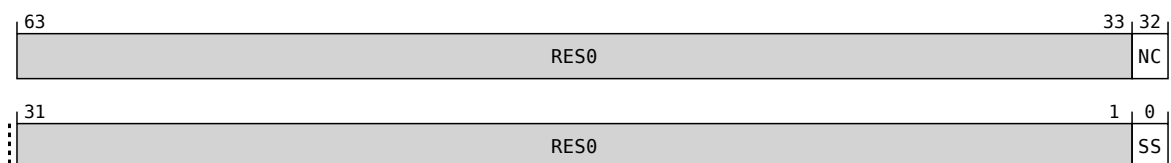
PMSSCR is a:

- 64-bit register when CS\_PMU.PMSSSR is not implemented.
- 32-bit register when CS\_PMU.PMSSSR is implemented.

This register is part of the CS\_PMU block.

##### Field descriptions

When CS\_PMU.PMSSSR is not implemented:



##### Bits [63:33]

Reserved, RES0.

##### NC, bit [32]

No capture. Indicates whether the PMU monitors have been captured.

NC	Meaning
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 1 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.

##### Bits [31:1]

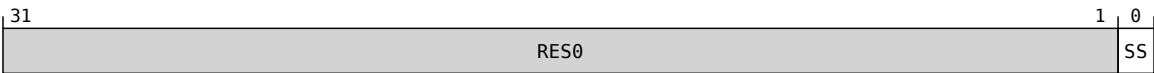
Reserved, RES0.

**SS, bit [0]**

Capture now.

SS	Meaning
0b0	Ignored.
0b1	Initiate a capture immediately.

**When CS\_PMU.PMSSSR is implemented:**



**Bits [31:1]**

Reserved, RES0.

**SS, bit [0]**

Capture now.

SS	Meaning
0b0	Ignored.
0b1	Initiate a capture immediately.

**Accessing PMSSCR**

Accesses to this register use the following encodings:

**When FEAT\_CSPMU\_SS is implemented and FEAT\_CSPMU\_EXT is implemented**

Accessible at offset 0xE30 from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.



#### 4.4.47 PMSSRR, Snapshot Reset Register

The PMSSRR characteristics are:

##### Purpose

Configure Snapshot to reset monitors after each sample is taken.

##### Configuration

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  $0 \times 6F4$ .

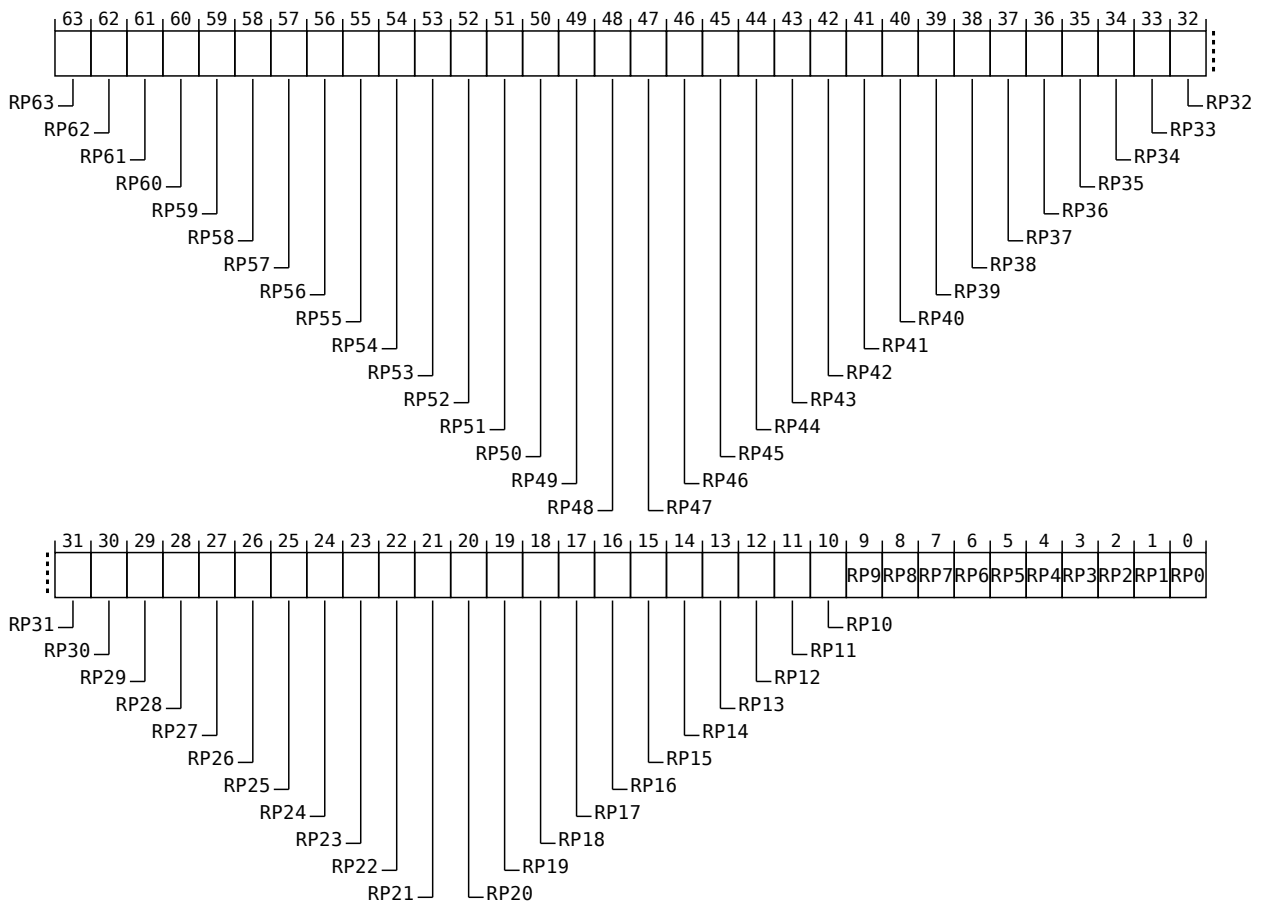
This register is present only when [FEAT\\_CSPMU\\_SS](#) is implemented and an implementation implements PMSSRR. Otherwise, direct accesses to PMSSRR are RES0.

##### Attributes

PMSSRR is a 64-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions



RP<m> , bits [m], for m = 63 to 0

Reset monitor. If  $m \geq \text{PMCFGR.N}$ , the number of implemented monitors, then  $\text{RP}[m]$  is RAZ/WI. Otherwise, indicates whether  $\text{PMEVCNTR}\langle m \rangle$  and  $\text{PMOVS}[m]$  are to be reset after a capture.

$\text{RP}\langle m \rangle$	Meaning
0b0	Do not reset $\text{PMEVCNTR}\langle m \rangle$ and $\text{PMOVS}[m]$ on capture.
0b1	Reset $\text{PMEVCNTR}\langle m \rangle$ and $\text{PMOVS}[m]$ to zero on capture.

If  $\text{FEAT\_CSPMU\_CCNTR}$  is implemented, then  $\text{PMSSRR.RP31}$  controls reset of the cycle counter.

### Accessing PMSSRR

Accesses to this register use the following encodings:

**When  $\text{FEAT\_CSPMU\_SS}$  is implemented and  $\text{FEAT\_CSPMU\_EXT}$  is implemented**

Accessible at offset  $0xE38$  from  $\text{CS\_PMU}$

- When  $\text{PMROOTCR.RA}$  is implemented,  $\text{IsAccessSecure}(\text{addrdesc})$ , and  $\text{PMROOTCR.RA} \in \{0b001, 0b000\}$ , accesses to this register are RAZ/WI.
- When  $\text{PMROOTCR.RA}$  is implemented,  $\text{IsAccessRealm}(\text{addrdesc})$ , and  $\text{PMROOTCR.RA} \in \{0b010, 0b000\}$ , accesses to this register are RAZ/WI.
- When  $\text{PMROOTCR.RA}$  is implemented,  $\text{IsAccessNonSecure}(\text{addrdesc})$ , and  $\text{PMROOTCR.RA} \neq 0b011$ , accesses to this register are RAZ/WI.
- When  $(\text{PMSCR.NSRA}$  is implemented  $\&\& (\text{IsAccessNonSecure}(\text{addrdesc}) \parallel \text{IsAccessRealm}(\text{addrdesc})) \&\& (\text{PMSCR.NSRA} == '0')$ , accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

#### 4.4.48 PMSSSR, Snapshot Status Register

The PMSSSR characteristics are:

##### Purpose

Holds status information about the captured monitors.

##### Configuration

PMSSSR is an optional one of the [PMSVR<n>](#) registers.

This register is present only when [FEAT\\_CSPMU\\_SS](#) is implemented, an implementation implements PMSSSR, and [FEAT\\_CSPMU\\_EXT32](#) is implemented. Otherwise, direct accesses to PMSSSR are RES0.

##### Attributes

PMSSSR is a 32-bit register.

This register is part of the CS\_PMU block.

##### Field descriptions

31		1	0
RES0			NC

##### Bits [31:1]

Reserved, RES0.

##### NC, bit [0]

No capture. Indicates whether the PMU monitors have been captured.

NC	Meaning
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 1 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.

##### Accessing PMSSSR

PMSSSR is an one of the [PMSVR<n>](#) registers. The index <n> of PMSSSR within the [PMSVR<n>](#) registers is an IMPLEMENTATION DEFINED value, IMPDEF\_PMSSSR\_INDEX. Arm recommends that PMSSSR is read after the other [PMSVR<n>](#) registers, meaning it might be placed last in the [PMSVR<n>](#) registers.

Accesses to this register use the following encodings:

**When [FEAT\\_CSPMU\\_SS](#) is implemented, [FEAT\\_CSPMU\\_EXT32](#) is implemented, and an implementation implements PMSSSR**

Accessible at offset  $0 \times 600 + (4 * \text{IMPDEF\_PMSSSR\_INDEX})$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.

- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

#### 4.4.49 PMSVR<n>, Saved Value Registers, n = 0 - 127

The PMSVR<n> 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.

##### Configuration

This register is present only when [FEAT\\_CSPMU\\_SS](#) is implemented. Otherwise, direct accesses to PMSVR<n> are RES0.

##### Attributes

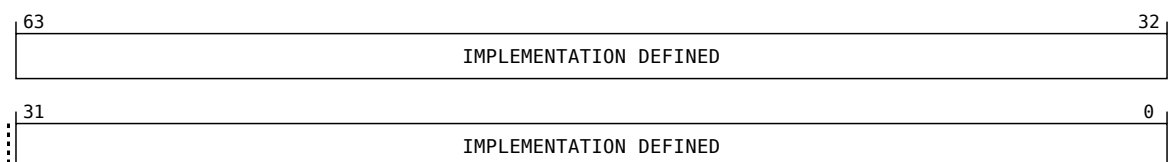
PMSVR<n> is a:

- 64-bit register when [FEAT\\_CSPMU\\_EXT64](#) is implemented.
- 32-bit register otherwise.

This register is part of the CS\_PMU block.

##### Field descriptions

When [FEAT\\_CSPMU\\_EXT64](#) is implemented:



##### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

##### Otherwise:



##### IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

##### Accessing PMSVR<n>

Accesses to this register use the following encodings:

When [FEAT\\_CSPMU\\_EXT32](#) is implemented

Accessible at offset  $0 \times 600 + (4 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

**When FEAT\_CSPMU\_EXT64 is implemented**

Accessible at offset  $0 \times 600 + (8 * n)$  from CS\_PMU

- When PMROOTCR.RA is implemented, IsAccessSecure(addrdesc), and PMROOTCR.RA IN {0b001, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessRealm(addrdesc), and PMROOTCR.RA IN {0b010, 0b000}, accesses to this register are RAZ/WI.
- When PMROOTCR.RA is implemented, IsAccessNonSecure(addrdesc), and PMROOTCR.RA != 0b011, accesses to this register are RAZ/WI.
- When (PMSCR.NSRA is implemented && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && (PMSCR.NSRA == '0'), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

# 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](#) ( $\lambda$ ) and Average occupancy ( $L$ ) with average [Response time](#) ( $W$ ).

## Memory System Resource Partitioning and Monitoring

See *Arm® Architecture Reference Manual System Component Specification; 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.