Arm® CoreSight™ Architecture

Performance Monitoring Unit Architecture

ARM IHI 0091 A.a-00bet0

CoreSight Performance Monitoring Unit Architecture

Release information

Date	Version	Changes
2020/Nov/04	00bet0	• First non-confidential release.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Contents

CoreSight Performance Monitoring Unit Architecture

Glossary

Preface

Document status

Beta release.

The information contained in this manual is at Beta quality. Beta quality means that all major features of the specification are described in the manual, some details might be missing.

In case of any apparent discrepancy or missing information, please contact Arm Limited.

About this book

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

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

Using this book

IWZYHY This manual has three main sections:

Description
: An *informative* section describing the architecture and motivation.

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

Programmers' Model
: A *normative* section that provides the definitions of the registers added by this manual.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

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

`monospace`

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

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

SMALL CAPITALS

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

Blue text

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

Numbers

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

Rules-based writing

This specification consists of a set of individual *content items*. Content items are classified into the following types:

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

Rules are normative statements. An implementation which is compliant with this specification must conform to all of the Rules in this specification.

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

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

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

Content item classes

Rule

A Rule is a statement which either

- describes the behaviour of a compliant implementation, or
- defines concepts or terminology.

A Rule is identified by the letter R.

Information

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

An Information statement is identified by the letter I.

Rationale

A Rationale statement explains why the specification was specified as it was.

A Rationale statement is identified by the letter X.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is identified by the letter 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 identified by the letter S.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001*, *0002*, . . .).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

R	This is a Rule.
R_{X001}	This is a Rule with an identifier.
X	This is a Rationale statement.
I	This is an Information statement.
U	This is an Implementation note.
S	This is a Software usage statement.

Additional reading

This section lists publications by Arm and by third parties.

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

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

[2] *Arm® Architecture Reference Manual Supplement; Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A*. (ARM DDI 0598) Arm Limited.

[3] *Arm® Architecture Reference Manual for ARMv8-A architecture profile*. (ARM DDI 0487) Arm Limited.

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

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

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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

- The title (CoreSight Performance Monitoring Unit Architecture).
- The number (ARM IHI 0091 A.a-00bet0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Description

This section is *informative*.

1.1 About Performance Monitors

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

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

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

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

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

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

A PMU has two main use models:

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

Both approaches typically use *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 used by profiling.

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

For some markets, performance merely has to be *good enough*. For example, mobile applications are dominated by frameworks and middleware. Optimizing the application software is a comparatively niche activity.

However:

- Profiling can be used to provide data for *Profile-guided Optimization* (PGO).
- Frameworks and middleware are themselves optimized. This improves all applications and so benefits the platform.
- For server and *High-performance Computing* (HPC) systems, software optimization is a mainstream activity.
  - These systems are often sold on their performance at key benchmarks.
  - For example, SPEC, TPC, Linpack, and so on.

As a result, these systems might be subject to continuous improvement.

Not all software is like this. There is much *dusty deck* code that is deemed *good enough* or too hard to optimize. Profiling can also be used as a mechanism to expose this code.

1.1.2 Monitoring

Monitoring is a tool 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 [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 use Performance Monitoring hardware to automatically make these changes.

1.1.3 Sampling

There are two main sampling models for HPM:

- Time-based sampling. *Time-based sampling with Arm DS-5 Streamline Performance Analyzer* and *Source-level profiling with Arm MAP source-level profiler* show example output of a time-based sampling tools.
- Event-based sampling. *Hot-spot analysis from event-based sampling with Arm DS-5 Streamline Performance Analyzer* shows example output of a event-based sampling tool.

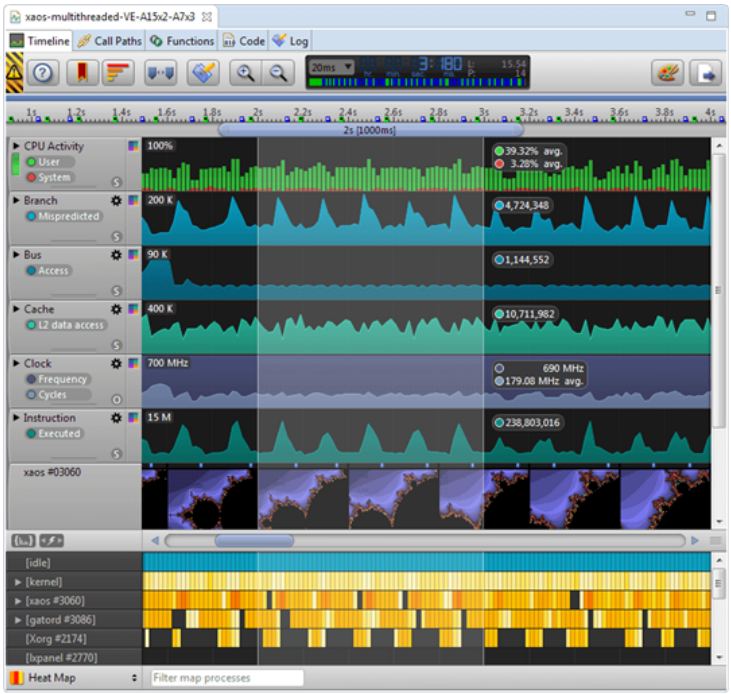


Figure 1.1: Time-based sampling with Arm DS-5 Streamline Performance Analyzer

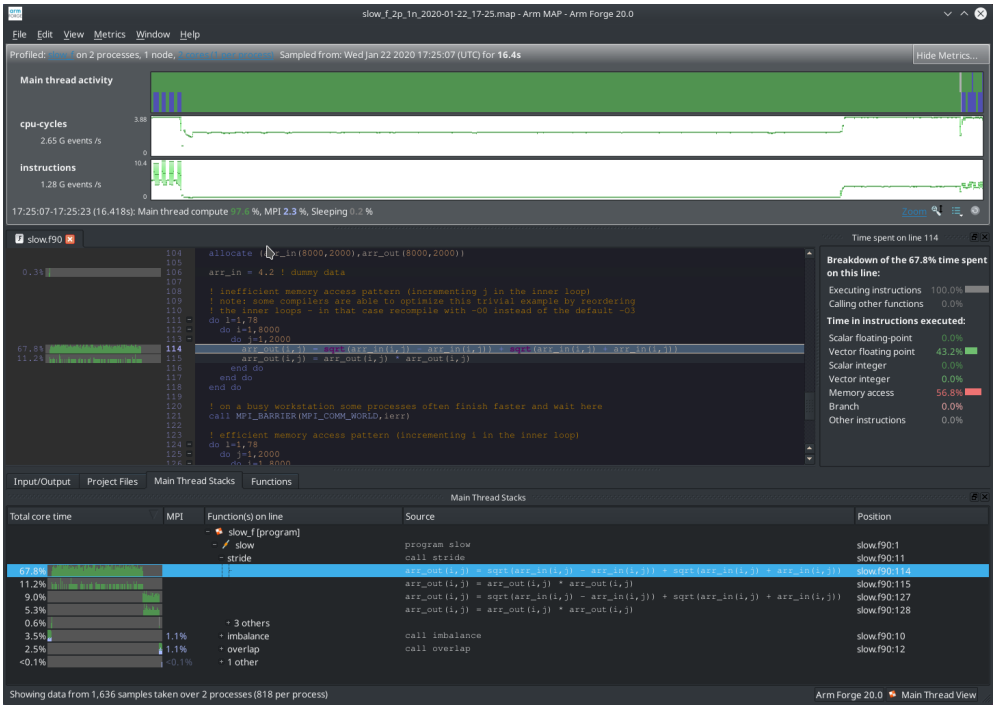
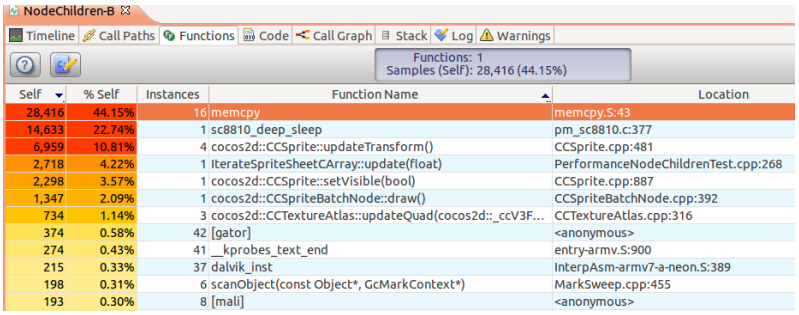


Figure 1.2: Source-level profiling with Arm MAP source-level profiler



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

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

1.2 Rationale for a standard PMU architecture

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

For example, a standard architecture for a PMU monitors is allows a shared *PMU driver* software layer. Such a driver abstracts the software interface to a PMU, and can be configured with target-specific data.

For instance, in a typical Arm processor PMU there are an IMPLEMENTATION DEFINED number of event counter monitors, most of which can be configured to count one from an IMPLEMENTATION DEFINED set of events. The Arm processor PMU also contains an example of a *fixed function* counter that, in this case, can only count CPU clock cycles. This target-specific knowledge does not need to be known at the PMU driver level.

Without a common architecture, the tool requires concrete implementations at many levels. This architecture provides such an abstraction. The Armv8 PMUv3 extension is a further abstraction layered on top of this abstraction.

1.3 What to measure

The following are suggestions for *what to measure*:

Utilization

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

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

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

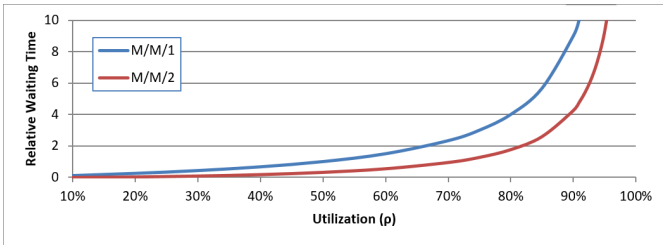


Figure 1.4: Queuing time as a function of utilization for M/M/1 and M/M/2 queues

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

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

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

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

Latency

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

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

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

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

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

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

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

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

Memory System Resource Partitioning and Monitoring (MPAM) [2] defines a standard bandwidth monitor. See *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* can be calculated by counting the total number of *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*.

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

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

Arrivals
Arrival rate (λ) can be calculated by counting the total number of *arrivals* and dividing by the elapsed time. Arrival rate is used in calculating Utilization (ρ) and other measurements. ($\rho = \lambda \times S$.)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note

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

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

1.4 MPAM

MPAM [2] 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.

Overview is taken from the introduction to [2].

MPAM v1.1 describes the MPAM v1.1 extensions.

1.4.1 Overview

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

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

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

These approaches are:

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

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

The MPAM extension describes:

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

1.4.2 MPAM v1.1

The following MPAM features are also described in [2]:

- A *Resource Instance Selector* (RIS) field is added to MPAMCFG_PART_SEL and MSMON_CFG_MON_SEL registers. Resource instance selection provides a means to access control settings in an MSC for multiple resources of the same type.
- Greater range for the memory bandwidth usage monitors through optional long registers.

Other minor additions to the MPAM programmers' model are also included in MPAM v1.1.

Chapter 2
Specification

This section is *normative*.

2.1 Organization

RDRPFZ Each monitor <n> comprises:

- A monitor value register, PMEVCNTR<n>.
- An optional event select register, PMEVTYPER<n>.
- An optional event filter register, PMEVFILTR<n>.
- A monitor enable bit, PMCNTEN[n].

If the monitor is an event counter, or defines an overflow condition, the monitor also comprises:

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

IZJTZQ The names of registers in an implementation might differ from those in this document. In particular:

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

RKRLSW The *Performance Monitoring Unit* (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 [3].

RKBRSS When event counters are implemented, or any monitor defines an overflow condition, the PMU includes an overflow interrupt request signal.

RMRBCQ Monitors might be split into *monitor groups*. See *Monitor group extension*.

RVPSKV The size of each monitor value is IMPLEMENTATION DEFINED, up-to 64 bits.

RJRSCZ The size of the largest monitor value is discoverable through PMCFGR.SIZE.

UNNHQG When event counters are implemented, Arm recommends that all event counters are the same size.

Exceptions can be made for fixed-function counters, such as cycle counters, as is the case in the Armv8-A PMU [3] when FEAT_PMUv3p5 is not implemented, and in the Armv8-M PMU [4].

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

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

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

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

  For example, if an overflow generates a CPU 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.

RBNQPR If the largest monitor value register is 32 bits or smaller, all monitor value registers are at word-aligned addresses. Otherwise, all monitor value registers are at doubleword-aligned addresses.

R_{HKCFV} The number of monitors is IMPLEMENTATION DEFINED and is discoverable through PMCFGR.N:

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

I_{XDDYT} If *Monitor group extension* or *Snapshot extension* are implemented, the maximum number of counters that can be implemented might be less than this. See the descriptions of the extensions for more information.

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

- A write of 1 to PM{*fn*}SET<m>[q] sets the PM{*fn*}[$q + (32 \times m)$] bit to 1.
- A write of 1 to PM{*fn*}CLR<m>[q] clears the PM{*fn*}<m>[$q + (32 \times m)$] bit to 0.
- A write of 0 to PM{*fn*}SET<m>[q] or PM{*fn*}CLR<m>[$q + (32 \times m)$] has no effect.
- A read of PM{*fn*}SET<m>[q] or PM{*fn*}CLR<m>[q] returns the current value of PM{*fn*}[$q + (32 \times m)$].

Note

Due to technical limitations of this manual, the set/clear registers are described in *Programmers' Model* as:

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

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

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 [5].

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

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

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

See also:

- *Accessing PMU registers*.
- *Extensions*.
- *Programmers' Model*.

2.2 Operation of the PMU

RWXSQT The PMU has a *RUN* state and a *STOP* state. *PMU state-machine* shows this.

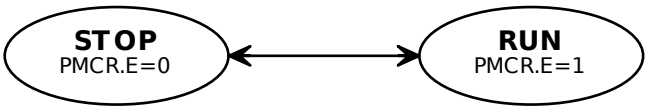


Figure 2.1: PMU state-machine

RHLTDF The PMU operating state is determined by the PMCR.E control bit.

PMCR.E	State
0b0	STOP
0b1	RUN

IKNMNQ *Freeze on overflow extension* describes a third state, WAIT.

SFJDNY Software configures the monitors using PMEVTYPER<n> and PMEVFILTR<n>. If these registers are programmable, software must program PMEVTYPER<n> and PMEVFILTR<n> for each monitor before enabling that monitor.

See also:

- *Accuracy of the PMU*.
- *Security*.
- *Extensions*.

2.2.1 Event counting

This section applies for monitors that are event counters.

RLXYWS 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 *Security*.
- The event conditions specified by PMEVTYPER<n> and, if implemented, PMEVFILTR<n> occur.

RTGVMJ Increments of event counters by the PMU are atomic operations on the event counter.

RXKQWT Event counters are unsigned integer counters.

RQLMFG The event conditions specified by PMEVTYPER<n> and PMEVFILTR<n> 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.

RVPCMF For a given event counter, *n*, if the event counter increment generates unsigned overflow of the event counter:

- The overflow status flag PMOVS[n] is set to 1.

- The event counter wraps through zero.

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

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

2.2.2 State or event monitoring

This section applies for monitors that are not event counters.

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

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

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

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

R_{HSHSJ} The monitoring conditions specified by PMEVTYPER<n> and PMEVFILTR<n> configure what the monitor monitors, including:

- The state or events monitored by the monitor.
- Any filtering of the state or events, for example, on the context of the component being monitored.

R_{VHWBS} For a given monitor, *n*, the monitor might define an overflow condition. When the monitor enters the overflow condition, the overflow status flag PMOVS[n] is set to 1.

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

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

2.2.3 Mapping controls and fixed-function monitors

R_{YSQVN} The mapping of the PMEVTYPER<n> and PMEVFILTR<n> register contents to controls is IMPLEMENTATION DEFINED, and might differ between monitors.

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

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

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

2.2.4 Interrupt signaling

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

- The overflow status flag PMOVS[n] is set to 1.
- The interrupt enable bit PMINTEN[n] is set to 1.
- The PMU is in the RUN or WAIT state.

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

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

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

R_{FDLKQ} A PMU might include the PMIRQCR0, PMIRQCR1, and PMIRQCR2 registers to configure a message-signaled interrupt. The status of the message is indicated in PMIRQSR. If this standard form of message-signaled interrupts are implemented then PMCFGR.MSI reads as 1.

I_{VBWMC} Arm strongly recommends that if a PMU has close affinity to a single PE, the interrupt request signal is configured as a *Private Peripheral Interrupt* (PPI) for that PE.

2.3 Accuracy of the PMU

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

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

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

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

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

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

2.4 Accessing PMU registers

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

If monitors are writable:

R_{CGWBM}
- It is IMPLEMENTATION DEFINED whether the monitor value and monitor configuration registers can be written to other than in the STOP state.

R_{CLXGJ}
- If the PMU does not allow monitor value and monitor configuration registers to be written to other than in the STOP state then writes to the registers other than in the STOP state are ignored.

R_{YNJLW}
- If the PMU does not allow monitor value and monitor configuration registers to be written to other than in the STOP state then PMCFGR.NA reads as 1.

R_{YHTDZ}
- If the PMU allows monitor value and monitor configuration registers to be written to other than in the STOP state, then writes to the registers other than in the STOP state take effect in finite time. *Take effect* means:
  - Before the write takes effect, reads of the register by the PMU made as part of the operation of the PMU yield the old value.
  - After the write takes effect, reads of the register by the PMU made as part of the operation of the PMU yield the new value.

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

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

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

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

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

I_{DCPZN} For example, the Armv8-A PMU [3] defines that external accesses to the PMU are not permitted when the OS Lock is locked.

R_{ZWQLG} A memory-mapped PMU must implement the supported access sizes defined by the section *Supported access sizes* in [3].

R_{ZQQKP} 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 must be supported.

R_{ZNTBP} It is IMPLEMENTATION DEFINED and recommended that doubleword-aligned 64-bit accesses are supported to the following pairs of 32-bit registers (*n* is even):

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

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

R_{DDZMD} It is IMPLEMENTATION DEFINED and recommended that permitted doubleword-aligned 64-bit accesses to 64-bit registers and supported pairs of 32-bit registers are single-copy atomic at doubleword granularity.

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

R_{ZWZGC} A memory-mapped PMU must implement the memory-mapped component synchronization rules defined by the section *Synchronization of memory-mapped registers* in [3].

R_{KRVRQ} A memory-mapped PMU must implement the access requirements for reserved and unallocated registers defined by the section *Access requirements for reserved and unallocated registers* in [3].

See also:

- *Security*.
- *Programmers' Model*.

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

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

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

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

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

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

I_{GRNVM} Example implementations of authentication controls are:

- 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.
- 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 can be programmed to allow the Non-secure PMU to monitor Secure operations.
- A fixed configuration. For example, always treating Secure operations as prohibited operations.
- A combination of these.

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

I_{GMSMS} An example of a PMU that is accessible to Non-secure software is a memory-mapped PMU that is mapped in the Non-secure physical address space.

R_{RYNLT} When a component incorporating a PMU implements separate Secure and Non-secure operating states, any IMPLEMENTATION DEFINED authentication controls are not configurable by Non-secure software.

I_{WTVDK} The CoreSight architecture [5] describes a signal authentication interface comprising four signals: **NIDEN**, **DBGEN**, **SPIDEN**, and **SPNIDEN**. These signals control:

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

I_{DNMWG} In the Armv8-A architecture profile Performance Monitors Extension [3]:

- 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 prohibits counting of events attributable to 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 and/or EL2.
- 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.

I_{TZLYG} 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 PMU [3].

2.6 Extensions

2.6.1 Freeze on overflow extension

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

R_{GYWBJ} If PMCR.FZO is implemented then PMCFGR.FZO reads as 1.

I_{FSDCY} If PMCR.FZO is implemented then the PMU has RUN, STOP, and *WAIT* states. *Freeze-on-overflow state-machine* shows this.

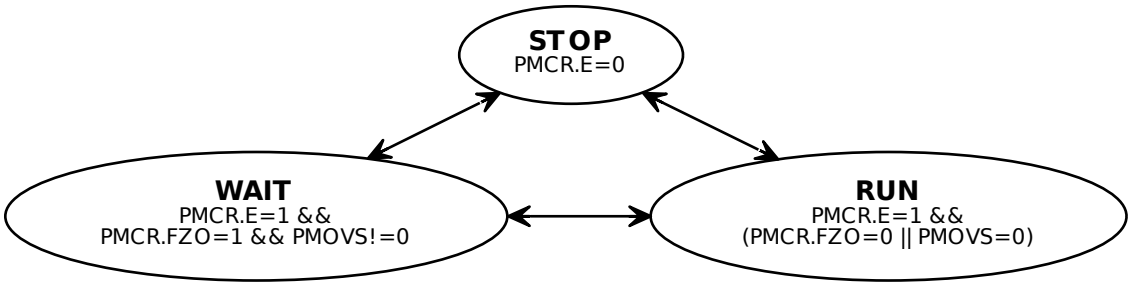


Figure 2.2: Freeze-on-overflow state-machine

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

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

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

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

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

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

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

- The CHAIN event is generated for the odd-numbered counter.

- The PMU enters the WAIT state.

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

R_{DPGBH} If a PMU implements the *Freeze on overflow extension* and *Fixed-function cycle counter extension*, it is IMPLEMENTATION DEFINED whether or not the operation of the cycle counter ignores the freeze-on-overflow feature. That is, whether or not, if enabled and allowed, the cycle counter will count cycles when the PMU is in the WAIT state.

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

2.6.2 Halt-on-debug extension

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

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

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

This might differ for groups of events.

R_{XPPSF} If the PMU implements PMCR.HDBG then PMCFGR.HDBG reads as 1.

2.6.3 Fixed-function cycle counter extension

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

An implementation can include other fixed-function monitors. If the cycle counter extension is not implemented, a PMU might still include a fixed-function cycle counter.

The only difference is that it does not implement it as monitor 31.

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

R_{KFHHP} If the Cycle Counter extension is implemented then PMCFGR.CC reads as 1.

R_{HJZYK} For compatibility with the Arm architecture PMU, the cycle counter is always an alias for PMEVCNTR<31>.

R_{CMLXS} If the Cycle Counter extension is implemented and the cycle counter is 32 bits or fewer, the implementation might include a prescaler for the cycle counter, such that it counts by one for every 64 cycles. If the prescaler is implemented then PMCFGR.CCD reads as 1.

I_{JNMYM} If the Cycle Counter extension is not implemented then monitor 31 (if implemented) is a regular monitor.

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

- Counts when PMCR.DP is 1 or the agent being monitored is in the first state.

- Does not count when PMCR.DP is 0 and the agent being monitored is in the second state.

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

I_{BNDHS} In the Armv8-A architecture profile [3], the first state is a state where counting events is allowed and the second state is a state where counting events is prohibited.

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

See *Security*.

I_{SVGVM} In the Armv8-M architecture profile [4], when the PMU and 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.

2.6.4 Monitor group extension

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

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

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

R_{TFNPJ} Each monitor group can have up-to 32 monitors. The maximum number of monitors per group depends on both the number of groups and the size of the largest implemented monitor. The following table shows this.

Number of monitor groups	Max. per group, if monitors are 32-bits or smaller	Max. per group, if at least one monitor is larger-than 32 bits
4 or fewer	32	32
Between 5 and 8	32	16
9 or more	16	8

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

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

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

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

2.6.5 Counter chaining

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

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

S_{PGJDB} For example where two 16-bit counters, PMEVCNTR<0> and PMEVCNTR<1> are provided, they can be chained to form a 32-bit counter by programming PMEVTYPER<1> to count the CHAIN event.

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

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

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

2.6.6 Snapshot extension

R_{WGXRN} It is IMPLEMENTATION DEFINED whether the PMU includes the Snapshot mechanism.

R_{PDXMS} If the Snapshot mechanism is implemented then PMCFGR.SS reads as 1.

R_{DNJSY} On a Capture event, the PMU writes snapshot values to the snapshot value registers, PMSVR<n>, and sets PMSSSR.NC to zero to indicate a successful capture.

R_{HSCPL} The snapshot values must include PMSSSR.

I_{NNVZK} The snapshot values typically include:

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

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

I_{SSNNY} Because of the limited number of snapshot value registers, and requirement to capture PMSSSR, the Snapshot mechanism will not capture all the implemented counters if either:

- 64 or more event counters are implemented, when all counters are 32-bits or smaller.
- 32 or more event counters are implemented, when at least one counter is larger-than 32-bits.

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

I_{TRJQW} PMSSSR and PMOVSSR<m> are aliases for PMSVR<n>, where n is IMPLEMENTATION DEFINED for each register.

R_{CVTPF} It is IMPLEMENTATION DEFINED whether the Snapshot mechanism also supports PMSSRR. If PMSSRR is implemented then each of the monitors so-configured by PMSSRR are reset to 0 after a Capture event.

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

- Assertion of an IMPLEMENTATION DEFINED snapshot request signal, **PMUSNAPSHOTREQ**.
- Writing 1 to PMSSCR.SS.

2.6.7 Trace generation extension

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

R_{SRKVT} If the PMU implements PMCR.TRO then PMCFGR.TRO reads as 1.

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

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

2.6.8 Export extension

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

R_{JSMMJ} If the PMU implements PMCR.X then PMCFGR.EX reads as 1.

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

2.6.9 Dual-page extension

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

R_{XZWHB} The following are Page 1 registers:

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

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

- PMCCNTR.

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

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

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

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

- PMCIDR3.

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

- PMCFGR.
- PMDEVARCH.
- PMDEVTYPE.

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

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

R_{RSCXF} In a dual-page implementation:

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

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

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

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

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

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

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

I_{CQFCY} The Armv8 PMU does not implement the dual-page view on its CoreSight interface. The PE implements a second view of the PMU through System registers.

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

Chapter 3
Programmers' Model

This section is *normative*.

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

Future development of this architecture might define an alternative structure and layout that is better suited for systems that only support 64-bit operation. For instance, by locating every register at a doubleword-aligned address. Such changes might not be backwards compatible.

See also:

- *Accessing PMU registers*.

3.1 Memory-mapped registers

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

32b Monitors up-to 32 bits in size.
64b Monitors up-to 64 bits in size.
CC *Fixed-function cycle counter extension*.
CS CoreSight registers.
MG
Monitor group extension.
MSI
Message-signaled interrupts.
SS *Snapshot extension*.

3.1.1 When the dual-page extension is not implemented

Table 3.1: Memory-mapped register map

Offset	Access	Version	Register	Description
0x000+4×*n*	R/W	32b	PMEVCNTR<n>	Event Count Register <*n*> (up-to 32 bits)
0x000+8×*n*	R/W	64b	PMEVCNTR<n>[31:0]	Event Count Register <*n*> (up-to 64 bits), bits[31:0]
0x004+8×*n*	R/W	64b	PMEVCNTR<n>[63:32]	Event Count Register <*n*> (up-to 64 bits), bits[63:32]
0x03C	R/W	32b,CC	PMCCNTR	Cycle Count Register (up-to 32 bits)
0x0F8	R/W	64b,CC	PMCCNTR[31:0]	Cycle Count Register (up-to 64 bits), bits[31:0]
0x0FC	R/W	64b,CC	PMCCNTR[63:32]	Cycle Count Register (up-to 64 bits), bits[63:32]
0x400+4×*n*	R/W		PMEVTYPER<n>	Event Type Select Register <*n*>
0x47C	R/W	CC	PMCCFILTR	Cycle Counter Filter Register
0x600+4×*n*	RO	SS	PMSVR<n>	Saved Value Register <*n*>
0x600+IMP DEF	RO	SS	PMSSSR	Snapshot Status Register
0x600+IMP DEF+4×*n*	RO	SS	PMOVSSR<n>	Overflow Status Snapshot Register <*n*>
0xA00+4×*n*	R/W		PMEVFILTR<n>	Event Type Select Register <*n*>
0xC00+4×*n*	R/W1S		PMCNTENSET<n>	Count Enable Set Register <*n*>
0xC20+4×*n*	R/W1C		PMCNTENCLR<n>	Count Enable Clear Register <*n*>
0xC40+4×*n*	R/W1S		PMINTENSET<n>	Overflow Interrupt Enable Set Register <*n*>
0xC60+4×*n*	R/W1C		PMINTENCLR<n>	Overflow Interrupt Enable Clear Register <*n*>
0xC80+4×*n*	R/W1C		PMOVSCLR<n>	Overflow Status Clear Register <*n*>
0xCC0+4×*n*	R/W1S		PMOVSSET<n>	Overflow Status Set Register <*n*>
0xCE0+4×*n*	RO	MG	PMCGCR<n>	Counter Group Configuration Register <*n*>
0xD80+4×*n*	R/W		IMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE04	R/W		PMCR	Control Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xE20+4×*n*	RO		PMCEID<n>	Common Event Identification Register <*n*>
0xE30	WO	SS	PMSSCR	Snapshot Capture Register
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]
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]

Offset	Access	Version	Register	Description
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		PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO		PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFB8	RO		PMAUTHSTATUS	Authentication Status Register
0xFBC	RO		PMDEVARCH	Device Architecture Register
0xFC8	RO		PMDEVID	Device Configuration Register
0xFCC	RO		PMDEVTYPE	Device Type Register
0xFD0	RO		PMPIDR4	Peripheral Identification Register 4
0xFD4	RO		PMPIDR5	Peripheral Identification Register 5
0xFD8	RO		PMPIDR6	Peripheral Identification Register 6
0xFDC	RO		PMPIDR7	Peripheral Identification Register 7
0xFE0	RO		PMPIDR0	Peripheral Identification Register 0
0xFE4	RO		PMPIDR1	Peripheral Identification Register 1
0xFE8	RO		PMPIDR2	Peripheral Identification Register 2
0xFEC	RO		PMPIDR3	Peripheral Identification Register 3
0xFF0	RO		PMCIDR0	Component Identification Register 0
0xFF4	RO		PMCIDR1	Component Identification Register 1
0xFF8	RO		PMCIDR2	Component Identification Register 2
0xFFC	RO		PMCIDR3	Component Identification Register 3

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

Table 3.2: Memory-mapped register map

Offset	Access	Version	Register	Description
0x400+4×*n*	R/W		PMEVTYPER<n>	Event Type Select Register <*n*>
0x47C	R/W	CC	PMCCFILTR	Cycle Counter Filter Register
0xA00+4×*n*	R/W		PMEVFILTR<n>	Event Type Select Register <*n*>
0xC00+4×*n*	R/W1S		PMCNTENSET<n>	Count Enable Set Register <*n*>
0xC20+4×*n*	R/W1C		PMCNTENCLR<n>	Count Enable Clear Register <*n*>
0xC40+4×*n*	R/W1S		PMINTENSET<n>	Overflow Interrupt Enable Set Register <*n*>
0xC60+4×*n*	R/W1C		PMINTENCLR<n>	Overflow Interrupt Enable Clear Register <*n*>
0xCE0+4×*n*	RO	MG	PMCGCR<n>	Counter Group Configuration Register <*n*>
0xD80+4×*n*	R/W		IMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE04	R/W		PMCR	Control Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xE20+4×*n*	RO		PMCEID<n>	Common Event Identification Register <*n*>
0xE30	WO	SS	PMSSCR	Snapshot Capture Register
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]
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

Offset	Access	Version	Register	Description
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		PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO		PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFB8	RO		PMAUTHSTATUS	Authentication Status Register
0xFBC	RO		PMDEVARCH	Device Architecture Register
0xFC8	RO		PMDEVID	Device Configuration Register
0xFCC	RO		PMDEVTYPE	Device Type Register
0xFD0	RO		PMPIDR4	Peripheral Identification Register 4
0xFD4	RO		PMPIDR5	Peripheral Identification Register 5
0xFD8	RO		PMPIDR6	Peripheral Identification Register 6
0xFDC	RO		PMPIDR7	Peripheral Identification Register 7
0xFE0	RO		PMPIDR0	Peripheral Identification Register 0
0xFE4	RO		PMPIDR1	Peripheral Identification Register 1
0xFE8	RO		PMPIDR2	Peripheral Identification Register 2
0xFEC	RO		PMPIDR3	Peripheral Identification Register 3
0xFF0	RO		PMCIDR0	Component Identification Register 0
0xFF4	RO		PMCIDR1	Component Identification Register 1
0xFF8	RO		PMCIDR2	Component Identification Register 2
0xFFC	RO		PMCIDR3	Component Identification Register 3

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

Table 3.3: 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]
0x03C	R/W	32b,CC	PMCCNTR	Cycle Count Register (up-to 32 bits)
0x0F8	R/W	64b,CC	PMCCNTR[31:0]	Cycle Count Register (up-to 64 bits), bits[31:0]
0x0FC	R/W	64b,CC	PMCCNTR[63:32]	Cycle Count Register (up-to 64 bits), bits[63:32]
0x600+4×*n*	RO	SS	PMSVR<n>	Saved Value Register <*n*>
0x600+IMPDEF	RO	SS	PMSSSR	Snapshot Status Register
0x600+IMPDEF+4×*n*	RO	SS	PMOVSSR<n>	Overflow Status Snapshot Register <*n*>
0xC80+4×*n*	R/W1C		PMOVSCLR<n>	Overflow Status Clear Register <*n*>
0xCC0+4×*n*	R/W1S		PMOVSSET<n>	Overflow Status Set Register <*n*>
0xD80+4×*n*	R/W		IMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0xE00	RO		PMCFGR	Configuration Register
0xE08	RO		PMIIDR	Implementation Identification Register
0xFA8	RO		PMDEVAFF[31:0]	Device Affinity Register, bits[31:0]
0xFAC	RO		PMDEVAFF[63:32]	Device Affinity Register, bits[63:32]
0xFBC	RO		PMDEVARCH	Device Architecture Register
0xFCC	RO		PMDEVTYPE	Device Type Register

Offset	Access	Version	Register	Description
0xFD0	RO		PMPIDR4	Peripheral Identification Register 4
0xFD4	RO		PMPIDR5	Peripheral Identification Register 5
0xFD8	RO		PMPIDR6	Peripheral Identification Register 6
0xFDC	RO		PMPIDR7	Peripheral Identification Register 7
0xFE0	RO		PMPIDR0	Peripheral Identification Register 0
0xFE4	RO		PMPIDR1	Peripheral Identification Register 1
0xFE8	RO		PMPIDR2	Peripheral Identification Register 2
0xFEC	RO		PMPIDR3	Peripheral Identification Register 3
0xFF0	RO		PMCIDR0	Component Identification Register 0
0xFF4	RO		PMCIDR1	Component Identification Register 1
0xFF8	RO		PMCIDR2	Component Identification Register 2
0xFFC	RO		PMCIDR3	Component Identification Register 3

3.2 IMPDEF<*n*>, IMPLEMENTATION DEFINED Register <0-31>

The IMPDEF<0-31> characteristics are:

Purpose

IMPLEMENTATION DEFINED extensions.

Usage constraints

None.

Configurations

If the dual-page extension is implemented, then for each IMPLEMENTATION DEFINED<*n*>, it is IMPLEMENTATION DEFINED whether the register is a Page 0 register, a Page 1 register, or both.

Attributes

IMPLEMENTATION DEFINED<*n*> is a 32-bit read/write memory-mapped register located at offset `0xD80` + 4×*n*.

A pair of IMPLEMENTATION DEFINED<*n*> registers at a doubleword-aligned offset might be defined as a single 64 bit register.

3.3 PMAUTHSTATUS, Authentication Status Register

The PMAUTHSTATUS characteristics are:

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

Usage constraints
None.

Configurations
Always implemented.

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

3.3.1 Field descriptions

The PMAUTHSTATUS bit assignments are:

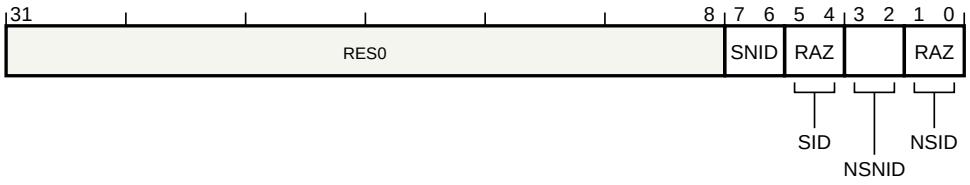


Figure 3.1: PMAUTHSTATUS

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

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

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

All other values are reserved.

This field reads-as-zeros if Security Extensions are not implemented and the *Performance Monitoring Unit* (PMU) is Non-secure.

SID, bits [5:4]
Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled. The possible values of this field are:

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

All other values are reserved.

This field reads-as-zero.

NSNID, bits [3:2]

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

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

All other values are reserved.

This field reads-as-zeros if Security Extensions are not implemented and the PMU is Secure.

NSID, bits [1:0]

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

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

All other values are reserved.

This field reads-as-zero.

3.4 PMCCFILTR, Cycle Counter Filter Register

The PMCCFILTR characteristics are:

Purpose
Configures the Cycle Counter.

Usage constraints
None.

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

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

3.4.1 Field descriptions

The PMCCFILTR bit assignments are:

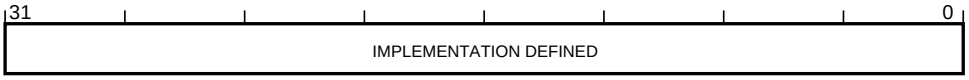


Figure 3.2: PMCCFILTR

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

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

The PMCCNTR characteristics are:

Purpose

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

Usage constraints

None.

Configurations

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

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

PMCCNTR is RES0 otherwise.

Attributes

PMCCNTR is a 32-bit read/write memory-mapped register located at offset `0x03C` in the Page 1 component.

3.5.1 Field descriptions

The PMCCNTR bit assignments are:

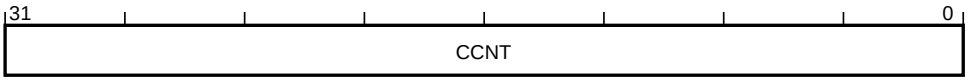


Figure 3.3: PMCCNTR

CCNT, bits [31:0]

Cycle Counter.

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

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

The PMCCNTR characteristics are:

Purpose

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

Usage constraints

None.

Configurations

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

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

PMCCNTR is RES0 otherwise.

Attributes

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

3.6.1 Field descriptions

The PMCCNTR bit assignments are:

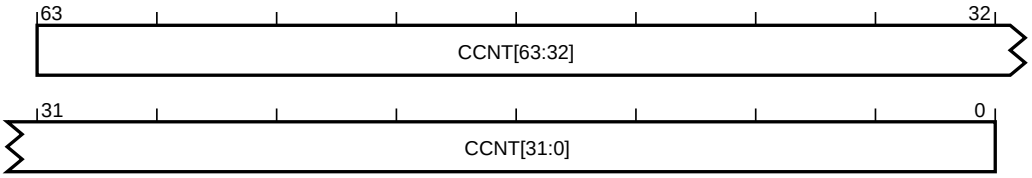


Figure 3.4: PMCCNTR

CCNT, bits [63:0]

Cycle Counter.

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

3.7 PMCEID<*n*>, Common Event Identification Register <0-3>

The PMCEID<0-3> characteristics are:

Purpose
Provide a mechanism for an implementation to describe which events it supports.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMCEID<*n*> is a 32-bit read-only memory-mapped register located at offset `0xE20` + 4×*n* in the Page 0 component.

3.7.1 Field descriptions

The PMCEID<0-3> bit assignments are:

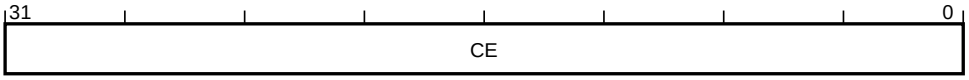


Figure 3.5: PMCEID<*n*>

CE[*m*], bit [*q*], for *q* = 0 to 31, where *m* = *q*+*x*
Common Event *m* implemented. The defined values of this bit are:

`0`	The Common event is not implemented or not counted.
`1`	The Common event is implemented.

The base value *x* for each PMCEID<*n*> register is IMPLEMENTATION DEFINED.

3.8 PMCFGR, Configuration Register

The PMCFGR characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
None.

Configurations
Always implemented.

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

3.8.1 Field descriptions

The PMCFGR bit assignments are:

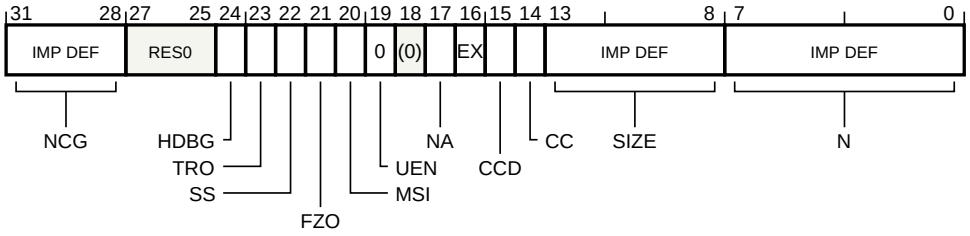


Figure 3.6: PMCFGR

NCG, bits [31:28]
Monitor Groups.

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

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

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

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

This field reads as an IMPLEMENTATION DEFINED value.

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

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

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

0	Halt-on-debug feature not supported.
1	Halt-on-debug feature supported.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

TRO, bit [23]
Trace features supported.

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

0	Trace features not supported.
1	Trace features supported.

The nature of any supported trace features are IMPLEMENTATION DEFINED.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

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

0	Snapshot mechanism not supported. PMSSCR and PMSSRR is reserved. The locations 0x600-0x7FC are IMPLEMENTATION DEFINED.
1	Snapshot mechanism supported. PMSVR<n>, PMSSCR and PMSSSR are implemented.

If the architecture-defined form of snapshot is not implemented, a PMU might nonetheless implement a snapshot mechanism, including one located at the IMPLEMENTATION DEFINED registers 0x600-0x6FC.

This bit reads as an IMPLEMENTATION DEFINED value.

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

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

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

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

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

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

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

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

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

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

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

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

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

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

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

EX, bit [16]
Export supported.

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

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

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
Reserved. This bit is RES0.

CCD, bit [15]
Cycle counter has pre-scale.

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

0	Cycle counter only ever counts every cycle. PMCR.D is RES0.

1	Cycle counter can count every 64th cycle. PMCR.D is read/write.

This bit reads as an IMPLEMENTATION DEFINED value.

Otherwise
This bit reads-as-zero.

CC, bit [14]
Dedicated Cycle counter implemented as PMEVCNTR31. The defined values of this bit are:

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

This bit reads as an IMPLEMENTATION DEFINED value.

SIZE, bits [13:8]
Monitor size. The size of the largest implemented monitor. The defined values of this field are:

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

All other values are reserved.

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

This field reads as an IMPLEMENTATION DEFINED value.

N, bits [7:0]
Number of monitors, minus one. The defined values of this field are:

0x00	1 monitor.
0x01	2 monitors.
..	..
0xFF	256 monitors.

If PMCFGR.CC == 0b1, PMEVCNTR31, the cycle counter, is one of the (N+1) monitors. For example, if PMCFGR.N == 0x00 and PMCFGR.CC == 0b1, there is a single monitor, PMEVCNTR31, and

PMEVCNTR0 is not implemented.

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

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

If the Snapshot mechanism is implemented, then the implementation includes at most 256 bytes of snapshot values, which equates to thirty-two 64-bit or sixty-four 32-bit monitors, although the snapshot values might include other, non-monitor, values.

This field reads as an IMPLEMENTATION DEFINED value.

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

The PMCGCR<0-3> characteristics are:

Purpose
Describes the performance monitor.

Usage constraints
None.

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

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

3.9.1 Field descriptions

The PMCGCR<0-3> bit assignments are:

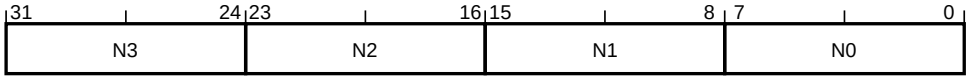


Figure 3.7: PMCGCR<*n*>

N*m*[*m*×8+7:*m*×8], bits [*m*×8+7:*m*×8], for *m* = 0 to 3
Number of monitors in group *n*×4+*m*.

The maximum size of each monitor group depends on the number of implemented groups and the largest implemented monitor size. Each monitor group starts with monitor:

- 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.
- 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 more than 4 monitor groups.
- There are at most 8 monitors per group if monitors are larger-than 32-bits and there are more than 8 monitor groups.

3.10 PMCIDR0, Component Identification Register 0

The PMCIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

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

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

3.10.1 Field descriptions

The PMCIDR0 bit assignments are:

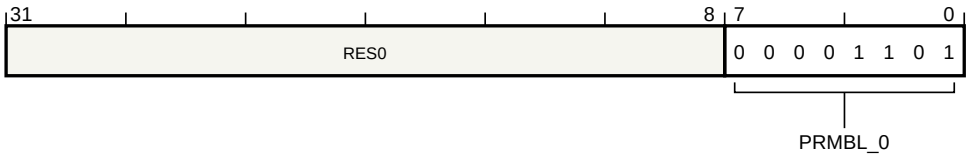


Figure 3.8: PMCIDR0

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

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

3.11 PMCIDR1, Component Identification Register 1

The PMCIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

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

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

3.11.1 Field descriptions

The PMCIDR1 bit assignments are:

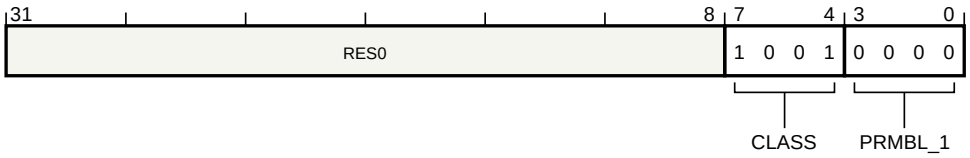


Figure 3.9: PMCIDR1

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

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

`0x9`	CoreSight peripheral.

Other values are defined by the CoreSight Architecture.

This field reads as `0x9`.

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

3.12 PMCIDR2, Component Identification Register 2

The PMCIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

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

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

3.12.1 Field descriptions

The PMCIDR2 bit assignments are:

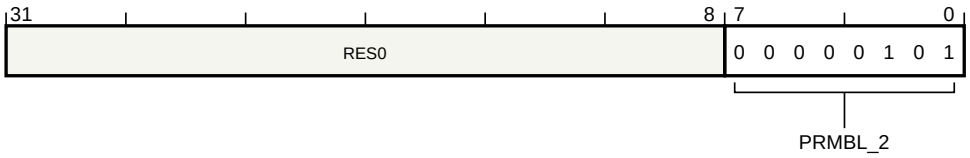


Figure 3.10: PMCIDR2

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

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

3.13 PMCIDR3, Component Identification Register 3

The PMCIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

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

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

3.13.1 Field descriptions

The PMCIDR3 bit assignments are:

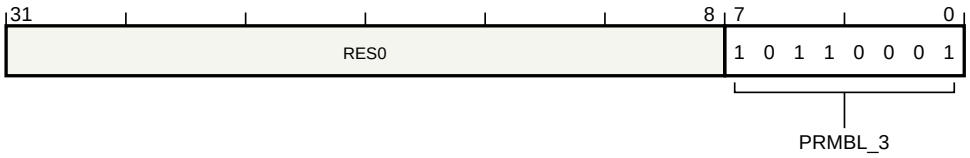


Figure 3.11: PMCIDR3

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

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

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

The PMCNTENCLR<*n*> characteristics are:

Purpose
Disable monitors.

Usage constraints
None.

Configurations
Always implemented.

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

3.14.1 Field descriptions

The PMCNTENCLR<*n*> bit assignments are:

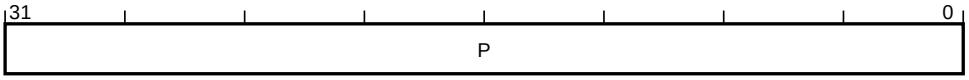


Figure 3.12: PMCNTENCLR<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
PMEVCNTR<m> disable. On writes, allows software to disable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status.

On a read, the defined values of this bit are:

`0`	PMEVCNTR<m> disabled.
`1`	PMEVCNTR<m> enabled.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Disable PMEVCNTR<m>.

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

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

The PMCNTENSET<*n*> characteristics are:

Purpose
Enable monitors.

Usage constraints
None.

Configurations
Always implemented.

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

3.15.1 Field descriptions

The PMCNTENSET<*n*> bit assignments are:

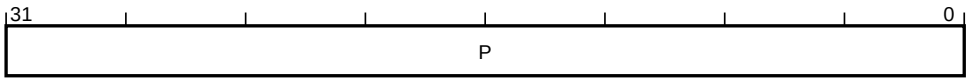


Figure 3.13: PMCNTENSET<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
PMEVCNTR<m> enable. On writes, allows software to enable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status.

On a read, the defined values of this bit are:

`0`	PMEVCNTR<m> disabled.
`1`	PMEVCNTR<m> enabled.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Enable PMEVCNTR<m>.

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

3.16 PMCR, Control Register

The PMCR characteristics are:

Purpose
Main control register for the performance monitors.

Usage constraints
None.

Configurations
Always implemented.

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

3.16.1 Field descriptions

The PMCR bit assignments are:

Figure 3.14: PMCR

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

TRO, bit [11]
Trace enable.

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

`0`	Trace disabled.
`1`	Trace enabled.

Otherwise
Reserved. This bit is RES0.

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

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

`0`	Do not stop counting when agent is halted.
`1`	Stop counting when agent is halted.

Otherwise
Reserved. This bit is RES0.

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

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

0	Do not freeze on overflow.
1	Monitors do not count when PMOVSR is nonzero.

Otherwise
Reserved. This bit is RES0.

NA, bit [8]
Not accessible.

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

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

This bit is read-only.

Otherwise
Reserved. This bit is RES0.

DP, bit [5]
Disable cycle counter when prohibited or not counting.

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

0	Cycle counting by PMCCNTR not affected by this bit.
1	Cycle counting by PMCCNTR disabled in prohibited regions.

Prohibited regions are IMPLEMENTATION DEFINED regions where event counting is prohibited.

Note:

This bit allows software to ignore cycle counts that might be accumulated during periods when the other counts are prohibited because of security prohibitions. It is not a control to enhance security. The function of this bit is to avoid corruption of the count.

Otherwise
Reserved. This bit is RES0.

X, bit [4]
Export enable.

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

0	Export of events is disabled.
1	Export of events is enabled.

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

Otherwise
Reserved. This bit is RES0.

D, bit [3]
Cycle counter divider.

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

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

Otherwise
Reserved. This bit is RES0.

C, bit [2]
Cycle counter reset.

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

0	Write is ignored.
1	Reset the cycle counter to zero.

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

Note:

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

Otherwise
Reserved. This bit is RES0.

P, bit [1]
Monitor reset. The possible values for writing to this bit are:

0	Write is ignored.
1	Reset all monitors to zero. The cycle counter is not reset.

Resetting the monitors does not affect any overflow flags.

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

E, bit [0]
Count enable. Controls the performance monitor. The possible values of this bit are:

0	All monitors are disabled.
1	All monitors are enabled by PMCNTENSET.

This bit resets to zero.

3.17 PMDEVAFF, Device Affinity Register

The PMDEVAFF characteristics are:

Purpose

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

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

Usage constraints

None.

Configurations

PMDEVAFF is present only if the monitor has affinity with a PE or cluster of PEs. PMDEVAFF is RES0 otherwise.

Attributes

PMDEVAFF is a 64-bit read-only memory-mapped register located at offset `0xFA8`.

3.17.1 Field descriptions

The PMDEVAFF bit assignments are:

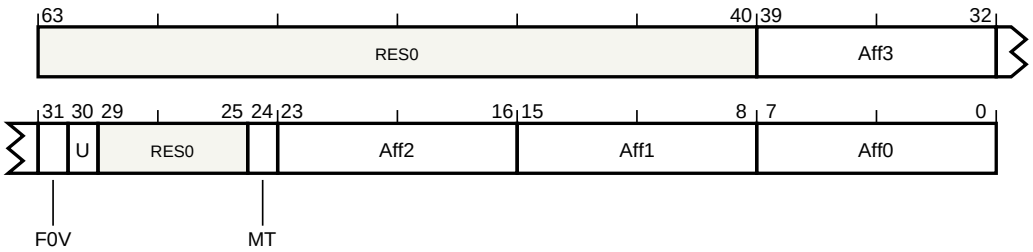


Figure 3.15: PMDEVAFF

Bits [63:40,29:25]

Reserved. This field is RES0.

Aff3, bits [39:32]

PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

F0V, bit [31]

Indicates that the PMDEVAFF.Aff0 field is valid. The defined values of this bit are:

0b0	PMDEVAFF.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
0b1	PMDEVAFF.Aff0 is valid, and the PE affinity is at level 0.

U, bit [30]
Uniprocessor.

When PMDEVAFF.F0V == 0b1
The MPIDR_EL1.U bit, viewed from the highest Exception level of the associated PE.

Otherwise
Reserved. This bit is UNKNOWN.

MT, bit [24]
Multithreaded.

When PMDEVAFF.F0V == 0b1
The MPIDR_EL1.MT bit, viewed from the highest Exception level of the associated PE.

Otherwise
Reserved. This bit is UNKNOWN.

Aff2, bits [23:16]
PE affinity level 2.

When affine with a PE or PEs at affinity level 2 or below
The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 2
Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

0bxxxxxxx1	PMDEVAFF.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PMDEVAFF.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PMDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PMDEVAFF.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PMDEVAFF.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PMDEVAFF.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PMDEVAFF.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.

Otherwise
Indicates whether the PE affinity is at level 3. The defined values of this field are:

0x80	PE affinity is at level 3.

All other values are reserved.

Aff1, bits [15:8]
PE affinity level 1.

When affine with a PE or PEs at affinity level 1 or below
The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 1
Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

`0bxxxxxxx1`	PMDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
`0bxxxxxx10`	PMDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
`0bxxxxx100`	PMDEVAFF.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
`0bxxxx1000`	PMDEVAFF.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
`0bxxx10000`	PMDEVAFF.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
`0bxx100000`	PMDEVAFF.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
`0bx1000000`	PMDEVAFF.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.

Otherwise
Indicates whether the PE affinity is at level 2. The defined values of this field are:

`0x00`	PE affinity is above level 2 or a subset of level 2.
`0x80`	PE affinity is at level 2.

Aff0, bits [7:0]
PE affinity level 0.

When affine with a PE at affinity level 0
The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

When affine with a sub-set of PEs at affinity level 0
Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

`0bxxxxxxx1`	PMDEVAFF.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
`0bxxxxxx10`	PMDEVAFF.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
`0bxxxxx100`	PMDEVAFF.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
`0bxxxx1000`	PMDEVAFF.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
`0bxxx10000`	PMDEVAFF.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
`0bxx100000`	PMDEVAFF.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
`0bx1000000`	PMDEVAFF.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.

Otherwise

Indicates whether the PE affinity is at level 1. The defined values of this field are:

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

3.18 PMDEVARCH, Device Architecture Register

The PMDEVARCH characteristics are:

Purpose
Provides discovery information for the component.

Usage constraints
None.

Configurations
Always implemented.

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

3.18.1 Field descriptions

The PMDEVARCH bit assignments are:

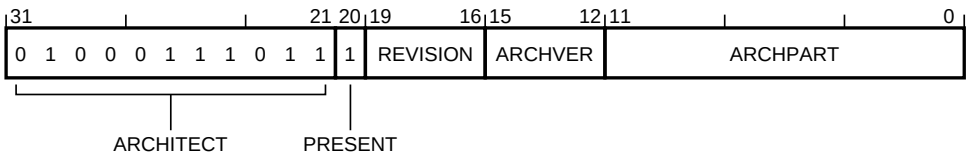


Figure 3.16: PMDEVARCH

ARCHITECT, bits [31:21]
Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code. The defined values of this field are:

`0x23B`	JEP106 continuation code `0x4`, ID code `0x3B`. Arm Limited.

Other values are defined by the JEDEC JEP106 standard.

This field reads as `0x23B`.

PRESENT, bit [20]
DEVARCH Present. Defines that the DEVARCH register is present. The defined values of this bit are:

`0b0`	Device Architecture information not present.
`0b1`	Device Architecture information present.

This bit reads as `0b1`.

REVISION, bits [19:16]
Revision. Defines the architecture revision of the component.

ARCHVER, bits [15:12]
Architecture Version. Defines the architecture version of the component.

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

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

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

3.19 PMDEVID, Device Configuration Register

The PMDEVID characteristics are:

Purpose
Provides discovery information for the component.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMDEVID is a 32-bit read-only memory-mapped register located at offset `0xFC8` in the Page 0 component.

3.19.1 Field descriptions

The PMDEVID bit assignments are:

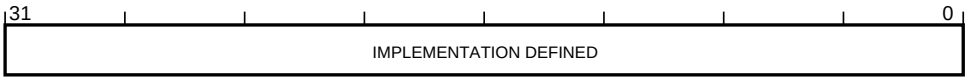


Figure 3.17: PMDEVID

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value.

3.20 PMDEVTYPE, Device Type Register

The PMDEVTYPE characteristics are:

Purpose
Provides discovery information for the component. If the part number field is not recognized, a debugger can report the information that is provided by PMDEVTYPE about the component instead.

Usage constraints
None.

Configurations
Always implemented.

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

3.20.1 Field descriptions

The PMDEVTYPE bit assignments are:

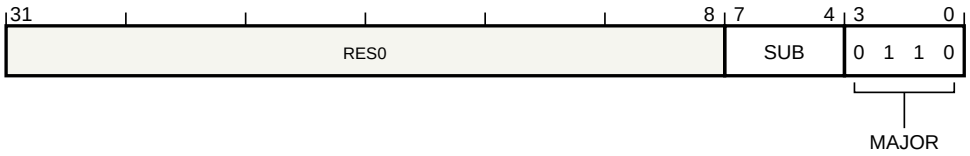


Figure 3.18: PMDEVTYPE

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

SUB, bits [7:4]
Component sub-type.

When MAJOR == `0x6` (Performance monitor), the defined values of this field are:

`0x0`	Other.
`0x1`	Associated with a PE.
`0x2`	Associated with a DSP.
`0x3`	Associated with a Data Engine or coprocessor.
`0x4`	Associated with a bus or stimulus derived from bus activity.
`0x5`	Associated with a memory management unit conforming to the Arm System MMU architecture.
`0x7`	Derived from generic signals.

Other values are defined by the CoreSight Architecture.

MAJOR, bits [3:0]
Component major type. The defined values of this field are:

`0x6`	Performance monitor.

Other values are defined by the CoreSight Architecture.

This field reads as `0x6`.

3.21 PMEVCNTR<*n*>, Event Count Register <*n*> (up-to 32 bits)

The PMEVCNTR<*n*> characteristics are:

Purpose

Monitor value <*n*>. If monitoring is not prohibited and the monitor is enabled, the monitor monitors the component as configured by PMEVTYPER<n> and PMEVFILTR<n>.

Usage constraints

None.

Configurations

PMEVCNTR<*n*> is present only if PMCFGR.SIZE <= `0b011111`, all monitors are 32 bits or smaller. PMEVCNTR<*n*> is RES0 otherwise.

Attributes

PMEVCNTR<*n*> is a 32-bit read/write memory-mapped register located at offset `0x000` + 4×*n* in the Page 1 component.

3.21.1 Field descriptions

The PMEVCNTR<*n*> bit assignments are:

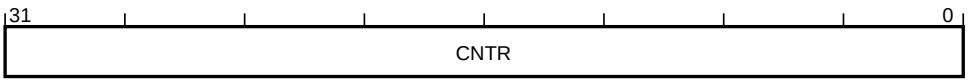


Figure 3.19: PMEVCNTR<*n*>

CNTR, bits [31:0]

Monitor value.

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

3.22 PMEVCNTR<*n*>, Event Count Register <*n*> (up-to 64 bits)

The PMEVCNTR<*n*> characteristics are:

Purpose

Monitor value <*n*>. If monitoring is not prohibited and the monitor is enabled, the monitor monitors the component as configured by PMEVTYPER<n> and PMEVFILTR<n>.

Usage constraints

None.

Configurations

PMEVCNTR<*n*> is present only if PMCFGR.SIZE > `0b011111`, at least one monitor is larger than 32 bits. PMEVCNTR<*n*> is RES0 otherwise.

Attributes

PMEVCNTR<*n*> is a 64-bit read/write memory-mapped register located at offset `0x000` + 8×*n* in the Page 1 component.

3.22.1 Field descriptions

The PMEVCNTR<*n*> bit assignments are:

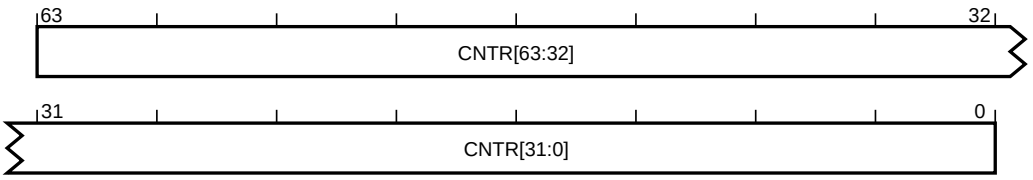


Figure 3.20: PMEVCNTR<*n*>

CNTR, bits [63:0]

Monitor value.

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

3.23 PMEVFILTR<*n*>, Event Type Select Register <*n*>

The PMEVFILTR<*n*> characteristics are:

Purpose
For performance monitors requiring event selection controls in addition to the 32-bit PMEVTYPER<n> register.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMEVFILTR<*n*> is a 32-bit read/write memory-mapped register located at offset `0xA00` + 4×*n* in the Page 0 component.

It is IMPLEMENTATION DEFINED whether this register is read-only or read/write.

3.23.1 Field descriptions

The PMEVFILTR<*n*> bit assignments are:

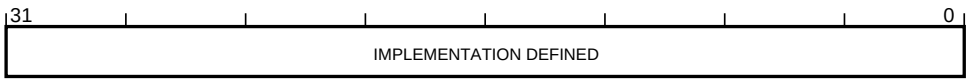


Figure 3.21: PMEVFILTR<*n*>

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

3.24 PMEVTYPER<*n*>, Event Type Select Register <*n*>

The PMEVTYPER<*n*> characteristics are:

Purpose
Configures monitor <*n*>.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMEVTYPER<*n*> is a 32-bit read/write memory-mapped register located at offset `0x400` + 4×*n* in the Page 0 component.

It is IMPLEMENTATION DEFINED whether this register is read-only or read/write.

3.24.1 Field descriptions

The PMEVTYPER<*n*> bit assignments are:

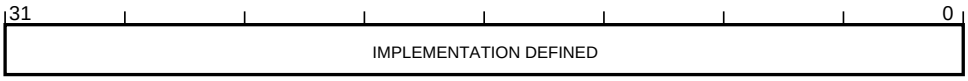


Figure 3.22: PMEVTYPER<*n*>

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

3.25 PMIIDR, Implementation Identification Register

The PMIIDR characteristics are:

Purpose
Defines the implementer of the component.

Usage constraints
None.

Configurations
It is IMPLEMENTATION DEFINED whether PMIIDR is present. PMIIDR is RES0 if not present.

Attributes
PMIIDR is a 32-bit read-only memory-mapped register located at offset `0xE08`.

3.25.1 Field descriptions

The PMIIDR bit assignments are:

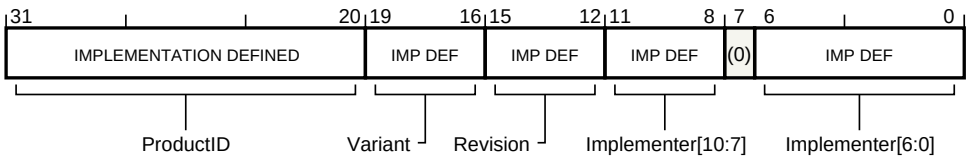


Figure 3.23: PMIIDR

ProductID, bits [31:20]
Part number, bits [11:0]. The part number is selected by the designer of the component.

Matches the {PMPIDR1.PART_1,PMPIDR0.PART_0} fields, if PMPIDR0 and PMPIDR1 are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Variant, bits [19:16]
Component major revision.

PMIIDR.Variant defines either a variant of the component defined by PMIIDR.ProductID, or the major revision of the component.

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

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

This field reads as an IMPLEMENTATION DEFINED value.

Revision, bits [15:12]
Component minor revision.

When a component is changed:

- If PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component then:
  - PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.
  - If Variant is increased then Revision should be set to `0b0000`.

- Otherwise, PMIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.

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

This field reads as an IMPLEMENTATION DEFINED value.

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

JEDEC-assigned JEP106 identification code. PMIIDR[11:8] is the JEP106 bank identifier minus 1 and PMIIDR[6:0] is the JEP106 identification code for the designer of the component. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

PMIIDR[11:8] matches PMPIDR4.DES_2 and PMIIDR[6:0] match the {PMPIDR2.DES_1,ERRPIDR1.DES_0} fields, if PMPIDR{1,2,4} are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

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.

Bit [7]

Reserved. This bit is RES0.

3.26 PMINTENCLR<*n*>, Overflow Interrupt Enable Clear Register <*n*>

The PMINTENCLR<*n*> characteristics are:

Purpose
Disable interrupt on monitor unsigned overflow.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMINTENCLR<*n*> is a 32-bit read/write-one-to-clear memory-mapped register located at offset `0xC60` + 4×*n* in the Page 0 component.

3.26.1 Field descriptions

The PMINTENCLR<*n*> bit assignments are:

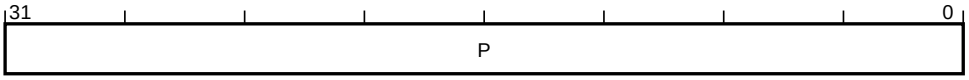


Figure 3.24: PMINTENCLR<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
Interrupt on overflow status of PMEVCNTR<m> disable. On writes, allows software to disable the interrupt on overflow status of PMEVCNTR<m>. On reads, returns the interrupt on overflow status of PMEVCNTR<m> enable status.

On a read, the defined values of this bit are:

`0`	Interrupt on overflow status of PMEVCNTR<m> disabled.
`1`	Interrupt on overflow status of PMEVCNTR<m> enabled.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Disable interrupt on overflow status of PMEVCNTR<m>.

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

3.27 PMINTENSET<*n*>, Overflow Interrupt Enable Set Register <*n*>

The PMINTENSET<*n*> characteristics are:

Purpose
Enable interrupt on monitor unsigned overflow.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMINTENSET<*n*> is a 32-bit read/write-one-to-set memory-mapped register located at offset `0xC40` + 4×*n* in the Page 0 component.

3.27.1 Field descriptions

The PMINTENSET<*n*> bit assignments are:

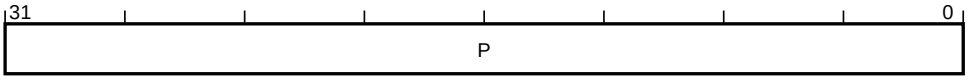


Figure 3.25: PMINTENSET<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
Interrupt on overflow status of PMEVCNTR<m> enable. On writes, allows software to enable the interrupt on overflow status of PMEVCNTR<m>. On reads, returns the interrupt on overflow status of PMEVCNTR<m> enable status.

On a read, the defined values of this bit are:

`0`	Interrupt on overflow status of PMEVCNTR<m> disabled.
`1`	Interrupt on overflow status of PMEVCNTR<m> enabled.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Enable interrupt on overflow status of PMEVCNTR<m>.

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

3.28 PMIRQCR0, Interrupt Configuration Register 0

The PMIRQCR0 characteristics are:

Purpose
Interrupt configuration register.

Usage constraints
None.

Configurations
PMIRQCR0 is present only if PMCFGR.MSI == `0b1`, message-signaled interrupts are implemented. PMIRQCR0 is RES0 otherwise.

Attributes
PMIRQCR0 is a 64-bit read/write memory-mapped register located at offset `0xE80` in the Page 0 component.

3.28.1 Field descriptions

The PMIRQCR0 bit assignments are:

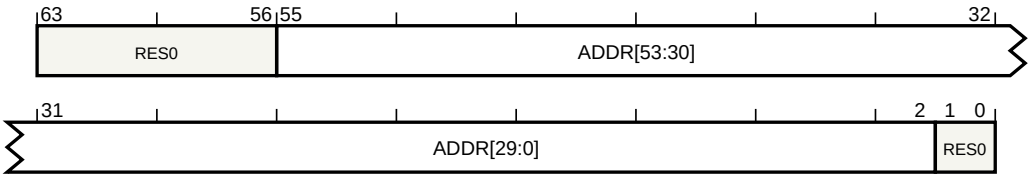


Figure 3.26: PMIRQCR0

Bits [63:56,1:0]
Reserved. This field is RES0.

ADDR, bits [55:2]
Message Signaled Interrupt address. (PMIRQCR0.ADDR << 2) is the address that the PMU writes to when signaling the Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the PMU is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

3.29 PMIRQCR1, Interrupt Configuration Register 1

The PMIRQCR1 characteristics are:

Purpose
Interrupt configuration register.

Usage constraints
If this component allows both Secure and Non-secure accesses, then when PMIRQCR2.NSMSI is implemented and set to `0b0` PMIRQCR1 is read-only to Non-secure accesses.

Configurations
PMIRQCR1 is present only if PMCFGR.MSI == `0b1`, message-signaled interrupts are implemented. PMIRQCR1 is RES0 otherwise.

Attributes
PMIRQCR1 is a 32-bit read/write memory-mapped register located at offset `0xE88` in the Page 0 component.

3.29.1 Field descriptions

The PMIRQCR1 bit assignments are:

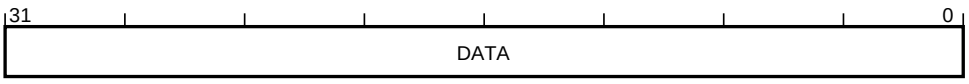


Figure 3.27: PMIRQCR1

DATA, bits [31:0]
Payload for the message signaled interrupt.

3.30 PMIRQCR2, Interrupt Configuration Register 2

The PMIRQCR2 characteristics are:

Purpose
Interrupt control and configuration register.

Usage constraints
If this component allows both Secure and Non-secure accesses, then when PMIRQCR2.NSMSI is implemented and set to `0b0` PMIRQCR2 is read-only to Non-secure accesses.

Configurations
PMIRQCR2 is present only if PMCFGR.MSI == `0b1`, message-signaled interrupts are implemented. PMIRQCR2 is RES0 otherwise.

Attributes
PMIRQCR2 is a 32-bit read/write memory-mapped register located at offset `0xE8C` in the Page 0 component.

3.30.1 Field descriptions

The PMIRQCR2 bit assignments are:

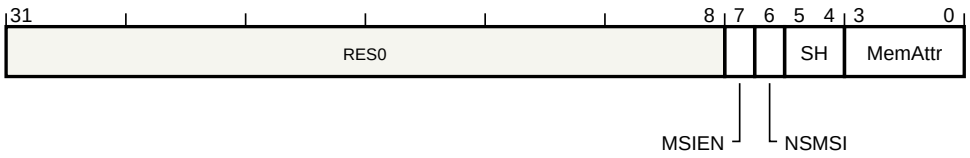


Figure 3.28: PMIRQCR2

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

MSIEN, bit [7]
Message signaled interrupt enable.

When the PMU supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

`0b0`	Disabled.
`0b1`	Enabled.

This bit resets to `0b0`.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [6]
Security attribute. Defines the physical address space for message signaled interrupts.

When the PMU supports configuring the Security attribute for message signaled interrupts, and the PMU does not allow Non-secure writes to PMIRQCR2
The possible values of this bit are:

0b0	Secure.
0b1	Non-secure.

This bit resets to an IMPLEMENTATION DEFINED value.

When the PMU allows Non-secure writes to PMIRQCR2
The Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0.

Otherwise
The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [5:4]
Shareability.

When the PMU supports configuring the Shareability domain for message signaled interrupts
Defines the Shareability domain for message signaled interrupts. The possible values of this field are:

0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when PMIRQCR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

This field resets to an architecturally UNKNOWN value.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [3:0]
Memory type.

When the PMU supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.

0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

Note:

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

Otherwise

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

This field is RES0.

3.31 PMIRQSR, Interrupt Status Register

The PMIRQSR characteristics are:

Purpose
Interrupt status register.

Usage constraints
If this component allows both Secure and Non-secure accesses, then when PMIRQCR2.NSMSI is implemented and set to `0b0` PMIRQSR is read-only to Non-secure accesses.

Configurations
PMIRQSR is present only if PMCFGR.MSI == `0b1`, message-signaled interrupts are implemented. PMIRQSR is RES0 otherwise.

Attributes
PMIRQSR is a 64-bit read/write memory-mapped register located at offset `0xEF8` in the Page 0 component.

3.31.1 Field descriptions

The PMIRQSR bit assignments are:

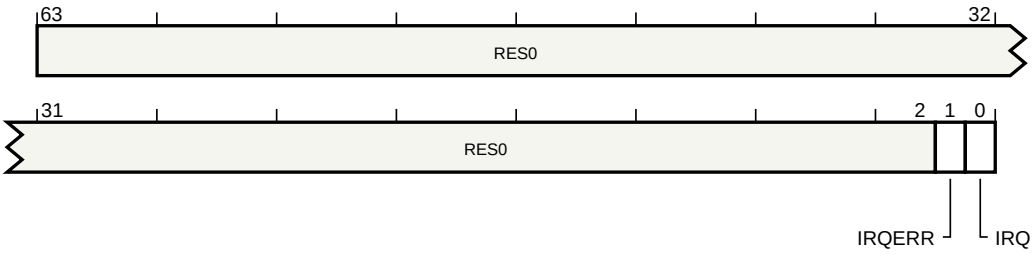


Figure 3.29: PMIRQSR

Bits [63:2]
Reserved. This field is RES0.

IRQERR, bit [1]
Interrupt error. The possible values of this bit are:

`0b0`	Interrupt write has not returned an error since this bit was last cleared to zero.
`0b1`	Interrupt write has returned an error since this bit was last cleared to zero.

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

IRQ, bit [0]
PMU Overflow Interrupt write in progress. The defined values of this bit are:

`0b0`	PMU Overflow Interrupt write not in progress.
`0b1`	PMU Overflow Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

Note:

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual PMOVSR<*n*> and PMINTEN<*n*> registers.

3.32 PMOVSCLR<*n*>, Overflow Status Clear Register <*n*>

The PMOVSCLR<*n*> characteristics are:

Purpose
Clear PMU monitor overflow status flags.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMOVSCLR<*n*> is a 32-bit read/write-one-to-clear memory-mapped register located at offset `0xC80` + 4×*n* in the Page 1 component.

3.32.1 Field descriptions

The PMOVSCLR<*n*> bit assignments are:

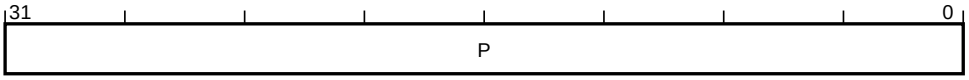


Figure 3.30: PMOVSCLR<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
Overflow status flag for PMEVCNTR<m> clear. On writes, allows software to clear the overflow status flag for PMEVCNTR<m> to 0. On reads, returns the overflow status flag for PMEVCNTR<m>.

On a read, the defined values of this bit are:

`0`	PMEVCNTR<m> has not overflowed.
`1`	PMEVCNTR<m> has overflowed.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Clear overflow status flag for PMEVCNTR<m> to 0.

If the Cycle Counter extension is implemented, then PMOVSCLR<*n*>.P[31] allows software to clear the overflow status flag for PMCCNTR to 0 and query the overflow status flag for PMCCNTR.

3.33 PMOVSSET<*n*>, Overflow Status Set Register <*n*>

The PMOVSSET<*n*> characteristics are:

Purpose
Set PMU monitor overflow status flags.

Usage constraints
None.

Configurations
Always implemented.

Attributes
PMOVSSET<*n*> is a 32-bit read/write-one-to-set memory-mapped register located at offset `0xCC0` + 4×*n* in the Page 1 component.

3.33.1 Field descriptions

The PMOVSSET<*n*> bit assignments are:

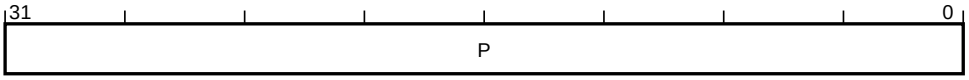


Figure 3.31: PMOVSSET<*n*>

P[*m*], bit [*m*], for *m* = 0 to 31
Overflow status flag for PMEVCNTR<m> set. On writes, allows software to set the overflow status flag for PMEVCNTR<m> to 1. On reads, returns the overflow status flag for PMEVCNTR<m>.

On a read, the defined values of this bit are:

`0`	PMEVCNTR<m> has not overflowed.
`1`	PMEVCNTR<m> has overflowed.

On a write, the possible values for writing to this bit are:

`0`	Write is ignored.
`1`	Set overflow status flag for PMEVCNTR<m> to 1.

If the Cycle Counter extension is implemented, then PMOVSSET<*n*>.P[31] allows software to set the overflow status flag for PMCCNTR to 1 and query the overflow status flag for PMCCNTR.

3.34 PMOVSSR<*n*>, Overflow Status Snapshot Register <*n*>

The PMOVSSR<*n*> characteristics are:

Purpose

Captured copy of PMOVSR. Once captured, the value in PMOVSSR is unaffected by writes to PMOVSSET and PMOVSCLR.

Usage constraints

PMOVSSR<*n*> is one of the PMSVR<n> registers. The location of PMOVSSR<*n*> within the PMSVR<n> registers is IMPLEMENTATION DEFINED.

Configurations

PMOVSSR<*n*> is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMOVSSR<*n*> is RES0 otherwise.

PMOVSSR<*n*> is an optional one of the PMSVR<n> registers. If PMSSRR is implemented, Arm recommends that PMOVSSR<*n*> is implemented, as this indicates whether a counter that is reset has overflowed during the sampling period. If PMSSRR is not implemented, counters are free-running across samples without being reset and could overflow at any time, meaning there is less benefit from sampling PMOVSR.

Attributes

PMOVSSR<*n*> is a 32-bit read-only memory-mapped register located at offset `0x600` + IMPLEMENTATION DEFINED + 4×*n* in the Page 1 component.

3.35 PMPIDR0, Peripheral Identification Register 0

The PMPIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR0 is present only if Peripheral Identification scheme is implemented. PMPIDR0 is RES0 otherwise.

Attributes
PMPIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFE0`.

3.35.1 Field descriptions

The PMPIDR0 bit assignments are:

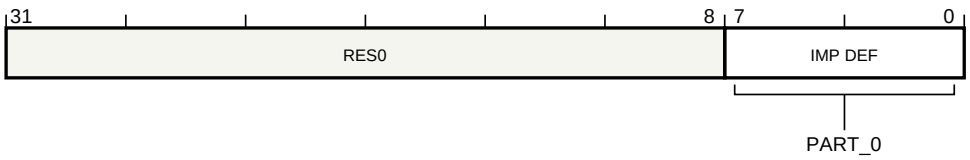


Figure 3.32: PMPIDR0

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

PART_0, bits [7:0]
Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, 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, it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

3.36 PMPIDR1, Peripheral Identification Register 1

The PMPIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR1 is present only if Peripheral Identification scheme is implemented. PMPIDR1 is RES0 otherwise.

Attributes
PMPIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFE4`.

3.36.1 Field descriptions

The PMPIDR1 bit assignments are:

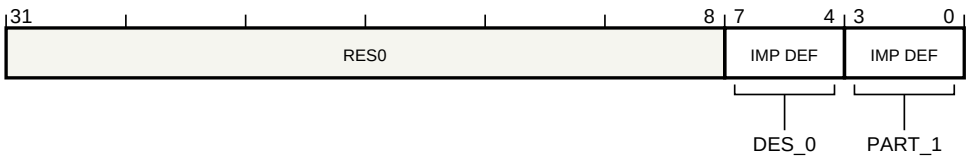


Figure 3.33: PMPIDR1

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

DES_0, bits [7:4]
Designer, JEP106 identification code, bits [3:0]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

PART_1, bits [3:0]
Part number, bits [11:8].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, 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, it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

3.37 PMPIDR2, Peripheral Identification Register 2

The PMPIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR2 is present only if Peripheral Identification scheme is implemented. PMPIDR2 is RES0 otherwise.

Attributes
PMPIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFE8`.

3.37.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

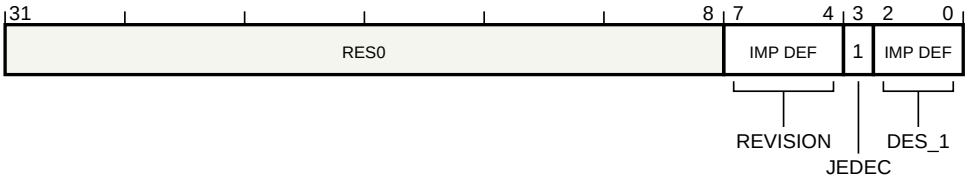


Figure 3.34: PMPIDR2 the component uses a 12-bit part number

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

REVISION, bits [7:4]
Component major revision. PMPIDR2.REVISION and PMPIDR3.REVAND together form the revision number of the component, with PMPIDR2.REVISION being the most significant part and PMPIDR3.REVAND the least significant part. When a component is changed, PMPIDR2.REVISION or PMPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. If PMPIDR2.REVISION is increased then PMPIDR3.REVAND should be set to `0b0000`.

This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]
JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

DES_1, bits [2:0]
Designer, JEP106 identification code, bits [6:4]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

3.37.2 The component uses a 16-bit part number

Configurations

Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

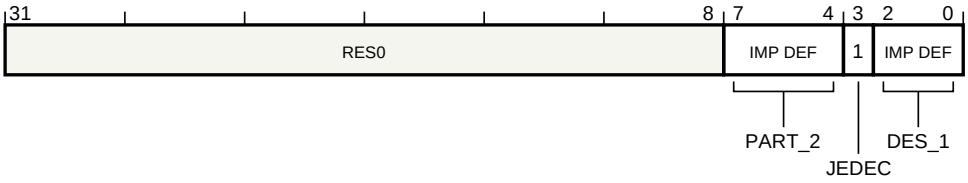


Figure 3.35: PMPIDR2 the component uses a 16-bit part number

Bits [31:8]

Reserved. This field is RES0.

PART_2, bits [7:4]

Part number, bits [15:12].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, 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, it is stored in PMPIDR2.PART_2, PMPIDR1.PART_1 and PMPIDR0.PART_0. There are 4 bits, PMPIDR3.REVISION, available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. PMPIDR1.DES_0 and PMPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

3.38 PMPIDR3, Peripheral Identification Register 3

The PMPIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR3 is present only if Peripheral Identification scheme is implemented. PMPIDR3 is RES0 otherwise.

Attributes
PMPIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFEC`.

3.38.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

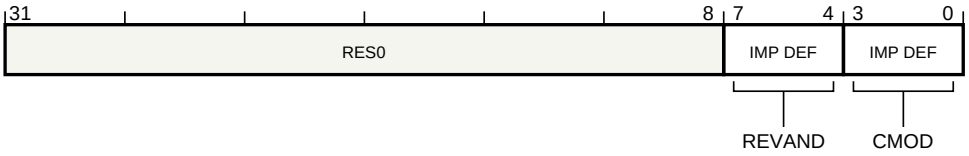


Figure 3.36: PMPIDR3 the component uses a 12-bit part number

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

REVAND, bits [7:4]
Component minor revision. PMPIDR2.REVISION and PMPIDR3.REVAND together form the revision number of the component, with PMPIDR2.REVISION being the most significant part and PMPIDR3.REVAND the least significant part. When a component is changed, PMPIDR2.REVISION or PMPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. If PMPIDR2.REVISION is increased then PMPIDR3.REVAND should be set to `0b0000`.

This field reads as an IMPLEMENTATION DEFINED value.

CMOD, bits [3:0]
Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

3.38.2 The component uses a 16-bit part number

Configurations
Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

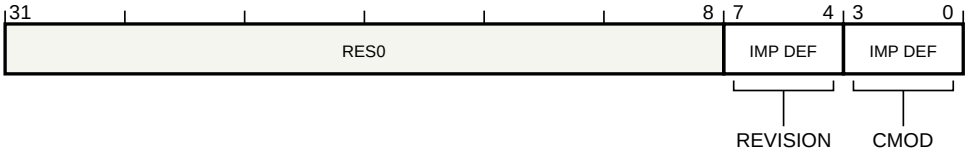


Figure 3.37: PMPIDR3 the component uses a 16-bit part number

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

REVISION, bits [7:4]
Component revision. When a component is changed, PMPIDR3.REVISION is increased to ensure that software can differentiate the different revisions of the component.

This field reads as an IMPLEMENTATION DEFINED value.

CMOD, bits [3:0]
Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

3.39 PMPIDR4, Peripheral Identification Register 4

The PMPIDR4 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR4 is present only if Peripheral Identification scheme is implemented. PMPIDR4 is RES0 otherwise.

Attributes
PMPIDR4 is a 32-bit read-only memory-mapped register located at offset `0xFD0`.

3.39.1 Field descriptions

The PMPIDR4 bit assignments are:

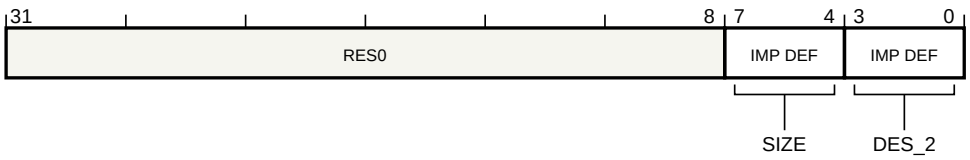


Figure 3.38: PMPIDR4

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

SIZE, bits [7:4]
Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of `0b0000` means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{PMPIDR4.SIZE}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field reads as an IMPLEMENTATION DEFINED value.

DES_2, bits [3:0]
Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

3.40 PMPIDR5, Peripheral Identification Register 5

The PMPIDR5 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR5 is present only if Peripheral Identification scheme is implemented. PMPIDR5 is RES0 otherwise.

Attributes
PMPIDR5 is a 32-bit read-only memory-mapped register located at offset `0xFD4`.

3.40.1 Field descriptions

The PMPIDR5 bit assignments are:

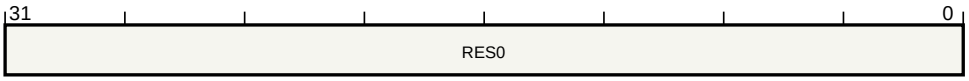


Figure 3.39: PMPIDR5

Bits [31:0]
This field is RES0.

3.41 PMPIDR6, Peripheral Identification Register 6

The PMPIDR6 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR6 is present only if Peripheral Identification scheme is implemented. PMPIDR6 is RES0 otherwise.

Attributes
PMPIDR6 is a 32-bit read-only memory-mapped register located at offset `0xFD8`.

3.41.1 Field descriptions

The PMPIDR6 bit assignments are:

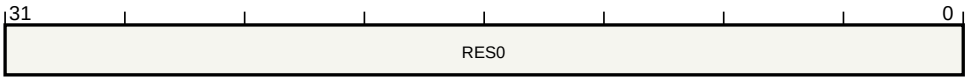


Figure 3.40: PMPIDR6

Bits [31:0]
This field is RES0.

3.42 PMPIDR7, Peripheral Identification Register 7

The PMPIDR7 characteristics are:

Purpose
Provides discovery information about the component.

Usage constraints
None.

Configurations
PMPIDR7 is present only if Peripheral Identification scheme is implemented. PMPIDR7 is RES0 otherwise.

Attributes
PMPIDR7 is a 32-bit read-only memory-mapped register located at offset `0xFDC`.

3.42.1 Field descriptions

The PMPIDR7 bit assignments are:

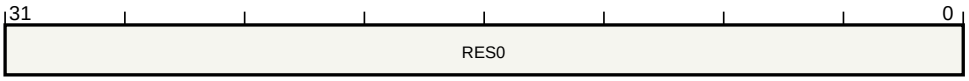


Figure 3.41: PMPIDR7

Bits [31:0]
This field is RES0.

3.43 PMSSCR, Snapshot Capture Register

The PMSSCR characteristics are:

Purpose
Provides a mechanism for software to initiate a sample.

Usage constraints
None.

Configurations
PMSSCR is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSCR is RES0 otherwise.

In some Arm PE implementations where the snapshot feature is an IMPLEMENTATION DEFINED extension, PMSSCR is located at offset `0x6F0`.

Attributes
PMSSCR is a 32-bit write-only memory-mapped register located at offset `0xE30` in the Page 0 component.

3.43.1 Field descriptions

The PMSSCR bit assignments are:

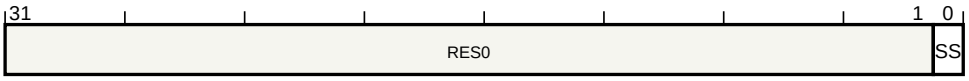


Figure 3.42: PMSSCR

Bits [31:1]
Reserved. This field is RES0.

SS, bit [0]
Capture now. The possible values for writing to this bit are:

`0`	Ignored.
`1`	Initiate a capture immediately.

3.44 PMSSRR, Snapshot Reset Register

The PMSSRR characteristics are:

Purpose
Configure Snapshot to reset monitors after each sample is taken.

Usage constraints
None.

Configurations
PMSSRR is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSRR is RES0 otherwise.

Support for the capability to reset counters after each sample is taken is optional. Arm recommends this feature is not implemented where the PMU might also be used in a non-snapshot mode, for example, in a PE.

If the capability is not implemented and the snapshot feature is implemented, this register is RAZ/WI.

In some Arm PE implementations where the snapshot feature is an IMPLEMENTATION DEFINED extension, PMSSRR is located at offset `0x6F4`.

Attributes
PMSSRR is a 64-bit read/write memory-mapped register located at offset `0xE38` in the Page 0 component.

3.44.1 Field descriptions

The PMSSRR bit assignments are:

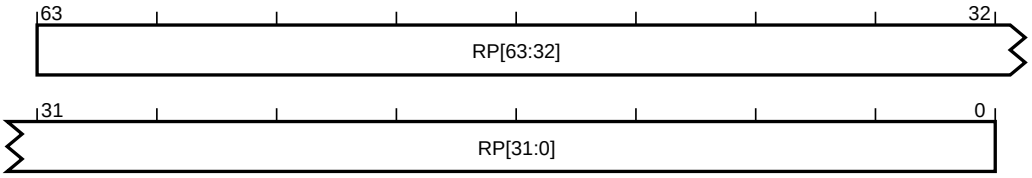


Figure 3.43: PMSSRR

RP[*m*], bit [*m*], for *m* = 0 to 63
Reset monitor. If *m* >= PMCFGR.N, the number of implemented monitors, then RP[*m*] is RAZ/WI. Otherwise, indicates whether PMEVCNTR<m> and PMOVSR[*m*] are to be reset after a capture. The possible values of this bit are:

`0`	Do not reset PMEVCNTR<m> and PMOVSR[*m*] on capture.
`1`	Reset PMEVCNTR<m> and PMOVSR[*m*] on capture.

If the Cycle Counter extension is implemented, then PMSSRR.RP[31] controls reset of the cycle counter.

3.45 PMSSSR, Snapshot Status Register

The PMSSSR characteristics are:

Purpose
Holds status information about the captured monitors.

Usage constraints
The location of PMSSSR within the PMSVR<n> registers is IMPLEMENTATION DEFINED. Arm recommends that PMSSSR is read after the other PMSVR<n> registers, meaning it might be placed last in the PMSVR<n> registers.

Configurations
PMSSSR is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSSSR is RES0 otherwise.

PMSSSR is a required one of the PMSVR<n> registers.

Attributes
PMSSSR is a 32-bit read-only memory-mapped register located at offset `0x600` + IMPLEMENTATION DEFINED in the Page 1 component.

3.45.1 Field descriptions

The PMSSSR bit assignments are:

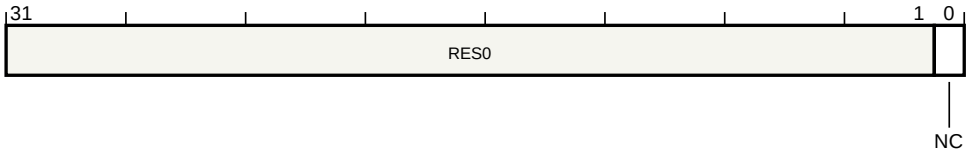


Figure 3.44: PMSSSR

Bits [31:1]
Reserved. This field is RES0.

NC, bit [0]
No capture. Indicates whether the PMU monitors have been captured. The defined values of this bit are:

`0`	PMU monitors captured.
`1`	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.

PMSSR.NC is reset to `0b1` by reset, but is overwritten at the first capture. Tools need to be aware that capturing over reset or power-down might lose data, as they are reliant on software saving and restoring the PMU state, including PMSSCR. There is no sampled sticky reset bit.

This bit resets to one.

3.46 PMSVR<*n*>, Saved Value Register <0-63>

The PMSVR<0-63> 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>.
- PMCCNTR.
- The overflow status flags, captured in PMOVSSR<n>, if PMSSRR is implemented.
- Any additional IMPLEMENTATION DEFINED syndrome information captured by the PMU.
- PMSSSR.

Once captured, the values in these registers are unaffected by direct or indirect writes to PMCR or any of the captured registers.

Usage constraints

None.

Configurations

PMSVR<*n*> is present only if PMCFGR.SS == `0b1`, snapshot extension is implemented. PMSVR<*n*> is RES0 otherwise.

Attributes

PMSVR<*n*> is a 32-bit read-only memory-mapped register located at offset `0x600` + 4×*n* in the Page 1 component.

A pair of PMSVR<*n*> registers at a doubleword-aligned offset might be defined as a single 64 bit register.

Glossary

Event-based sampling

The location in the program, or some other measurement, is recorded when a *sampled* event occurs. This builds up a statistical model of where events occur.

FDO

Feedback-directed Optimization

Feedback-directed Optimization

See *Profile-guided Optimization*.

Hardware Performance Monitor

A hardware resource that helps an engineer measure and profile software.

HPC

High-performance Computing

HPM

Hardware Performance Monitors

Little's Law

Little's Law ($L = \lambda W$) relates Arrival rate (λ) and Average occupancy (L) with average Response time (W).

Memory System Resource Partitioning and Monitoring

See [2].

MPAM

Memory System Resource Partitioning and Monitoring

PGO

Profile-guided Optimization

PMU

Performance Monitoring Unit

PPI

Private Peripheral Interrupt

Profile-guided Optimization

Profile-guided optimization, also known 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 HPM 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.