System Monitoring Control Framework Architecture Specification

v1.0

Document number	DEN0108
Document quality	BET
Document version	00bet0
Document confidentiality	Non-confidential

System Monitoring Control Framework Architecture Specification

Release information

Date	Version	Changes
2021/Jul/12	00bet0	• First Beta Release.
2020/Sep/25	00alp0	• First Alpha Release.

Arm Non-Confidential Document Licence ("Licence")

This Licence is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PETMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

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

LES-PRE-21585 version 4.0

Contents

System Monitoring Control Framework Architecture Specification

Preface

This preface introduces the System Monitoring Control Framework Architecture Specification. It contains the following sections:

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 for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

Numbers

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

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Rules-based writing

This specification consists of a set of individual rules. Each rule is clearly identified by the letter R.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. Rationale statements are identified by the letter X.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements are clearly identified by the letter I.

Implementation notes are identified by the letter U.

Software usage descriptions are identified by the letter S.

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

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

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 statement.

R_{X001} This is a rule statement.

I This is an information statement.

X This is a rationale statement.

U This is an implementation note.

S This is a software usage description.

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] *AMBA Low Power Interface Specification: Arm Q-Channel and P-Channel Interfaces*. (ARM IHI 0068 C) Arm Ltd.

[2] *Arm Power Control System Architecture*. (ARM DEN 0050 C) Arm Ltd.

[3] *ARM® AMBA® 5 AHB Protocol Specification*. (ARM IHI 0033 B) Arm Ltd.

[4] *AMBA® AXI and ACE Protocol Specification*. (ARM IHI 0022 H) Arm Ltd.

[5] *ARM® System Control and Management Interface*. (DEN0056 C) Arm Ltd.

[6] *AMBA® 5 CHI Architecture Specification*. (ARM IHI 0050 E) Arm Ltd.

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 (System Monitoring Control Framework Architecture Specification).
- The number (DEN0108 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
Introduction

This section introduces the System Monitoring Control Framework (SMCF).

1.1 Background

Modern System on Chip (SoC) designs contain an increasing number of on-chip sensors and monitors. These are commonly used for collecting data about the SoC for use in its management.

For example, temperature measurements taken at various points in the SoC are collected and used in the power and thermal management of the system.

This increases the requirement for software to be aware of many different monitor contexts and manage the collection of an increasing amount of monitor data.

1.2 Introduction

The System Monitoring Control Framework is designed to manage a large and diverse set of on-chip sensors and monitors. It does this by presenting software with a standard interface to control the monitors, regardless of type, and reducing software load of controlling the monitor sampling and data collection.

The SMCF reduces the burden on monitor control by enabling sampling on multiple monitors to be controlled together and by various triggers either internal or external to the SMCF. The number of monitors that the SMCF supports can be configured.

The SMCF eases data collection requirements by allowing the data from multiple monitors to be collated in a single location or writing out data to a memory-mapped location that is easier for the monitoring agent to access.

The SMCF can also reduce the requirement on the monitoring agent to constantly monitor data by providing programmable alerts that can inform the monitoring agent when certain changes happen, or thresholds are crossed.

The monitoring agent is only required to perform an initial setup and then process data when required on a constant basis or when an alert informs it that analysis or action is required.

The structure of the SMCF is one of distributed monitor groups located around the system where monitoring is required, reusing existing interconnect infrastructure where applicable to communicate.

Figure 1.1 shows an example of the SMCF structure.

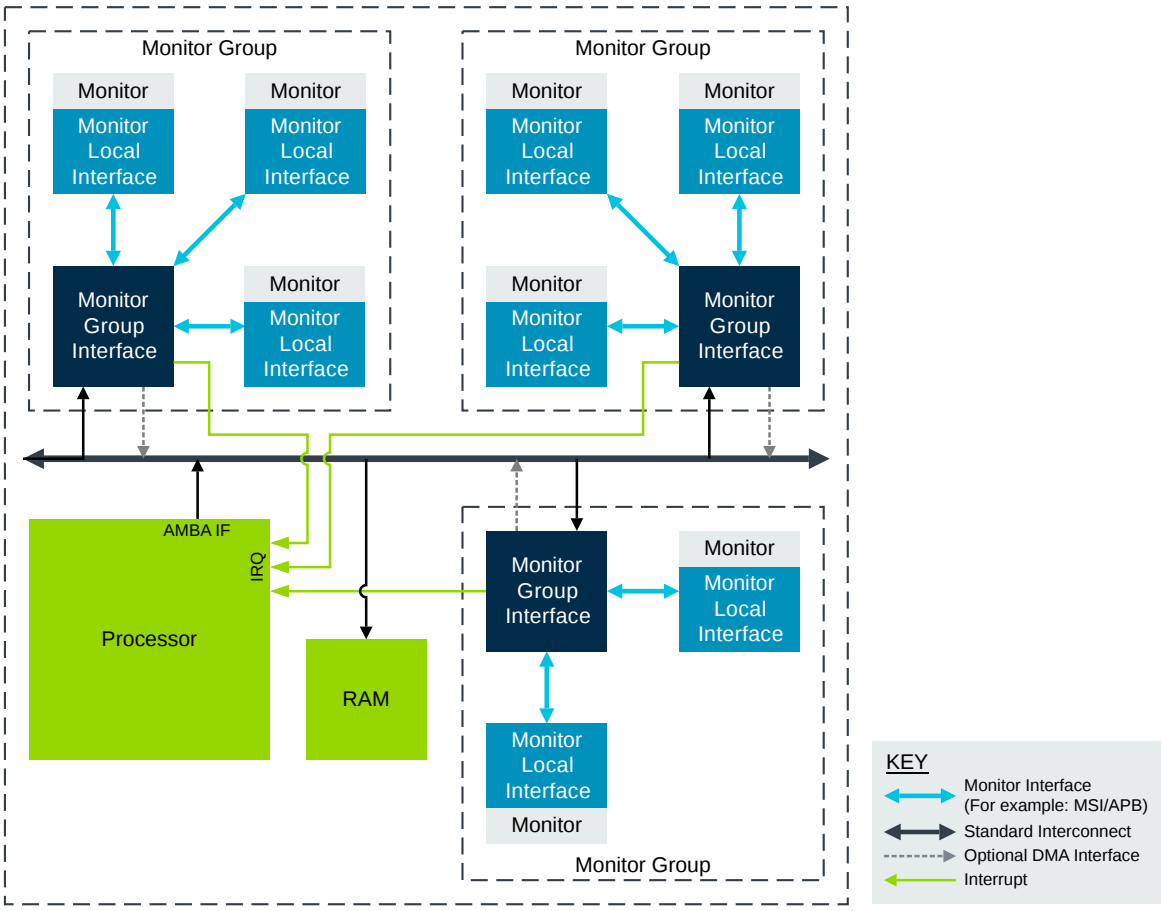


Figure 1.1: System Monitor Control Framework overview

Power management is an essential feature in modern SoCs. The SMCF structure, along with the ability to write out monitor data to a memory-mapped location, is beneficial when areas of the SoC are power managed. Because the SMCF writes data to a memory-mapped location, there is no requirement for the power management to centrally control or coordinate which monitors data can be collected related to their availability. Additionally, features of the SMCF allow identification of which groups of monitors are power managed.

1.2.1 Features

The SMCF includes the following features:

- Standard software interface
- Support for 1 to 32 sensors in each Monitor Group Interface (MGI)
- Support for up to 4096 MGIs
- Support for continuous periodic monitor sampling through various triggers
- Optional alert interrupts on programmed conditions
- Proprietary monitor adaptation through Monitor Local Interface (MLI)
- Definition for optional Serial Monitor Interface
- Optional DMA interface to send monitor data to a programmable memory-mapped location

The SMCF consists of two components, an MGI and an MLI. An MGI is a configurable generic component that provides a standard software interface and common functions to multiple monitors. An MGI communicates with one or more MLIs depending on the number and type of monitors it supports. An MLI is typically associated with a single sensor or monitor and provides local adaptation to a particular type and implementation of that sensor or monitor.

An MGI and an MLI communicate using a set of standard commands. For more information on these commands see Chapter 3 *Commands*. These commands define the interface between an MGI and an MLI. This means that either component can be designed separately and communicate together, given the same physical interface.

1.2.1.1 Monitor Group Interface

The Monitor Group Interface (MGI) provides a software interface and a command interface to a group of monitors. An MGI converts required operations into commands that are sent to its MLIs. These operations can be generated either by software programming or internal actions. Each MLI converts these commands into operations for the sensor or monitor. An MGI receives commands from an MLI that are sent as a result of operations performed on and by the sensor or monitor and converts them to the appropriate operations and register updates.

An MGI is intended to be designed as a configurable and reusable component that is not specific to any sensor or monitor type or technology. It can therefore be reused for multiple monitor types across different SoC implementations and technology developments.

An MGI supports the reuse of existing interconnect to receive communications from the controlling agent, while enabling low resource routing between an MGI and its MLIs. SoCs can implement multiple MGIs for the same or different types of monitors to allow, for example, for functional separation or physical placement that maximizes the reuse of existing interconnect.

For example, an SoC can have multiple CPU cores, and each core can have multiple temperature monitors. Each core's monitors could be accessed through a local MGI. The monitors in a core are accessed from its MGI by a serial interface. This causes less disruption in the core routing where existing interconnect does not reach. Each separate core MGI is accessed by the existing interconnect to ease SoC-level congestion. This is just an example and the exact requirements of SoC topology can differ.

More use case examples are illustrated in Chapter A2 *Use models*.

1.2.1.2 Monitor Local Interface

I A Monitor Local Interface (MLI) takes the commands sent from its MGI and converts them to the required sensor or monitor actions to perform the required function. It also takes sensor or monitor outputs and converts them to commands to send to its MGI.

The use of standard commands allows maximum reuse of the MLI structure, with only the conversion of commands to monitor specific actions required to be modified across different monitors and technology.

1.2.1.3 Physical interfaces between monitor group and local interfaces

To allow for accommodation of various monitor types and topologies, the physical interface between an MGI and an MLI is not constrained by this specification. Regardless of the physical interface, these components use the SMCF-defined commands to communicate.

There can be one or many interfaces between an MGI and its MLIs. An implementation might choose to have separate serial interfaces to address physically separate monitors. This could reduce the requirement for additional interconnect logic and reduce the routing effort for those interfaces.

Implementations should also consider the issue of reuse and compatibility when choosing a physical interface for the components. Therefore, this document includes a specification for a recommended Serial Monitor Interface as a basis for a common compatible and low resource connection between MGIs and MLIs.

Chapter 2
Monitor Group Interface functional description

This section describes the functionality of the Monitor Group Interface (MGI).

2.1 Introduction

An MGI provides a software interface and a set of common functions to a group of monitors.

An MGI initiates operations in the monitors by sending commands to them through the Monitor Local Interface (MLI). These operations can be initiated directly by software or by internal MGI functions. Example operations are monitor configuration or instructions to start a sample.

An MGI receives commands from monitors through its MLIs and converts them to the appropriate actions and register updates. Example commands are configuration confirmations and the return of monitor data.

An MGI contains the programmer's interface, interrupts, and the interface to its MLIs. It can also optionally include other functions like a periodic timer, alerts, inputs and output triggers, tag input, and the DMA interface.

Figure 2.1 shows the external interfaces on an MGI. The interfaces shown in gray are optional.

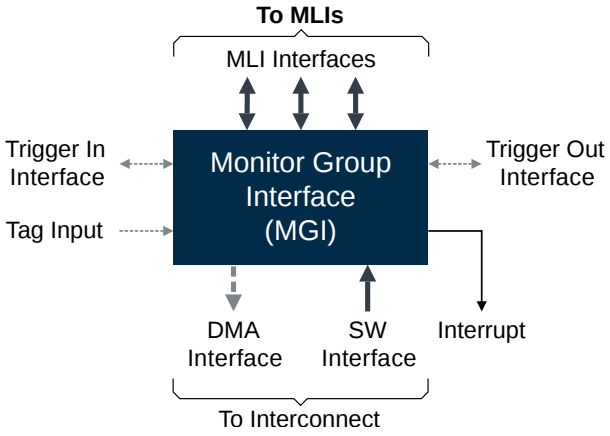


Figure 2.1: MGI interfaces

An MGI is used to perform the functions described in the following sections.

2.2 Enabling and disabling monitors

I Each monitor supported by an MGI can be enabled and disabled.

R An enabled monitor must be ready to start a sample when a Start Sample command is received.

I The exact state of a disabled monitor is IMPLEMENTATION DEFINED. However, it typically allows the monitor to enter a lower power state.

A monitor might also be disabled to prevent it from sampling when its data is not required. This can reduce dynamic power during sampling. For example, a group with eight monitors might not require all monitors to be sampled continuously, so only the required monitors are enabled.

I Monitor enabling and disabling is performed using the following registers:

- Monitor Enable Request Register (MGI_MON_REQ)
- Monitor Enable Status Register (MGI_MON_STAT)

There is a bit inside each of these registers for each monitor.

Table 2.1 shows the status of the monitor based on these register values.

Table 2.1: Monitor enable status

MGI_MON_REQ[x]	MGI_MON_STAT[x]	Monitor x Status
0b0	0b0	Disabled
0b1	0b0	Enabling
0b1	0b1	Enabled
0b0	0b1	Disabling

S To enable a monitor, set the relevant MGI_MON_REQ bit to 0b1. For example, bit 0 represents monitor 0, bit 4 represents monitor 4.

The enabled or disabled state of a monitor can be read from MGI_MON_STAT.

If the relevant monitor bits in MGI_MON_REQ and MGI_MON_STAT are different, the monitor is in transition between enabled and disabled states.

Software should ensure that MGI_MON_REQ and MGI_MON_STAT are in the same state before performing further updates to MGI_MON_REQ. This prevents race conditions when reading the status register.

R If a monitor is disconnected when it is requested to be enabled or disabled, MGI_MON_STAT[x] is updated immediately with the value written to MGI_MON_REQ[x]. For more information on monitor connection and disconnection, see section 2.9 *MLI connection and disconnection*.

X This immediate update of the enable status when a monitor is disconnected allows software to control monitor enabling regardless of the monitor connection status. The latest enable status is sent to the monitor when it starts a connection sequence. Therefore, it is always in the correct state when connected. For more information on monitor connection and disconnection see section 2.9 *MLI connection and disconnection*.

For details of the commands relating to enabling and disabling monitors see:

- 3.2.1 *Enable Monitor command*
- 3.2.2 *Disable Monitor command*
- 3.2.8 *Monitor Enabled command*
- 3.2.9 *Monitor Disabled command*

2.3 Monitor modes

I Monitor modes are used to configure the monitor for the required operation. This allows IMPLEMENTATION DEFINED monitor configuration to be performed using standard MGI registers. The type of configuration required depends on the sensor or monitor being supported. Multiple monitors can be programmed simultaneously, using the mode broadcast register to control which monitors are sent the command. This allows for single or different types of monitors to be combined within the same MGI.

Monitor modes are programmed using the following registers:

- Monitor Mode Broadcast Register (MGI_MODE_BCAST)
- Monitor Mode Request Register <n> (MGI_MODE_REQ<n>)
- Monitor Mode Status Register <n> (MGI_MODE_STAT<n>)

There can be multiple mode request and status registers, depending on the amount of configuration required. Each set operates as a pair. For instance, MGI_MODE_REQ0 and MGI_MODE_STAT0 are a pair, and MGI_MODE_REQ1 and MGI_MODE_STAT1 are a pair.

I When a MGI_MODE_REQ<n> is written, a Set Monitor Mode command is sent to all enabled monitors that have their broadcast enable bit set in MGI_MODE_BCAST.

S Software should ensure the MGI_MODE_REQ<n> and MGI_MODE_STAT<n> pair are equal before further updating MGI_MODE_REQ<n>. If the registers are not equal when MGI_MODE_REQ<n> is written, then the behavior is unpredictable.

S The mapping of MGI_MODE_REQ<n> values to monitor functions is IMPLEMENTATION DEFINED.

S The monitor mode values supported might be a subset of the possible values programmable in MGI_MODE_REQ<n>. Programming values not supported by the monitor might result in a Monitor Mode Error. This is reported in the Error Code Register (MGI_ERR_CODE) and can generate an interrupt.

For details of the commands relating to setting monitor modes see:

- 3.2.4 *Set Monitor Mode command*
- 3.2.11 *Monitor Mode Complete command*

2.4 Monitor sampling

I Monitor sampling is when the sensor or monitor is instructed to initiate a collection of data. The initiation of monitor sampling can be controlled in several ways. This section describes the ways that monitor sampling can be controlled.

Monitor sampling is controlled using the following registers:

- Monitor Sample Enable Register (MGI_SMP_EN)
- Monitor Sample Configuration Register (MGI_SMP_CFG)
- Monitor Sample Period Register (MGI_SMP_PER)
- Monitor Data Registers (MGI_DATA<n>)

R A monitor sample is ongoing when a sample trigger has occurred, but all monitors the sample was started on have not yet done one of the following:

- Returned their data with a Monitor Data command.
- Responded with a Monitor Sample Error command.
- Sent a Monitor Disconnect command.

R A monitor sample is complete when all monitors the sample was started on have done one of the following:

- Returned their data with a Monitor Data command.
- Responded with a Monitor Sample Error command.
- Sent a Monitor Disconnect command.

R When a monitor responds with a Monitor Sample Error or Monitor Disconnect command, the data for that monitor is set to zero.

2.4.1 Sample trigger types

I An MGI supports the following events to trigger the start of a sample:

- Manual start by software write, see section 2.4.1.1 *Manual trigger*
- Continuous sample based on programmed period, see section 2.4.1.2 *Periodic sample*
- Data read, see section 2.4.1.3 *Data read*
- Input trigger, see section 2.4.1.4 *Input trigger*

The sample trigger type is programmed in MGI_SMP_CFG.SMP_TYP.

2.4.1.1 Manual trigger

I Manual sampling is used when software wants to perform a single sampling of the monitor group.

S Software selects manual sampling by setting the sample type field, MGI_SMP_CFG.SMP_TYP, to 0b00.

2.4.1.2 Periodic sample

I Periodic sampling is used when regular periodic monitor data sampling is required.

I When this sampling type is enabled, an MGI starts a sample every software programmed period without software intervention.

S Software selects periodic sampling by setting the sample type field, MGI_SMP_CFG.SMP_TYP, to 0b01.

S Software programs the sample period in MGI_SMP_PER).

I The sample period is timed from when a sample starts. When the sample period time expires, another sample is started. This also starts the timing of another sample period.

Figure 2.2 shows the relationship between the monitor sample time and the sample period.

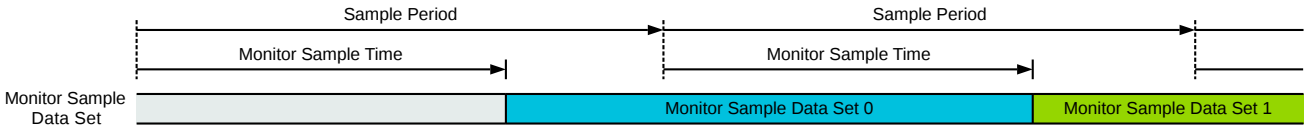


Figure 2.2: Monitor sample period

If the programmed sample period is less than the monitor sample time, the next sample starts immediately when the previous sample is completed.

Figure 2.3 shows the relationship between the monitor sample time and the sample period, when the sample period is less than the monitor sample time.

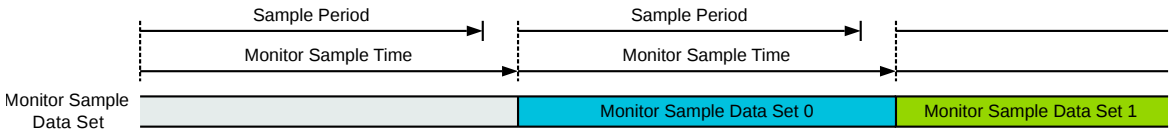


Figure 2.3: Monitor sample period shorter than sample time

In this scenario, an MGI generates a Sample Period Error.

R For periodic sampling, a sample start and sample period start is triggered when the sample enable bit, MGI_SMP_EN.EN, is 0b1 and either:

- The ongoing bit, MGI_SMP_EN.OG, is 0b0. This begins the initial sample and period count.
- A count to the programmed period from a previous sample start is reached.

2.4.1.3 Data read

I Data read sampling is used when a sample is required to be started when the data from the previous monitor sample data set is consumed. When the last data value from a monitor sample data set is read, a new sample begins.

S Software selects data read sampling by setting the sample type field, MGI_SMP_CFG.SMP_TYP, to 0b10.

R Accesses to MGI_DATA<n> while a monitor sample is ongoing do not trigger a monitor sample start.

R For data read sampling, a sample start is triggered in two scenarios.

Scenario 1, when sampling is first enabled, this occurs when all the following are true:

- The sample enable bit, MGI_SMP_EN.EN, is 0b1.
- The ongoing bit, MGI_SMP_EN.OG, is 0b0.

Scenario 2, when data from a previous sample is read, this occurs when all the following are true:

- The sample enable bit, MGI_SMP_EN.EN, is 0b1.
- The ongoing bit, MGI_SMP_EN.OG, is 0b1.
- There is no ongoing monitor sample.
- There is a read access to the highest number MGI_DATA<n>.

I A sample start is not triggered if sample data is read from an alternate address or external memory-mapped location.

2.4.1.4 Input trigger

I Input trigger sampling is used when a sample is required to be started from an event that is external to an MGI. The input trigger is an optional MGI interface that allows an external event to cause operations in an MGI. For more information on input triggers, see section 2.6 *Input and output triggers*.

S Software selects input trigger sampling by setting the sample type field, MGI_SMP_CFG.SMP_TYP, to 0b11.

R Input trigger events that occur while a monitor sample is ongoing do not trigger a monitor sample start. They are ignored for the purposes of triggering a monitor sample start.

R For input trigger sampling, a sample start is triggered when all the following are true:

- The sample enable bit, MGI_SMP_EN.EN, is 0b1.
- There is not an ongoing monitor sample.
- A trigger event is received on the input trigger interface.

X This use model allows a sample to be triggered from an agent that is external to an MGI, that might also be common to multiple monitor groups. For example, the programming of a single register to send an input trigger to multiple MGI, a system timer, or another system trigger, including from another MGI. These examples are not a complete list of the possible uses of this mode.

2.4.2 Enabling monitor sampling

I Monitor sampling is enabled and disabled using MGI_SMP_EN.

The behavior of this register varies between the manual sample type and other sample types.

2.4.2.1 Enabling with manual sample type

I Manual sampling is used to enable a single monitor sample based on register writes. The sample enable bit is used to start the sample. The ongoing bit is used to indicate that the sample is ongoing.

S Software selects manual sampling by setting the sample type field, MGI_SMP_CFG.SMP_TYP, to 0b00.

S Software enables a single sample of all enabled and connected monitors by setting the sample enable bit MGI_SMP_EN.EN to 0b1.

S If the sample enable bit, MGI_SMP_EN.EN, is set to 0b1 while a sample is ongoing, it remains set to 0b1. Another sample starts if the enable is still set to 0b1 when the sample completes.

R For manual sampling, a sample start is triggered when the sample enable bit, MGI_SMP_EN.EN, is 0b1 and the ongoing bit, MGI_SMP_EN.OG, is 0b0.

R For manual sampling, the sample enable bit, MGI_SMP_EN.EN, is cleared to 0b0 by an MGI when the sample ongoing bit, MGI_SMP_EN.OG, is 0b0.

R For manual sampling, the sample ongoing bit, MGI_SMP_EN.OG is:

- Set to 0b1 by an MGI when it is 0b0 and the sample enable bit, MGI_SMP_EN.EN, is 0b1.
- Cleared to 0b0 by an MGI when a monitor sample completes.

I Figure 2.4 shows a timing diagram of a single manual sample.

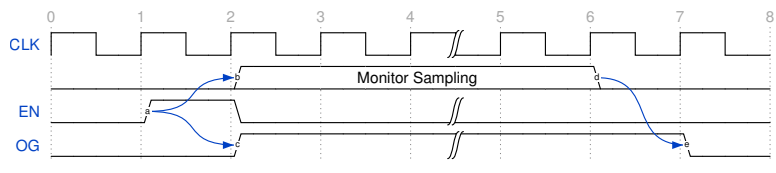


Figure 2.4: Monitor enable for manual sampling

Figure 2.5 shows a timing diagram for manual sampling when the enable is set and cleared while a sample is ongoing.

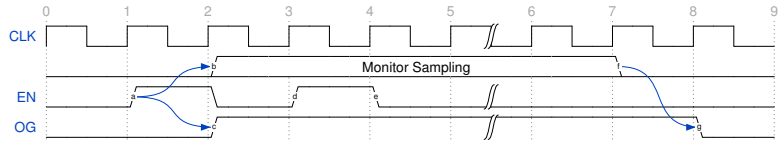


Figure 2.5: Monitor enable for manual sampling when the enable is set and cleared during a sample.

Figure 2.6 shows a timing diagram for manual sampling when the enable is set while the sample is ongoing.

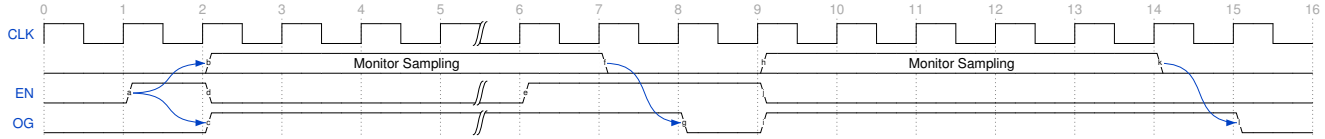


Figure 2.6: Monitor enable for manual sampling when the enable is set during a sample.

2.4.2.2 Enabling with other sample types.

I Other sample types use triggers other than the sample enable to start a sample. In this case, the sample enable bit is used to enable the selected trigger to start a sample. The ongoing bit is used to indicate when a sample start can be triggered or when there is an ongoing sample. It is only guaranteed that an MGI is not able to start a sample, and for there not to be a sample ongoing, when both the enable and the ongoing bits are 0b0.

I If the sample type is set as periodic, data read, or trigger input in MGI_SMP_EN.SMP_TYP, then when the sample enable, MGI_SMP_EN.EN, is set to 0b1 by software, it remains set to 0b1 until cleared to 0b0 by software.

In this case, when a sample starts is dependent on the sample type, a new sample starts when:

- MGI_SMP_EN.EN = 0b1.
- A sample is not currently ongoing.
- The sample trigger for the configured sample type occurs.

R For non-manual sampling, the sample ongoing bit, MGI_SMP_EN.OG, is set to 0b1 by an MGI when the sample enable bit, MGI_SMP_EN.EN, is 0b1.

R For non-manual sampling, the sample ongoing bit, MGI_SMP_EN.OG, is cleared to 0b0 by an MGI when MGI_SMP_EN.EN is 0b0 and there is not an ongoing monitor sample.

I Figure 2.7 shows an example timing diagram for a periodic sample enable.

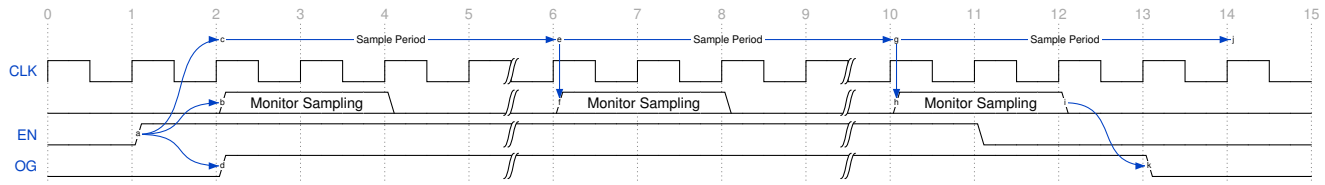


Figure 2.7: Monitor enable for a periodic sample

2.4.3 Sample delay

I The start of a sample can be delayed by a programmed number of cycles from when the trigger event occurs. This delay takes effect regardless of the sample trigger.

Figure 2.8 shows the relationship between the sample trigger and the sample starting.

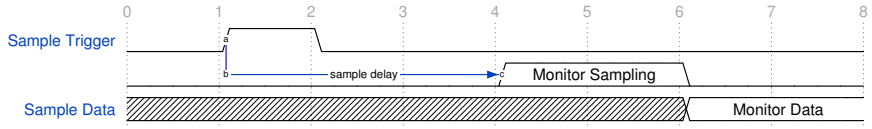


Figure 2.8: Monitor sample delay

S Software programs the sample delay length in MGI_SMP_DLY.

X An example use-case is when the sample time is relatively short in comparison to the periodicity that monitor data is required, for instance a monitor with a sample time of 60us sampled every 4ms. Without the sample delay, if an agent triggers a sample start based on its processing of a previous sample data set the sample is taken immediately. Then when it comes to process this next data set it might not be a recent enough sample, almost 4ms in this example. The sample delay allows the sampling to be shifted so the monitor data is more recent without placing more load on SW to time the start of every sample.

Figure 2.9 and Figure 2.10 show examples when using and not using a sample delay.

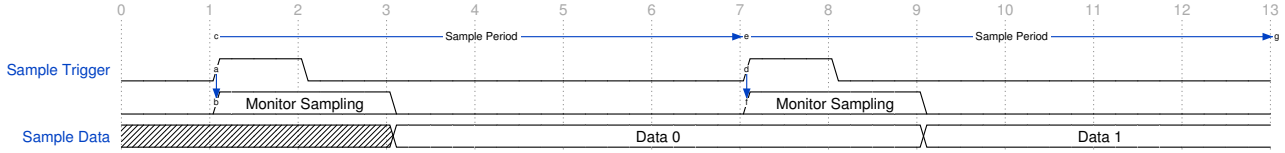


Figure 2.9: Example - Monitor data read trigger without a sample delay

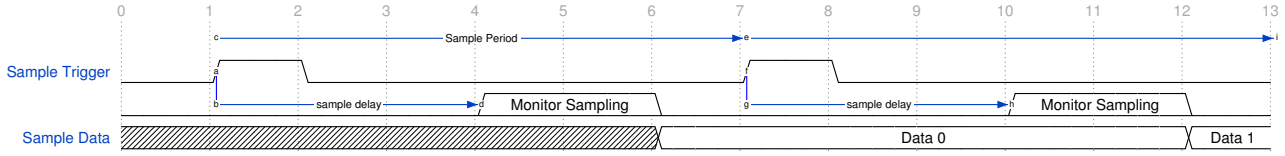


Figure 2.10: Example - Monitor data read trigger with a sample delay

2.5 Alerts

I Alerts are an optional feature used to indicate when monitor sample data values change in certain ways that can be specified by software. They can set interrupts based on programmed alert conditions.

Monitor alerts are controlled using the following registers:

- Alert <n> Type Register (MGI_ATYP<n>)
- Alert <n> Value Register (MGI_AVAL<n>)

Each alert can monitor a single condition at a set level and can be separately configured for condition type and threshold. These values are programmed in the MGI_ATYP<n> and the MGI_AVAL<n>, where n is the alert number.

The alert trigger events are:

- Upper threshold
- Lower threshold
- Rising delta (optional)
- Falling delta (optional)

R An MGI can support from 0 to 7 alerts.

S The number of alerts supported can be determined by reading MGI_FEAT0.ALERT_NUM.

S The rising and falling delta conditions are optional. Whether they are supported can be determined by reading MGI_FEAT0.ALT_DELTA.

2.5.1 Upper threshold alert condition

I The upper threshold alert triggers when a data value from any monitor is greater than the value that is programmed in MGI_AVAL<n>.

R The upper threshold alert triggers when a Monitor Data command is received from an MLI with a value greater than the value programmed in MGI_AVAL<n>.

2.5.2 Lower threshold alert condition

I The lower threshold alert triggers when a data value from any monitor is less than the value programmed in MGI_AVAL<n>.

R The lower threshold alert triggers when a Monitor Data command is received from an MLI with a value less than the value programmed in the MGI_AVAL<n>.

2.5.3 Rising delta alert condition

I The rising delta alert triggers when a data value from a monitor is greater than the previous data value from that monitor by the value programmed in MGI_AVAL<n>. It can be used to indicate that the rate of increase of the monitor value has exceeded a limit that is of interest. For example, for a temperature monitor the value might not have exceeded a specific threshold but is rising at a rate to cause concern. This might mean that action or more constant monitoring is required.

R The rising delta alert triggers when a Monitor Data command is received from an MLI with a data value that is greater, by the value programmed in MGI_AVAL<n>, than the value from the same monitor in the previous monitor sample data set.

2.5.4 Falling delta alert condition

I The falling delta alert triggers when a data value from a monitor is less than the previous data value from that monitor by the value programmed in MGI_AVAL<n>. It can be used to indicate that the rate of decrease of the monitor value has exceeded a limit that is of interest.

R The falling delta alert triggers when a Monitor Data command is received from an MLI with a data value that is less, by the value programmed in MGI_AVAL<n>, than the value from the same monitor in the previous monitor sample data set.

2.6 Input and output triggers

I The input and output trigger interfaces can be used to interface with external hardware. The input trigger allows external hardware to trigger events within an MGI. The output trigger allows an MGI to trigger external hardware based on internal events.

2.6.1 Input trigger interface

I The input trigger interface can be used by external hardware to trigger events within an MGI.

The events that can be caused by the input trigger interface are:

- Sample Start
- Output Trigger Event

For more information on using the input trigger to start monitor sampling, see 2.4.1.4 *Input trigger*.

2.6.2 Output trigger interface

I The output trigger interface can be used to signal to external hardware that an event has occurred in an MGI.

Each event that can cause an output trigger can be masked separately in the Output Trigger Mask Register (MGI_TRG_MASK). By default, all output trigger events are masked.

The events that can cause an output trigger are:

- Monitor Sample Data Set Complete
- Monitor Enable Request Complete
- Monitor Mode Request Complete
- User-Defined Command Received
- Error
- Monitor Trigger
- Input Trigger
- Configuration Request
- Data Write Complete
- Alert

The definition for these events is the same as those for MGI interrupts. For further information see section 2.10 *Monitor Group Interface interrupt events*.

2.7 Tag input

I The tag input is an optional interface that can be used to interface with external hardware. The tag input allows an externally specified tag to be provided with the sample data. This is provided:

- Internally in the:
  - Tag Start Registers (MGI_TAG<n>_START).
  - Tag End Registers (MGI_TAG<n>_END).
- Optionally in a sample identifier when the monitor sample data set is written by the Direct Memory Access (DMA) interface to a memory-mapped location.

For example, this input could be connected to a timestamp value to allow this information to be available with all monitor sample data sets.

The tag input is sampled when the first data from a monitor sample is received from any monitor.

S Support for this feature can be determined by reading MGI_FEAT0.TAG_IN.

2.8 Direct Memory Access interface

I The Direct Memory Access (DMA) interface is an optional feature that is used to write monitor data to a memory-mapped location when a sample is complete. Having the data written to a convenient location adds efficiency, because an agent using the monitor data does not have to make peripheral accesses to collect the data. It also allows monitor data from multiple MGI instances to be collated where applicable.

The DMA interface is controlled using the following registers:

- Data Write Enable Register (MGI_WREN)
- Data Write Configuration Register (MGI_WRCFG)
- Data Write Address Register 0 (MGI_WADDR0)
- Data Write Address Register 1 (MGI_WADDR1)

U The protocol and size of the DMA interface is IMPLEMENTATION DEFINED.

S Support for this feature can be determined by reading MGI_FEAT0.DMA_IF.

2.8.1 Data write configuring and enabling

S The base memory-mapped address that the monitor data is written to can be programmed in MGI_WADDR0 and MGI_WADDR1.

S The writing out of monitor data is enabled by setting MGI_WREN.WREN to 0b1.

I The monitor data write can be configured to occur:

- Every time a monitor sample data set completes
- When a monitor sample data set completing causes an alert to be triggered

If an MGI is configured to write data on an alert, this data write behavior can be configured to:

- Write only the monitor sample data set that caused the alert interrupt event to occur.
  - This only occurs when a first alert condition is triggered. It only writes another monitor sample data set when all alert conditions are cleared, and a further alert condition event occurs.
- Start continuous data writing of every monitor sample data set starting with the monitor sample data set that caused the alert interrupt event to occur.
  - This only stops when data writing is disabled by software programming MGI_WREN.WREN.
  - If data writing is subsequently re-enabled, it returns to normal operation. It only starts to write data again when all alert conditions are cleared, and a further alert condition event occurs.

Figure 2.11 shows example write behavior when an MGI is programmed to write only when an alert condition is generated.

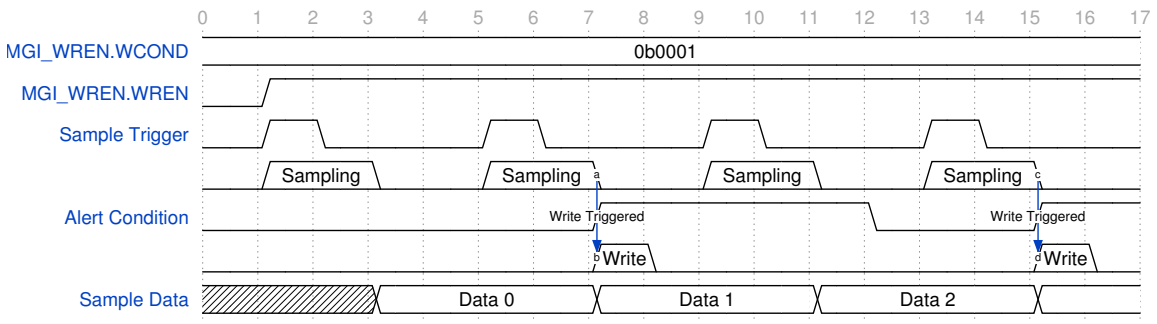


Figure 2.11: MGI write behavior when programmed to write only on an alert condition

When the monitor sample data set completes at t3, there is no alert condition, so a write does not occur. At t7 the sample completion generates an alert, so a write occurs. At t11, because the alert condition is still active no write occurs. This is the case even if the sample data would have caused an alert condition if one had not already been present. At t15, the alert condition has been cleared and another alert condition is triggered, so a write occurs.

Figure 2.12 shows example write behavior when an MGI is programmed to write continuously when an alert condition is generated.

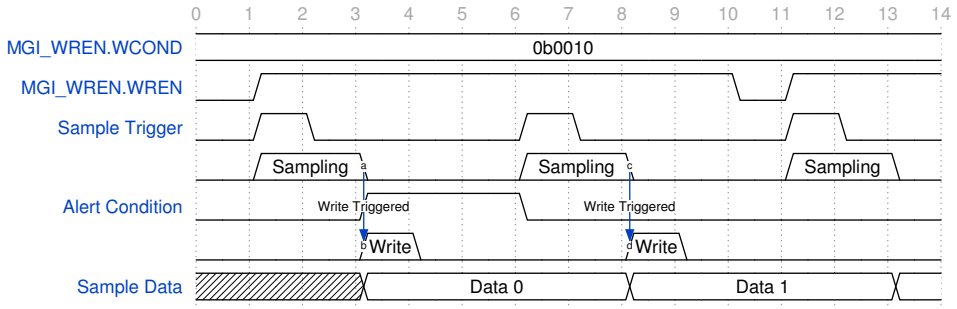


Figure 2.12: MGI write behavior when programmed to write continuously on an alert condition

When the monitor sample data set completes at t3 an alert condition is generated so a write occurs. At t8, because continuous writing is enabled, a write occurs, even though the alert condition has been cleared. At t10, the WREN is cleared to 0b0, stopping further writing of sample data. No further data is written until data writing is enabled again and another alert condition is generated. Therefore, when the monitor sample data set completes at t13, even though the WREN has been set to 0b1 at t11, no write occurs. This is because no alert condition has been generated.

R A data write occurs when MGI_WREN.WREN is 0b1, and one of the following occurs:

- MGI_WRCFG.WR_COND is 0b000, and a monitor sample data set completes.
- MGI_WRCFG.WR_COND is 0b001, and all the following are true:
  - A monitor sample data set completes.
  - An alert condition becomes active (the MGI_IRQ_STAT.ALT_IRQ_STAT field is non-zero) because of the monitor sample data set completing when no previous alert condition was active (the MGI_IRQ_STAT.ALT_IRQ_STAT field was zero).
- MGI_WRCFG.WR_COND is 0b010, and all the following are true:
  - A monitor sample data set completes.
  - Either:
    - An alert condition becomes active (the MGI_IRQ_STAT ALT_IRQ_STAT field is non-zero) because of the monitor sample data set completing when no previous alert condition was active, the MGI_IRQ_STAT ALT_IRQ_STAT field was previously zero.
    - The preceding condition has occurred since the write enable MGI_WREN.WREN last changed from 0b0 to 0b1.

2.8.2 Memory-mapped location size requirement

I The amount of space required to hold the monitor sample data set is dependent on:

- The number of monitors.
  - MGI_GRP_ID.MON_NUM.
- The number of data values generated by each monitor.
  - MGI_DATA_INFO.DATA_PER_MON.
- The width of the monitor data.
  - MGI_DATA_INFO.MON_DATA_WIDTH.
- If the monitor data is packed.
  - MGI_DATA_INFO.PACKED.
- The number and type of sample identifiers written. For more information see section 4.2 *Sample identifiers*.
  - MGI_WRCFG.INCR_ID_EN.
  - MGI_WRCFG.TAG_ID_EN.
  - MGI_WRCFG.NUM_SAMPLE_ID.
  - MGI_FEAT0.TAG_LEN.
- If the Monitor Group ID is written.
  - MGI_WRCFG.GRP_ID_EN.
- If the Data Valid bits are written.
  - MGI_WRCFG.DATA_VLD_EN.

It is calculated as follows:

Calculate the number of 32-bit words required for the sample ID and other meta-data (SAMPLE_ID_WORDS):

```
SAMPLE_ID_WORDS = (NUM_SAMPLE_ID * SMPID_32_WORDS) + GRP_ID_EN + DATA_VLD_EN
```

Calculate the number of 32-bit words required for the data (DATA_WORDS):

```
DATA_WORDS = ((MON_NUM + 1) * (DATA_PER_MON + 1)) / (DATA_PER_32)
```

Then the total is:

```
TOTAL_WORDS = SAMPLE_ID_WORDS + DATA_WORDS
```

SMPID_32_WORDS is the number of 32-bit words needed for a Sample ID and is calculated as follows:

```
if INCR_ID_EN = 0b0 and TAG_ID_EN = 0b0 then       # No sample IDs
    SMPID_32_WORDS = 0
else if INCR_ID_EN = 0b1 and TAG_ID_EN = 0b0 then  # Only the incrementing ID
    SMPID_32_WORDS = 1
else if INCR_ID_EN = 0b0 and TAG_ID_EN = 0b1 then  # Only the tag ID
  if TAG_LEN < 32 then
    SMPID_32_WORDS = 2
  else if TAG_LEN < 64 then
    SMPID_32_WORDS = 3
  else if TAG_LEN < 96 then
    SMPID_32_WORDS = 4
  else if TAG_LEN < 128 then
    SMPID_32_WORDS = 5
else if INCR_ID_EN = 0b1 and TAG_ID_EN = 0b1 then  # Both sample IDs
  if TAG_LEN < 32 then
    SMPID_32_WORDS = 3
  else if TAG_LEN < 64 then
    SMPID_32_WORDS = 4
  else if TAG_LEN < 96 then
    SMPID_32_WORDS = 5
  else if TAG_LEN < 128 then
    SMPID_32_WORDS = 6
```

DATA_PER_32 is the number of data items in a 32-bit word and is calculated as follows:

```
if MON_DATA_WIDTH > 31 AND MON_DATA_WIDTH < 64 then
    DATA_PER_32 = 0.5
else if P == 0 OR MON_DATA_WIDTH > 15 then
    DATA_PER_32 = 1
else if MON_DATA_WIDTH > 7
    DATA_PER_32 = 2
else
    DATA_PER_32 = 4
```

2.8.2.1 Memory size location example:

I The following gives an example calculation for the number of 32-bit words that are required to store the monitor sample data set with the configuration shown in Table 2.2. This is a mix of the static and dynamic configuration of an MGI.

Table 2.2: Example MGI static and dynamic configuration

Configuration	Related register value	Value
8 monitors	MGI_GRP_ID.MON_NUM	7
1 data value from each monitor	MGI_DATA_INFO.DATA_PER_MON	0
12-bit data	MGI_DATA_INFO.MON_DATA_WIDTH	11
Monitor data is packed.	MGI_DATA_INFO.PACKED	1
Incrementing count sample ID	MGI_WRCFG.INCR_ID_EN MGI_WRCFG.TAG_ID_EN	1 0
Start and end sample identifiers are written.	MGI_WRCFG.NUM_SAMPLE_ID	2
Monitor Group ID is not written.	MGI_WRCFG.GRP_ID	0
Data Valid bits are not written.	MGI_WRCFG.DATA_VLD_EN	0

Calculation

I The calculation for this configuration is as follows:

```
SMPID_32_WORDS = 1.

DATA_PER_32 = 2.

SAMPLE_ID_WORDS = (2 * 1) + 0 = 0 = 2.

DATA_WORDS = ((7+1) * (0+1)) / 2) = 4.

TOTAL_WORDS = 2 + 4 = 6.
```

This is the requirement for a single MGI as specified in Table 2.2. In a system with multiple MGIs, this calculation is performed for each MGI. Then, the requirement from each MGI is summed to reach a combined memory requirement.

2.9 MLI connection and disconnection

I MLI connection and disconnection is used to support MLIs that can become unavailable independently of its MGI and the software controlling that MGI. This feature is optional and can be configured for each MLI. MLIs without this feature are considered permanently connected.

A typical use case is when an MLI can power on and off separately from its MGI based on hardware mechanisms or controlled by software that is independent from the software that is controlling that MGI.

To make use of this method, an MLI must have a controlled procedure related to powering on and off. In Arm components, this would typically be a Q-Channel or P-Channel performing functional power control of the component. For more details on these interfaces and their use, see [1] and [2].

The connection status of monitors can be determined using the Monitor Connection Status Register (MGI_CON_STAT). The connection status for monitor x can be determined from MGI_CON_STAT[x]. Table 2.3 shows the monitor connection status values.

Table 2.3: Monitor connection status

MGI_CON_STAT[x]	Monitor x Status
0b0	Disconnected
0b1	Connected

R An MGI must be in a relatively always-on power domain to any of its MLIs that are using the connection and disconnection mechanism.

I This relatively always-on power domain relationship means that, if an MGI can be powered off, it must be powered on before, or simultaneously, with any of its MLIs.

2.9.1 Connection sequence

I When an MLI wants to connect to its MGI, the following sequence is performed:

1. The MLI sends a Monitor Connect command to its MGI.
   - The MLI must only send this command when it is ready to receive commands from its MGI.
2. Upon receiving this command, an MGI:
   a. Records the relevant MLI as connected, setting MGI_CON_STAT[x] to 0b1.
   b. Sends any required Enable Monitor and Set Monitor Mode commands to the relevant MLI to set its current enable and mode settings.
   c. Sends a Monitor Connect Acknowledge command.

An MGI can then send further commands generated for this MLI.

2.9.2 Disconnection sequence

I When an MLI wants to disconnect from an MGI, the following sequence is performed:

1. The MLI sends a Monitor Disconnect command to its MGI.
   - This MLI still needs to process and respond to any incoming commands from its MGI, until it receives the Monitor Disconnect Acknowledge command.
   - This MLI must not initiate any new commands unless they are responses to MGI commands.
2. Upon receipt of this command, its MGI:

a. Records the relevant MLI as disconnected, setting MGI_CON_STAT[x] to 0b0.
b. Completes any currently outstanding commands and responses to the relevant MLI. It must not initiate any new commands to that MLI unless they are responses to that MLIs commands.
c. Sends a Monitor Disconnect Acknowledge command to the relevant MLI.
   - After this command, an MGI must not send any more commands to that MLI, until it receives a Monitor Connect command from that MLI.

3. Upon receipt of the Monitor Disconnect Acknowledge command this MLI can perform actions, like powering off, that depend upon it not receiving commands.

2.10 Monitor Group Interface interrupt events

I An MGI has a single interrupt that can be triggered by multiple interrupt events.

The following is a list of supported interrupt events:

- Monitor Sample Data Set Complete
- Monitor Enable Request Complete
- Monitor Mode Request Complete
- User-Defined Command Received
- Error
- Monitor Trigger
- Input Trigger
- Configuration Request
- Data Write Complete
- Alert

Interupts are controlled using the following registers:

- Interrupt Status Register (MGI_IRQ_STAT)
- Interrupt Mask Register (MGI_IRQ_MASK)

S Software can read MGI_IRQ_STAT to determine the event that caused an interrupt and write to clear the interrupt status.

S Software can program MGI_IRQ_MASK to mask interrupt events.

2.10.1 Monitor Sample Data Set Complete

I This event occurs when the monitor sample data set is complete. This event is used to indicate that monitor data is ready to be collected and processed.

R The Monitor Sample Data Set Complete event occurs when the monitor sample data set is complete following a monitor sample start.

2.10.2 Monitor Enable Request Complete

I This event indicates that the enabling and or disabling of one or more monitors is complete. It occurs when the requested state of monitors becomes equal to the actual monitor status.

R The Monitor Enable Request Complete event occurs when MGI_MON_REQ becomes equal to MGI_MON_STAT.

2.10.3 Monitor Mode Set

I This event indicates that monitor mode setting is complete. It occurs when the requested mode of the monitors becomes equal to the actual monitor mode.

R The Monitor Mode Set event occurs when MGI_MODE_REQ<n> becomes equal to MGI_MODE_STAT<n>.

2.10.4 User-Defined Command Received

I This event indicates that a User-Defined command has been received from an MLI.

R The User-Defined Command Received event occurs when MGI_CMD_RECV0 and MGI_CMD_RECV1 are updated because a User-Defined command is received from an MLI.

2.10.5 Error

I This event indicates that an error command has been received from an MLI, or that an internal MGI error has been generated. For more information on errors, see Chapter 5 *Error codes*.

R The Error event occurs when MGI_ERR_CODE is updated because an error command is received from an MLI or there is an internally generated MGI error.

2.10.6 Monitor Trigger

I This event indicates that a Monitor Trigger command has been received from an MLI.

R The Monitor Trigger event occurs when a Monitor Trigger command is received from an MLI.

2.10.7 Input Trigger

I This event indicates that a trigger is received on the input trigger interface. For more information on the input trigger, see section 2.6 *Input and output triggers*.

R The Input Trigger event occurs when a trigger is received on the input trigger interface.

2.10.8 Configuration Request

I This event indicates that an MGI requires configuration following a reset or power on event. It can be used to ensure that recently powered on components are configured to provide monitor data.

R The Configuration Request event occurs following a reset deassertion when an MGI is capable of being programmed.

U For a component without a Q-Channel or P-Channel Low Power Interface (LPI), or other power control interface, this interrupt occurs following reset deassertion.

U For a component with a power control Q-Channel, this interrupt is sent when the power control Q-Channel moves from Q_STOPPED to Q_RUN.

U For a component with a power control P-Channel, this interrupt is sent when the P-Channel transitions from a non-functional mode to a functional mode, for example, from OFF to ON.

2.10.9 Data Write Complete

I This event indicates that an MGI DMA interface has completed writing of the monitor sample data set to a memory-mapped location. This can be used by a processing agent to allow it to read and process the monitor sample data set.

R The Data Write Complete event occurs when the writing of the monitor sample data set to a memory-mapped location is completed.

U If an end sample identifier is configured to be written, then the monitor sample data set writing is only complete when this end sample identifier is written.

I For more information on sample identifiers and external memory data storage, see section 4.2 *Sample identifiers* and section 4.3 *Memory-mapped data storage*.

2.10.10 Alert

I This event indicates that a programmed alert condition has been triggered.

I There is a separate event indication for each supported alert.

R The Alert event occurs when the relevant alert condition is triggered.

2.11 Configuration options

This section describes the parameters for specifying the configuration and optional features of the Monitor Group Interface (MGI).

The configurations and features that are supported is determined by reading the following registers:

- Group Identification Register (MGI_GRP_ID)
- Group Data Information Register (MGI_DATA_INFO)
- Feature Identification Register 0 (MGI_FEAT0)
- Feature Identification Register 1 (MGI_FEAT1)

Table 2.4 shows an overview of the configuration options.

Table 2.4: MGI configuration options

Config. Parameter	Value Range	Description
GRP_ID_CFG	0-4095	Specifies a unique identifier for an MGI.
MON_NUM_CFG	0-31	Specifies the number of monitors in an MGI. The number of monitors is MON_NUM_CFG+1.
DATA_PER_MON_CFG	0-65535	Specifies the number of data values generated from each monitor. The number of data values generated is DATA_PER_MON_CFG+1.
MON_DATA_WIDTH_CFG	0-63	Specifies the bit width of each monitor data value. The data width is MON_DATA_WIDTH_CFG+1.
ALT_ADDR_CFG	0/1	Specifies where monitor data is read from, it is either: 0: The MGI_DATA<n> registers. 1: The address specified in MGI_RADDR0 / MGI_RADDR1.
DEF_RADDR_CFG	64-bit address	Specifies the default alternate read address. This sets the default value for MGI_RADDR0 / MGI_RADDR1. Only required if A_CFG = 1.
PACKED_CFG	0/1	Specifies if monitor data is packed. 0: Data is not packed. 1: Data is packed. For more information see 4.1.1 *Packed Data*.
PER_TIMER_CFG	0/1	Specifies the presence of the periodic timer. 0: The periodic timer is not present. 1: The periodic timer is present. For more information see 2.4.1.2 *Periodic sample*.
DMA_IF_CFG	0/1	Specifies the presence of the DMA interface. 0: DMA interface is not present. 1: DMA interface is present. For more information see 2.8 *Direct Memory Access interface*.
DEF_WADDR_CFG	64-bit address	Specifies the default write address for the DMA interface. This sets the default value for MGI_WADDR0 / MGI_WADDR1. Only required if DMA_IF_CFG = 1.
MODE_REG_CFG	0-4	Specifies the number of MGI_MODE_REQ/STAT pairs. A value of 0 means that no MGI_MODE_REQ/STAT pairs are present.
MODE_LEN_CFG	0-31	Specifies the bit width of each MGI_MODE_REQ/STAT. The number of bits is MODE_LEN_CFG+1.

Config. Parameter	Value Range	Description
SMP_DLY_LEN_CFG	0-32	Specifies the bit width of MGI_SMP_DLY. The number of bits in SMP_DLY_LEN_CFG. A value of 0 means this register is not present.
TRIG_OUT_CFG	0/1	Specifies the presence of the output trigger interface. 0: The trigger out interface is not present. 1: The trigger out interface is present.
TRIG_IN_CFG	0/1	Specifies the presence of the input trigger interface. 0: The trigger in interface is not present. 1: The trigger in interface is present.
TAG_IN_CFG	0/1	Specifies the presence of the tag input. 0: The tag input is not present. 1: The tag input is present.
TAG_LEN_CFG	0-127	Specifies the bit width of the tag input. The tag bit width is TAG_LEN_CFG+1.
ALT_DELTA_CFG	0/1	Specifies the presence of alert rising and falling delta functions. 0: The rising and falling delta functions are not present. 1: The rising and falling delta functions are present.
ALERT_NUM_CFG	0-7	Specifies the number of alerts present. A value of 0 means that no alerts are present.
USER_DEF_CMD_CFG	0/1	Specifies if an MGI supports User-Defined commands. 0: User-Defined commands are not supported. 1: User-Defined commands are supported.
DEF_CFG_IRQ_MASK	0/1	Specifies the default value of the configuration interrupt event mask. Sets the reset value of MGI_IRQ_MASK.CFG_IRQ_MASK.
DEF_CFG_TRIG_MASK	0/1	Specifies the default value of the configuration trigger event mask. Sets the reset value of MGI_TRG_MASK.CFG_TRIG_MASK.
MON_DISCON_CFG	32-bit Value	Specifies if each monitor supports being disconnected. Each bit represents a monitors disconnection support. For example, bit 0 represents monitor 0, and bit 3 represents monitor 3. 0b0: Monitor does not support being disconnected. It is reset to connected. 0b1: Monitor supports being disconnected. It is reset to disconnected. Bits for unused monitors must be set to 0b0. Sets the reset value of MGI_FEAT0.MON_DISCON,MGI_DISCON_ID and MGI_CON_STAT.

2.11.1 Monitor configuration

I This section contains additional information on configuration options of the SMCF MGI.

2.11.1.1 Monitor data width

I The monitor data width is configured to size the SMCF MGI appropriately for the monitor data widths that it supports.

If an MGI supports monitors with more than one data width, it is configured to support the largest data width.

U When data is sent from a monitor with a data width smaller than its MGI supports, its MGI pads the MSBs of the data with zeros.

2.11.2 Monitor optional features

2.11.2.1 DMA interface

I When the DMA interface is supported, DMA_IF_CFG = 1, an MGI is able to write monitor data to a memory-mapped location.

S Whether the DMA interface is supported is determined by reading MGI_FEAT0.DMA_IF.

R If the DMA interface is not supported, MGI_WRCFG is RES0.

R If the DMA interface is not supported, MGI_WREN is RES0.

R If the DMA interface is not supported, MGI_WADDR0 is RES0.

R If the DMA interface is not supported, MGI_WADDR1 is RES0.

2.11.2.2 Alerts

I The number of alerts, ALERT_NUM, is configured from 0 to 7. This determines how many MGI_ATYP and MGI_AVAL registers are present. There is one of each of these registers for each alert supported.

S The number of alerts supported is determined by reading MGI_FEAT0.ALERT_NUM.

2.11.2.3 Alert delta functions

I When the rising and falling delta function is supported these conditions are available to be used to trigger alerts.

I If these functions are not supported, they cannot be used as alert trigger conditions and cannot be programmed in the MGI_ATYP<n>.ALT_TYP field.

S Whether the alert delta functions are supported is determined by reading MGI_FEAT0.ALT_DELTA.

2.11.2.4 Tag input

I When the tag input is supported, this value is available for a monitor sample data set from (MGI_TAG<n>_START) and (MGI_TAG<n>_END) or as a sample identifier for data written out by the DMA interface.

S Whether the tag input is supported is determined by reading MGI_FEAT0.TAG_IN.

I If the tag input is not supported, it cannot be programmed as the sample identifier in MGI_WRCFG.TAG_ID_EN.

R If the tag input is not supported, then all (MGI_TAG<n>_START) and (MGI_TAG<n>_END) are RES0.

2.11.2.5 Input trigger

I When the input trigger is supported this is available to trigger sampling or output triggers. See section 2.6.1 *Input trigger interface*.

S Whether the input trigger is supported is determined by reading MGI_FEAT0.TRIG_IN.

I If the input trigger is not supported, this cannot be programmed as the sample type in MGI_SMP_CFG.SMP_TYP.

R If the input trigger is not supported, then MGI_TRG_MASK.IN_TRIG_TRIG_MASK is RES0.

R If the input trigger is not supported, there is no related interrupt event, therefore, MGI_IRQ_STAT.IN_TRIG_IRQ_STAT and MGI_IRQ_MASK.IN_TRIG_IRQ_MASK are RES0.

2.11.2.6 Output trigger

I When the output trigger is supported, the MGI interrupt and input trigger events are capable of producing a output trigger event, see section 2.6.2 *Output trigger interface*.

S Whether the output trigger is supported is determined by reading MGI_FEAT0.TRIG_OUT.

R If the output trigger is not supported, MGI_TRG_MASK is RES0.

2.11.3 User-Defined commands

I When User-Defined commands are supported, software can use MGI_CMD_SEND0, MGI_CMD_SEND1, MGI_CMD_RECV0 and MGI_CMD_RECV1 to send and receive IMPLEMENTATION DEFINED commands to and from the monitors.

I If User-Defined commands are not supported, MGI_CMD_SEND0, MGI_CMD_SEND1, MGI_CMD_RECV0 and MGI_CMD_RECV1 are RES0.

2.11.4 Monitor connection and disconnection configuration

I When monitor disconnection is supported for monitor x, MGI_DISCON_ID[x] = 1, monitor x is disconnected at MGI reset. Monitor x can connect and disconnect with commands between an MLI and its MGI.

I When monitor disconnection is not supported for monitor x, MGI_DISCON_ID[x] = 0, monitor x is always connected.

S Whether monitor connection and disconnection is supported by any monitor is determined by reading MGI_FEAT0.MON_DISCON.

R If any bit of MON_DISCON_CFG is HIGH, then MGI_FEAT0.MON_DISCON is HIGH.

S Whether monitor connection and disconnection is supported for each monitor can be determined by reading MGI_DISCON_ID.

Chapter 3
Commands

This section defines the commands that are used by the System Monitoring Control Framework (SMCF) to communicate between the Monitor Group Interface (MGI) and the Monitor Local Interface (MLI).

It also defines when commands are sent in relation to register accesses in an MGI, and other events and actions that result from the receipt of commands.

3.1 Command overview

R Table 3.1 shows an overview of the available commands.

Table 3.1: Command overview

ID	Command	CMD Address / Sub ID	CMD Data Value	Data Width	From	To
0b00000	Enable Monitor	None	None	0	MGI	MLI
0b00001	Disable Monitor	None	None	0	MGI	MLI
0b00010	Start Sample	None	None	0	MGI	MLI
0b00011	Set Monitor Mode	MGI_MODE_REQ No.[a]	MODE_REQ	32	MGI	MLI
0b00100	Clear Monitor Error	N/A	Error Type	8	MGI	MLI
0b00101	Reserved	N/A	N/A	N/A	N/A	N/A
0b10110	Monitor Connect Ack	None	None	0	MGI	MLI
0b10111	Monitor Disconnect Ack	None	None	0	MGI	MLI
0b00101 - 0b01111	Reserved	N/A	N/A	N/A	N/A	N/A
0b10000	Monitor Enabled	None	None	0	MLI	MGI
0b10001	Monitor Disabled	None	None	0	MLI	MGI
0b10010	Monitor Data	Monitor Data Width	Monitor Data	8/16/32/64	MLI	MGI
0b10011	Monitor Mode Complete	MGI_MODE_REQ No.[a]	MODE_REQ / Completion Status	32	MLI	MGI
0b10100	Monitor Error	None	Error Code	8	MLI	MGI
0b10101	Monitor Trigger	None	None	0	MLI	MGI
0b10110	Monitor Connect	None	None	0	MLI	MGI
0b10111	Monitor Disconnect	None	None	0	MLI	MGI
0b10110 - 0b11110	Reserved	N/A	N/A	N/A	N/A	N/A
0b11111	User-Defined	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	32	Any	Any

[a] For example, a value of 0 indicates that the value comes from MGI_MODE_REQ0, and 2 indicates the value comes from MGI_MODE_REQ2.

3.2 Command descriptions

The following sections describe the commands in more detail.

3.2.1 Enable Monitor command

I The Enable Monitor command is used to enable a monitor, so it is ready to start a sample when instructed. For example, this command can be used to remove the monitor from a low power mode, so it is available for sampling.

I An Enable Monitor command is sent from an MGI to the MLI for monitor x when a write of 0b1 occurs to MGI_MON_REQ[x] and that MLI is connected, but disabled.

R An Enable Monitor command is sent from an MGI to the MLI for monitor x when all the following are true:

- There is a write of 0b1 to MGI_MON_REQ[x].
- MGI_MON_STAT[x] is 0b0.
- MGI_CON_STAT[x] is 0b1.

Or when all the following are true:

- An MGI receives a Monitor Connect command.
- MGI_MON_STAT[x] is 0b1.

I If a monitor is disconnected, MGI_CON_STAT[x] is 0b0, then Enable Monitor commands are not sent to that MLI. Instead MGI_MON_STAT[x] is updated immediately with the value written to MGI_MON_REQ[x]. For more information see 2.2 *Enabling and disabling monitors*.

R When an MLI receives an Enable Monitor command it performs IMPLEMENTATION DEFINED actions to enable the monitor so that it can start a sample when instructed, then it sends a Monitor Enabled command to its MGI.

R If an MLI receives an Enable Monitor command when the monitor is already enabled, or no actions are required to enable the monitor, it immediately sends a Monitor Enabled command to its MGI.

3.2.2 Disable Monitor command

I The Disable Monitor command is used to disable a monitor if its data is not required, typically so that it can enter a low-power state.

I A Disable Monitor command is sent from an MGI to the MLI for monitor x when a write of 0b0 occurs to MGI_MON_REQ[x] and that MLI is connected and enabled.

R A Disable Monitor command is sent to from an MGI to the MLI for monitor x when all the following are true:

- There is a write of 0b0 to MGI_MON_REQ[x].
- MGI_MON_STAT[x] is 0b1.
- MGI_CON_STAT[x] is 0b1.

I If a monitor is disconnected, MGI_CON_STAT[x] is 0b0, then Disable Monitor commands are not sent to that MLI, instead MGI_MON_STAT[x] is updated immediately with the value written to MGI_MON_REQ[x]. For more information see 2.2 *Enabling and disabling monitors*.

R When an MLI receives a Disable Monitor command, it performs IMPLEMENTATION DEFINED actions to disable the monitor, then sends a Monitor Disabled command to its MGI.

R If an MLI receives a Disable Monitor command when the monitor is already disabled, or no actions are required to disable the monitor, it immediately sends a Monitor Disabled command to its MGI.

R If an MLI receives a Disable Monitor command when there is an ongoing sample it can choose to either:

- Disable the monitor immediately by taking the following actions:

1. Stop the sampling and disable the monitor.
2. Return zero data to its MGI with a Monitor Data command.
3. Send a Monitor Disabled command to its MGI.

- Wait until the sample is complete by taking the following actions:
  1. Wait until the monitor sample is complete.
  2. Return the sample data to its MGI with a Monitor Data command.
  3. Disable the monitor.
  4. Send a Monitor Disabled command to its MGI.

3.2.3 Start Sample command

I The Start Sample command is used to start a monitor sample. This command is sent from an MGI to all MLI whose monitors are enabled and connected when a trigger to start a sample occurs.

R A Start Sample command is sent from an MGI to the MLI for monitor x when all the following are true:

- A start sample trigger occurs.
- MGI_MON_REQ[x] and MGI_MON_STAT[x] are 0b1.
- MGI_CON_STAT[x] is 0b1.

R When a Start Sample command is sent, MGI_DVLD is cleared.

R When an MLI receives a Start Sample command, it performs IMPLEMENTATION DEFINED actions to start a monitor sample.

I For more information on sampling, see section 2.4 *Monitor sampling*.

3.2.4 Set Monitor Mode command

I The Set Monitor Mode command is used to set the mode of the monitor. Typically, this is used to configure the type of sampling performed or other IMPLEMENTATION DEFINED monitor settings required for it to operate as desired. When there is a write to any MGI_MODE_REQ<n>, a Set Monitor Mode command is sent to all monitors that are connected, enabled, and configured for mode broadcast in MGI_MODE_BCAST.

I Additionally, the Set Monitor Mode command is sent to an MLI, for each MGI_MODE_REQ<n> present, when an MGI receives a Monitor Connect command from that MLI.

R A Set Monitor Mode command is sent from an MGI to the MLI for monitor x when all the following are true:

- A write occurs to any MGI_MODE_REQ<n>.
- MGI_MON_REQ[x] and MGI_MON_STAT[x] are both 0b1.
- MGI_CON_STAT[x] is 0b1.
- MGI_MODE_BCAST[x] is 0b1.

The data for the Set Monitor Mode command is the content of the written MGI_MODE_REQ<n>.

Additionally, a Set Monitor Mode command is sent from an MGI to the MLI for monitor x, for each MGI_MODE_REQ supported, when all the following are true:

- An MGI receives a Monitor Connect command from monitor x.
- MGI_MON_REQ[x] and MGI_MON_STAT[x] are 0b1.
- MGI_MODE_REQn is present.
  - The number of MGI_MODE_REQ<n> present is indicated by MGI_FEAT1.MODE_REG.

The data for these commands is the last Monitor Mode value that was sent to this MLI from the relevant MGI_MODE_REQ<n>. If no data was sent to an MLI previously then the default value is sent.

One command is sent for each MGI_MODE_REQ<n> that is present, the command address specifies which MGI_MODE_REQ<n> the data is from.

X On receipt of a Monitor Connect command the monitor mode data is only sent if the monitor is enabled, if software enables a monitor it is required to manually reconfigure the monitor.

U The mapping of MGI_MODE_REQ<n> values to monitor functions is IMPLEMENTATION DEFINED.

R The Set Monitor Mode command address bits [1:0] indicate which MGI_MODE_REQ<n> the data is from. Table 3.2 shows the encoding.

Table 3.2: Monitor Mode Request register encoding

CMD Address [1:0]	Mode Request Register
0b00	MGI_MODE_REQ0
0b01	MGI_MODE_REQ1
0b10	MGI_MODE_REQ2
0b11	MGI_MODE_REQ3

R When an MLI receives a Set Monitor Mode command it performs IMPLEMENTATION DEFINED actions to set the mode of the monitor according to the address and data values sent with the command. It then sends a Monitor Mode Complete command to its MGI duplicating the address and data values of the received Set Monitor Mode command.

R If an MLI receives a Set Monitor Mode command and the monitor indicated in the monitor ID is disabled, then it sends a Monitor Error command with a Monitor Mode on Disabled Monitor error code.

R If an MLI receives a Set Monitor Mode command and the command data is not a supported value, then it sends a Monitor Error command with a Monitor Mode error code to its MGI.

3.2.5 Clear Monitor Error command

I The Clear Monitor Error command is used to clear an error in the monitor or MLI.

R A Clear Monitor Error command is sent from an MGI to an MLI when a Monitor Error Interrupt Event is cleared and the related monitor is enabled and connected.

R The Clear Monitor Error command is sent to the MLI indicated in MGI_ERR_CODE.MON_ID. The monitor ID is set to indicate this monitor value.

R The Clear Monitor Error command data is set to the error type in MGI_ERR_CODE.ERROR_CODE. For error types see Chapter 5 *Error codes*.

R When an MLI receives a Clear Monitor Error command it can take IMPLEMENTATION DEFINED actions to remove the error status. It is IMPLEMENTATION DEFINED if the error condition that generated the error is removed.

3.2.6 Monitor Connect Acknowledge command

I The Monitor Connect Acknowledge command is used to acknowledge the connection of an MLI.

R A Monitor Connect Acknowledge command is sent from an MGI to an MLI when a Monitor Connect command is received. It is only sent when the MGI has completed any necessary Set Monitor Mode and Enable Monitor commands as part of the connection sequence. For more information see section 2.9 *MLI connection and disconnection*.

R When an MLI receives a Monitor Connect Acknowledge command it is allowed to send commands other than responses to the connection sequence Set Monitor Mode and Enable Monitor commands.

3.2.7 Monitor Disconnect Acknowledge command

I The Monitor Disconnect Acknowledge command is used to acknowledge the disconnection of an MLI.

R A Monitor Disconnect Acknowledge command is sent from an MGI to an MLI when a Monitor Disconnect command is received and the MGI has completed any outstanding commands.

R The Monitor Disconnect Acknowledge command must be the last command an MGI sends to an MLI, until the MGI receives another Monitor Connect command from that MLI.

U When an MLI receives a Monitor Disconnect Acknowledge command it can perform actions, like powering off, that depend upon it not receiving commands.

3.2.8 Monitor Enabled command

I The Monitor Enabled command is used to indicate that a monitor has been enabled.

R A Monitor Enabled command is sent from an MLI to its MGI in response to a Enable Monitor command when monitor enabling is complete, or when the monitor is already enabled.

R When an MGI receives a Monitor Enabled command from monitor x it records the monitor as enabled by:

- Setting MGI_MON_STAT[x] to 0b1.

I The MGI_MON_STAT[x] might already be 0b1 if the Monitor Enabled command is sent because of the sequence initiated when a Monitor Connect command is received, in this case it remains 0b1. For more information see section 2.9 *MLI connection and disconnection*

3.2.9 Monitor Disabled command

I The Monitor Disabled command is used to indicate that a monitor has been disabled.

R A Monitor Disabled command is sent from an MLI to its MGI in response to a Disable Monitor command when monitor disabling is complete, or when the monitor is already disabled.

R When an MGI receives a Monitor Disabled command from monitor x it records the monitor as disabled by:

- Setting MGI_MON_STAT[x] to 0b0.

3.2.10 Monitor Data command

I This command is used to send monitor sample data from an MLI to an MGI.

R The command address field bits [1:0] for this command contains the width of the command data field.

If the width of monitor data generated at an MLI does not match one of these lengths, the next largest value is chosen. Command data MSBs are padded with zeros to make the data width align with that specified in the command address field.

Table 3.3 shows the data width encoding.

Table 3.3: Monitor data width encoding

Value	Data Width
0b00	8-Bits
0b01	16-Bits
0b10	32-Bits
0b11	64-Bits

R The Monitor Data command is sent from an MLI to its MGI when a monitor sample completes. Multiple monitor data commands are sent if more than one data value, or data values larger than 32-bits, are generated from the monitor sample. Monitor data commands are sent as many times as required to send all the data produced from the monitor sample.

R If a monitor produces multiple data values from a sample, as indicated by MGI_DATA_INFO.DATA_PER_MON, each data value is sent in a separate Monitor Data command.

U An MGI determines the required amount of monitor data for the monitor sample data set from:

- The number of monitors it started sampling.
- The number of data values generated from each monitor, (MGI_DATA_INFO.DATA_PER_MON).
- The size of each data value, (MGI_DATA_INFO.MON_DATA_WIDTH).

R The Monitor Data command data is the sample data or portion of sample data being returned.

R Monitor data must always be sent in the same order.

R If sending data larger than 32-bits, which requires more than one Monitor Data command, data bits 31:0 are sent first, followed by the higher bit values in the following command. These commands must be sent sequentially.

R When a Monitor Data command is received at an MGI it sets the appropriate MGI_DATA<n> with the received monitor sample data value. When it has received all required monitor data from monitor x it sets MGI_DVLD[x] to 0b1.

R If a Monitor Data command is the first response to a Start Sample command, including error and disconnect responses, from any monitor then:

- The MGI_SMPID_START incrementing count sample ID value is incremented.
- The MGI_TAG<n>_START values, if supported, are updated with the current tag input value.

R A monitor sample data set is complete when all MLIs that were sent the Start Sample command for the most recent trigger event have returned either:

- All data from the MLI monitor sample.
- A Sample or Sample on Disabled error.
- A Monitor Disconnect command.

R If the receipt of a Monitor Data command completes the monitor sample data set then:

- If unmasked, the Monitor Sample Complete interrupt event occurs.
- The MGI_SMPID_END and MGI_TAG<n>_END values are updated to be equal to their equivalent start register values.
- If data writing is supported, and is enabled, an MGI begins writing the monitor data to the memory-mapped location specified in MGI_WADDR0 and MGI_WADDR1. For more information, see 2.8 *Direct Memory Access interface*.

3.2.11 Monitor Mode Complete command

I The Monitor Mode Complete command is used to indicate that the setting of a monitor mode has been completed. It also contains an indication of the success or failure of setting the mode.

R A Monitor Mode Complete command is sent from an MLI to its MGI in response to a Set Monitor Mode command when the IMPLEMENTATION DEFINED actions required because of the Set Monitor Mode command are complete.

R The Monitor Mode Complete command data value is the same as the command data value in the Set Monitor Mode command to which this is a response.

R The Monitor Mode Complete command address value bits [1:0] indicate which MGI_MODE_REQ<n> the command relates, the encoding for these bits are shown in Table 3.4.

Table 3.4: Monitor Mode Complete command address field status encoding

CMD Address [1:0]	Mode Request Register
0b00	MGI_MODE_REQ0
0b01	MGI_MODE_REQ1
0b10	MGI_MODE_REQ2
0b11	MGI_MODE_REQ3

R The Monitor Mode Complete command address value bits [7:6] represents the success or failure of the mode setting, the encoding for these bits are shown in Table 3.5.

Table 3.5: Monitor Mode Complete command address field status encoding

CMD Address [7:6]	Mode Completion Status
0b00	Success
0b01	Failure - Invalid Mode Value
0b10	Failure - Monitor Disabled
0b11	Reserved

R A monitor mode setting is complete when a Monitor Mode Complete command has been received from all monitors to which a Set Monitor Mode command was sent following a MGI_MODE_REQ<n> write.

When the monitor mode setting is complete:

- If all Monitor Mode Complete commands report success, then the MGI_MODE_STAT<n> is set to the MGI_MODE_REQ<n> value.
- If any Monitor Mode Complete command reports a failure, then the respective MGI_MODE_STAT<n> is not updated.

X Not updating the MGI_MODE_STAT<n> in the event of a failure prevents software incorrectly assuming that the mode setting has successfully completed from a read of this register. This could be the case if it reads this status before the error interrupt event, generated by the failure, has been processed.

R When an MGI receives a Monitor Mode Complete command with a failure status:

- If there is not already an active Monitor Error Interrupt Event:

  - Set the MGI_ERR_CODE Second Error (SECOND_ERROR) bit to 0b0 to indicate this is the first error.
  - Set the MGI_ERR_CODE Monitor ID (MON_ID) field to the value from the Monitor Mode Complete command.
  - Set the MGI_ERR_CODE Error Code (ERROR_CODE) field to either Monitor Mode or Monitor Mode on Disabled Monitor error dependent on the failure status.
- If there is already an active Monitor Error Interrupt Event:
  - Set MGI_ERR_CODE.SECOND_ERROR to 0b1 to indicate that another error has occurred.

X If one or more monitor mode settings fails it is indicated by an error interrupt and software can take IMPLEMENTATION DEFINED actions to correct and then retry the mode setting.

3.2.12 Monitor Error command

I The Monitor Error command is used to report an error from a monitor or MLI.

R A Monitor Error command is sent from an MLI to its MGI when an error condition occurs, with an identifier of the error type. For error types see Chapter 5 *Error codes*.

R When the command is received at an MGI:

- If there is not already an active Monitor Error Interrupt Event:
  - Set the MGI_ERR_CODE Second Error (SECOND_ERROR) bit to 0b0 to indicate this is the first error.
  - Set the MGI_ERR_CODE Monitor ID (MON_ID) and Error Code (ERROR_CODE) fields to those in the Monitor Error command.
- If there is already an active Monitor Error Interrupt Event:
  - Set MGI_ERR_CODE.SECOND_ERROR to 0b1 to indicate that another error has occurred.
- If the error type is a Sample Error or Sample on Disabled Error, in addition to setting the registers in the preceding conditions, this is used as receipt of data for that sample and sets all data values for the sample to zero
  - If this command is the first response, including data and disconnect responses, to a Start Sample from any monitor then the MGI_SMPID_START and MGI_TAG<n>_START values are incremented and updated with the current tag input value respectively.
  - If the receipt of this command completes the monitor sample data set then the MGI_SMPID_END and MGI_TAG<n>_END values are updated to be equal to their equivalent start registers.

If two or more errors are received at the same time, then the error from the lowest number MLI is processed first.

3.2.13 Monitor Trigger command

I The Monitor Trigger command is used to report a trigger event from an MLI. The source of the trigger event that is generated is IMPLEMENTATION DEFINED.

R A Monitor Trigger command is sent from an MLI to an MGI when an IMPLEMENTATION DEFINED trigger event in an MLI, or monitor occurs.

R When the Monitor Trigger command is received at an MGI:

- If unmasked, set the Monitor Trigger Interrupt Event. See 2.10.6 *Monitor Trigger*.
- If supported and unmasked, send a trigger event on the output trigger.

I It is not supported to start a sample based on a monitor trigger.

3.2.14 Monitor Connect command

I The Monitor Connect command is used to connect an MLI to its MGI so it can start receiving commands.

R A Monitor Connect command is sent from an MLI to its MGI when it wants to start receiving commands.

R If an MLI is disconnected, the Monitor Connect command must be the first command sent to its MGI.

R Following the sending of the Monitor Connect command an MLI must not send further commands until it has received the Monitor Connect Acknowledge command.

R At the point of sending the Monitor Connect command an MLI must be ready to receive commands from its MGI.

R When an MGI receives the Monitor Connect command it performs the following sequence:

1. Sets MGI_CON_STAT[x] to 0b1.
2. Sends the relevant MLI commands for the current monitor mode using one or more Set Monitor Mode commands.
3. Sends the relevant MLI an Enable Monitor command if the monitor is enabled.
4. Sends the relevant MLI an Monitor Connect Acknowledge command.

Following this sequence, an MGI can send any further commands that are targeted at the relevant MLI.

Commands generated after the Monitor Connect command was received must be sent after the Monitor Connect Acknowledge command.

X This ordering of the commands prevents race conditions. For example, if a monitor is enabled when the Monitor Connect command is received but is requested to be disabled before the Monitor Connect Acknowledge command is sent, the monitor might be left in an incorrect enabled state. The monitor is enabled as part of the connection process, but then disabled after the acknowledge is sent.

R If an MGI receives a Monitor Connect command from an MLI for which it does not support disconnection then an Unknown Command error code is generated.

3.2.15 Monitor Disconnect command

I The Monitor Disconnect command is used to disconnect an MLI from its MGI so it can stop receiving commands from its MGI. This is typically because the monitor is being requested to enter a low power mode.

R A Monitor Disconnect command is sent from an MLI to its MGI when it wants to stop receiving commands from its MGI.

R Following the sending of this command the MLI must process commands from its MGI until it receives the Monitor Disconnect Acknowledge command.

R When an MGI receives this command it must perform the following sequence:

1. Sets MGI_CON_STAT[x] to 0b0.
2. If there was a sample ongoing at the respective MLI, treat this command as receipt of data for that sample and set all data values for the sample to zero.
   - If this command is the first response, including data and disconnect responses, to a Start Sample from any monitor then the MGI_SMPID_START and MGI_TAG<n>_START values are incremented and updated with the current tag input value respectively.
   - If the receipt of this command completes the monitor sample data set then the MGI_SMPID_END and MGI_TAG<n>_END values are updated to be equal to their equivalent start registers.
3. Complete any other currently outstanding commands and responses to the relevant MLI, so that it does not expect to send or receive any more commands from the relevant MLI. It must not start any new commands to the relevant MLI that are not responses to existing commands.
4. Send the Monitor Disconnect Acknowledge command.
   - When this acknowledge is sent by an MGI it must not send any more commands to the relevant MLI until it receives a Monitor Connect command.

R If an MGI receives a Monitor Disconnect command from an MLI for which it does not support disconnection then an Unknown Command error code is generated.

3.2.16 User-Defined command

I The User-Defined command is used to control additional IMPLEMENTATION DEFINED monitor functions.

U IMPLEMENTATION DEFINED monitor configuration is covered as part of the monitor mode commands, (3.2.4 *Set Monitor Mode command* and 3.2.11 *Monitor Mode Complete command*), and registers, (MGI_MODE_REQ<n> and MGI_MODE_STAT<n>), so should not require the use of these commands.

R The User-Defined command uses the command address/sub-ID as the command identifier. See 3.3 *Command format* for more details.

U Mappings of User-Defined command sub-IDs to specific command identifiers are IMPLEMENTATION DEFINED.

I These commands are processed explicitly using MGI_CMD_SEND{<0|1>} and MGI_CMD_RECV{<0|1>}.

R A User-Defined command is sent by an MGI with the values set in MGI_CMD_SEND{<0|1>} when MGI_CMD_SEND0 is written.

R When a User-Defined command is received by an MGI:

- The User-Defined command information is put into the MGI_CMD_RECV{<0|1>}.
- If unmasked, set the User-Defined Command Received interrupt event.

S A User-Defined command can only be sent as a broadcast command when a response is not required.

R The data width of User-Defined commands is always 32-bits. User-Defined commands requiring more than 32-bits of data must send the data in multiple User-Defined commands.

X This fixed data width makes the sending and receiving of IMPLEMENTATION DEFINED commands by an MGI generic by not requiring additional support like a configurable amount of data registers.

3.3 Command format

I The command format is the organization for commands transported between MGI and MLI.

The format supports both serial, including MSI, and parallel interfaces.

R On a serial interface the address section bits 31 to 0 are sent first, followed by the data bits, if required, from MSB to LSB.

R For a parallel interface the address and data sections are placed on the respective address and data buses.

R Commands must be processed in the order they are received.

R Figure 3.1 shows the command format.

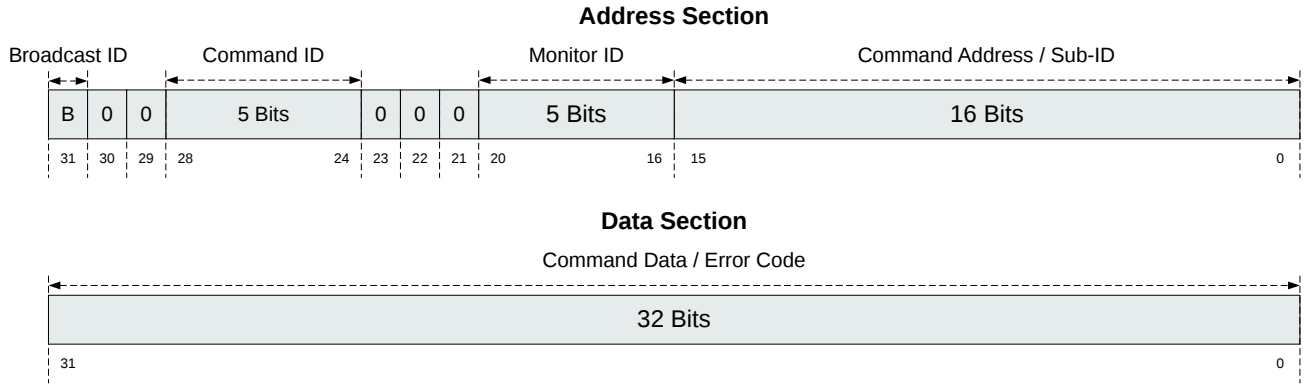


Figure 3.1: Command format

3.3.1 Broadcast identifier and monitor identifier

I The command contains a broadcast identifier and monitor identifier.

R The broadcast ID, when 0b1, indicates the command is to be sent to all enabled and connected monitors, in this case the monitor ID is set to 0b00000. When the broadcast ID is 0b0 the command is sent to the monitor indicated by the monitor ID.

3.3.2 Command address/sub-ID

I This field specifies further information about the command, its value is command specific, see 3.1 *Command overview*.

R If a command does not require a command address/sub-ID value but requires command data then:

- For a serial interface, the command address/sub-ID bits are set to all zeros.
- For a parallel interface, the command address/sub-ID bits are set to all zeros.

R If a command does not require both command address/sub-ID and command data then:

- For a serial interface, the command address/sub-ID bits are not sent.
- For a parallel interface, the command address/sub-ID bits are set to all zeros.

3.3.3 Command data

I The amount of data required by the command can be determined from the command ID and the monitor data width (MON_DATA_WIDTH).

The command data width is rounded up to the nearest supported data width, either 8, 16, 32 or 64 bits.

R On a serial interface all data bits are sent following the address in the order MSB to LSB.

I On a parallel interface, if the data requirement is greater than the data width of the bus interface, additional commands are sent with the same address until all the data is sent. The data is sent with the most significant portion first. The number of commands needed depends on the data requirement and the width of the parallel data bus.

R If no data is required for a command, then for a serial interface no data bits are sent. For a parallel interface, the data bus is set to all zeros.

Chapter 4
Data formats

This section describes the data formats in which monitor sample data sets are stored both in Monitor Group Interface (MGI) registers and memory-mapped locations.

4.1 Data formats

I The formatting of data depends upon the data width produced by the monitors and how it is configured to be stored. This section describes these parameters and their effects.

S The data format parameters are read from the Data Information Register (MGI_DATA_INFO). They are the MON_DATA_WIDTH field, indicating the monitor data width, and the PACKED bit, indicating if the data is packed or not.

R In all cases data values for monitors that are not enabled or connected at the sample start, or that return errors, are represented by zero values.

R Monitor data values are stored in little-endian format.

4.1.1 Packed Data

I Packed data refers to having multiple monitor data values concatenated in a single 32-bit Data Register (MGI_DATA) or memory-mapped location.

R When data is unpacked each 32-bit location contains either:

- a single monitor data value in the case the monitor data width is equal to or less than 32-bits.
- part of a single monitor data value in the case the monitor data width is greater than 32-bits.

I When data is packed, each 32-bit location can contain more than one monitor data value, dependent on the monitor data width.

R Table 4.1 shows the number of monitor data values in each 32-bit location based on if the monitor data is packed and the width of the monitor data.

Table 4.1: Data values in each 32-bit register

Packed	Data Width	Data Values in each 32 Bit Register	Data Alignment
No	Any	1, or part of 1 (if data width > 32)	LSB at bit 0
Yes	Greater than 16 bits	1, or part of 1 (if data width > 32)	LSB at bit 0
Yes	9 to 16 bits	2	LSBs at bits 0 / 16
Yes	1 to 8 bits	4	LSBs at bits 0 / 8 / 16 / 24

I Figure 4.1 shows some examples of monitor data formats.

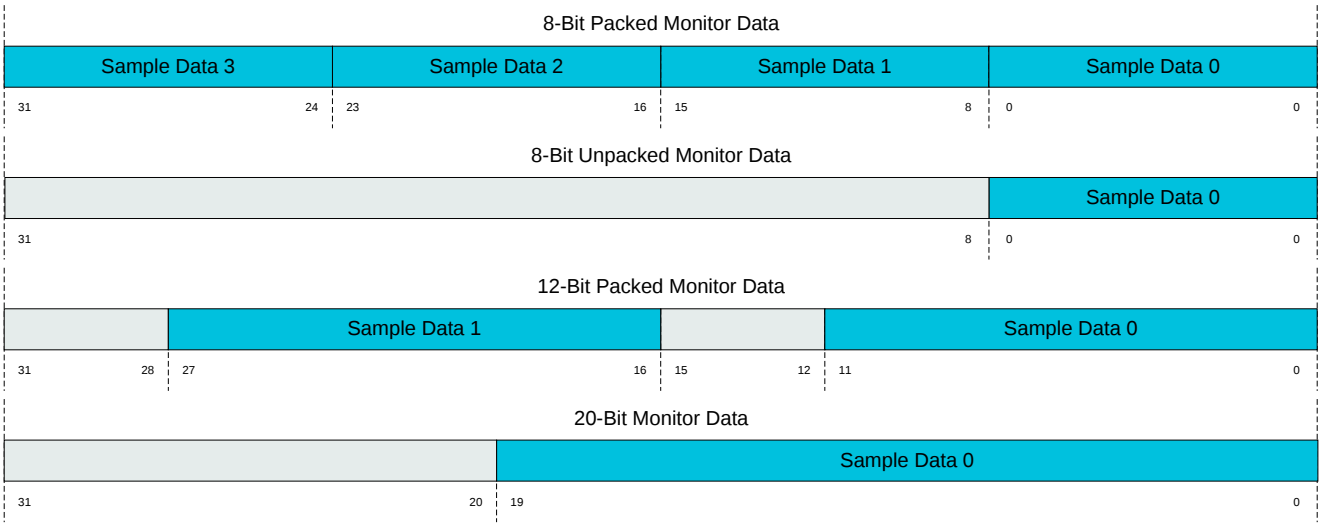


Figure 4.1: Monitor data value packed data format examples

4.1.1.1 Data values greater than 32 bits

R Data value widths greater than 32 bits take up the number 32-bit locations required to fit their value. They never share 32-bit locations with other values.

I Figure 4.2 shows some examples for storage of monitor data values greater than 32-bits.

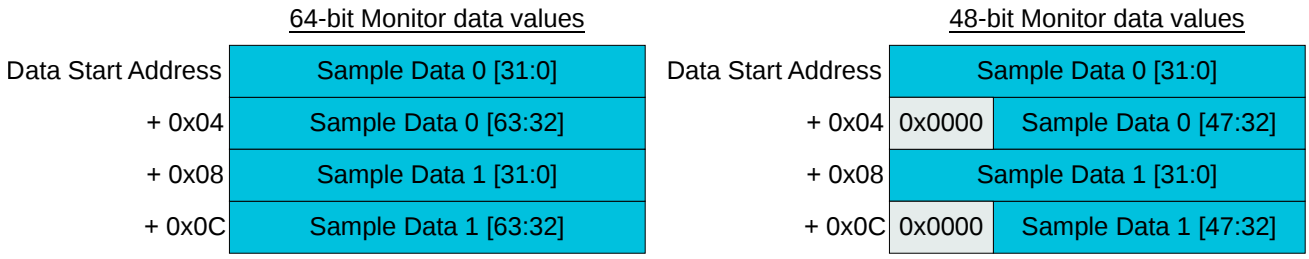


Figure 4.2: Storage example for monitor data values greater than 32-bits

4.1.2 Data Storage

I The MGI programmers model supports up to 512 32-bit registers for data storage.

I Table 4.2 shows the number of monitor data values these 512 registers can accommodate based on the monitor sample data width and if the data values are packed.

Table 4.2: Programmers model monitor data storage capacity

Data Width	Packed	Number of data values
32 to 64-bits	N/A	256
Up-to 32-bits	No	512
16 to 32-bits	Yes	512
9 to 16-bits	Yes	1024
1 to 8-bits	Yes	2056

4.1.2.1 Alternate data read address

I If the required capacity for monitor data values is greater than can be provided by the internal MGI address space an alternate monitor data address can be specified. All other MGI functions remain the same.

The alternate address support and the alternate data read address are specified as design time configuration values ALT_ADDR_CFG and DEF_RADDR_CFG.

R This alternate monitor data address is aligned on a 4KByte boundary.

U The size and location of this alternate monitor data address space is not limited any further than specified.

S Whether the monitor data is read from the alternate address is determined by reading MGI_DATA_INFO.ALT_ADDR.

S The alternate data read address can be read from MGI_RADDR0 and MGI_RADDR1

4.2 Sample identifiers

I Sample identifiers are used to uniquely identify a monitor sample data set, they can be:

- An incrementing count value, increasing by one for each monitor sample data set.
- A tag value, based on a tag input value recorded when the first sample data is received.
  - This is an optional feature.

A sample identifier can also be used to indicate the status of the data, for example if it is uninitialized or is not being generated as an MGI and related monitors are powered off.

The sample identifier is duplicated as a start sample identifier and end sample identifier. These are updated at different times to allow for the validity of the monitor sample data set to be checked. When the monitor sample data set is valid these values are identical.

The Sample Identifier Start and End registers, MGI_SMPID_START and MGI_SMPID_END, contain the incrementing count sample ID. When the tag input is supported the Tag Start and Tag End registers, MGI_TAG<n>_START and MGI_TAG<n>_END, contain the tag value sample ID.

When using the DMA interface to write out the monitor sample data set to a memory-mapped location, the sample identifiers can also be written. This can be configured to include either the incrementing count sample ID, the tag value sample ID, or both. It can also include either the start sample ID only, or both the start and end sample IDs. These options are configured using the Data Write Configuration Register (MGI_WRCFG).

R The start sample identifier is written immediately before the first data from the monitor sample data set is updated.

R The end sample identifier is written immediately after the last data from the monitor sample data set is updated.

U The sample identifier is written when the monitor sample data set is updated. For example, the sample identifier is updated in MGI_SMPID_START immediately before the first sample data set value is updated in MGI_DATA<n>. This is different to when the sample identifier is updated by the DMA in the memory-mapped location. This sample identifier is updated immediately before the entire monitor sample data set is written.

R When the incrementing count sample ID value reaches its maximum value it wraps back to zero.

S Software reading the monitor sample data set can use the sample identifiers to identify the sample and check that the sample was not updated during the read. The procedure is as follows:

1. Software reads both start and end sample identifiers.
2. Software checks that the start sample ID equals the end sample ID.
   - If they are not equal, the monitor sample data set is currently being updated. Software should abandon the process and repeat from the start.
3. Software reads the monitor sample data set.
4. Software reads both start and end sample IDs.
5. Software checks the start sample ID equals the end sample ID.
   - If they are not equal, this indicates that a monitor sample data set update has started during the reading process. Software should abandon the process and repeat from the start.
6. Software checks the start and end sample ID values are equal to those read before the monitor sample data was read.
   - If they are not the same this indicates a complete update of the data has occurred during the monitor sample data set read. Software should abandon the process and repeat from the start.
7. If all checks are correct, the monitor data values are valid and can be used.

4.2.1 Sample identifier format

I This section describes the format of the sample identifier.

4.2.1.1 Sample Start and End Register sample identifier format

I This section describes the format of the sample identifier presented in MGI_SMPID_START and MGI_SMPID_END.

R Bits [31:28] of the sample identifier indicates its type, the remainder of the bits represent different items dependent on the type.

Table 4.3 shows the sample identifier encoding.

Table 4.3: Sample Start and End Register sample identifier encoding

Bits[31:28]	ID type / status	Associated value in bits [27:0]
0x0	Uninitialized/Invalid	Set to Zero
0x1-0x7	Reserved	N/A
0x8	Count ID	Monitor Sample Data Set Count Value
0x9-0xF	Reserved	N/A

4.2.1.2 Tag Start and End Register sample identifier format

I This section describes the format of the sample identifier presented in MGI_TAG<n>_START and MGI_TAG<n>_END.

R MGI_TAG<n>_START and MGI_TAG<n>_END contain the tag input value directly with no additional status information.

4.2.1.3 Memory-mapped location sample identifier format

I This section describes the format of the sample identifier when written to a memory mapped location.

R Bits [31:28] of the sample identifier indicates its type, the remainder of the bits represent different items dependent on the type.

Table 4.4 shows the sample identifier encoding.

Table 4.4: Memory-mapped location sample identifier encoding

Bits[31:28]	ID type / status	Associated value in bits [27:0]
0x0	Uninitialized/Invalid	Set to Zero
0x1	Powered off	Set to Zero
0x2-0x7	Reserved	N/A
0x8	Count ID	Monitor Sample Data Set Count Value
0x9	Tag ID	No. of following 32-bit values that represent the tag value
0xA-0xF	Reserved	N/A

Figure 4.3 shows some examples of the sample ID formats.

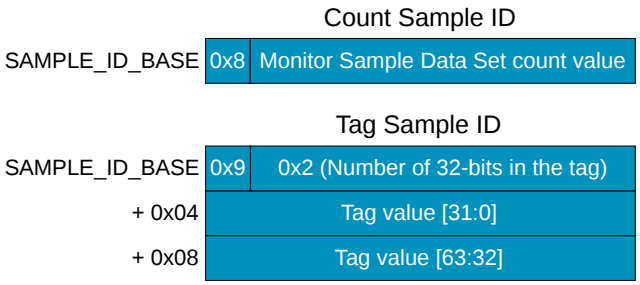


Figure 4.3: Example sample ID formats

4.3 Memory-mapped data storage

I When data is written to a memory-mapped location with the DMA interface it can optionally be written with:

- A monitor group ID
- The data valid bits
- Sample IDs, consisting of:
  - Start sample IDs only
  - Start and End sample IDs

For more details on sample IDs, see section 4.2 *Sample identifiers*

Figure 4.4 shows the data structure when the monitor sample data set is written to a memory-mapped location.

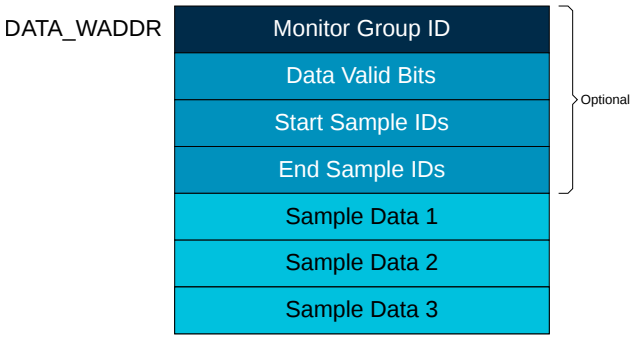


Figure 4.4: Monitor sample data set memory-mapped data structure

R The entire monitor sample data set is written out, including zeroed values for monitors disabled or disconnected at the sample start, or that return an error or disconnected during the monitor sample.

I Figure 4.5 shows data structure examples when the monitor sample data set is written to a memory-mapped location. Not all possible configurations are shown.

In these examples, every 32-bit word contains only one monitor data value. However, if the data is packed or is larger than 32-bits then this might not be the case. See section 4.1 *Data formats*.

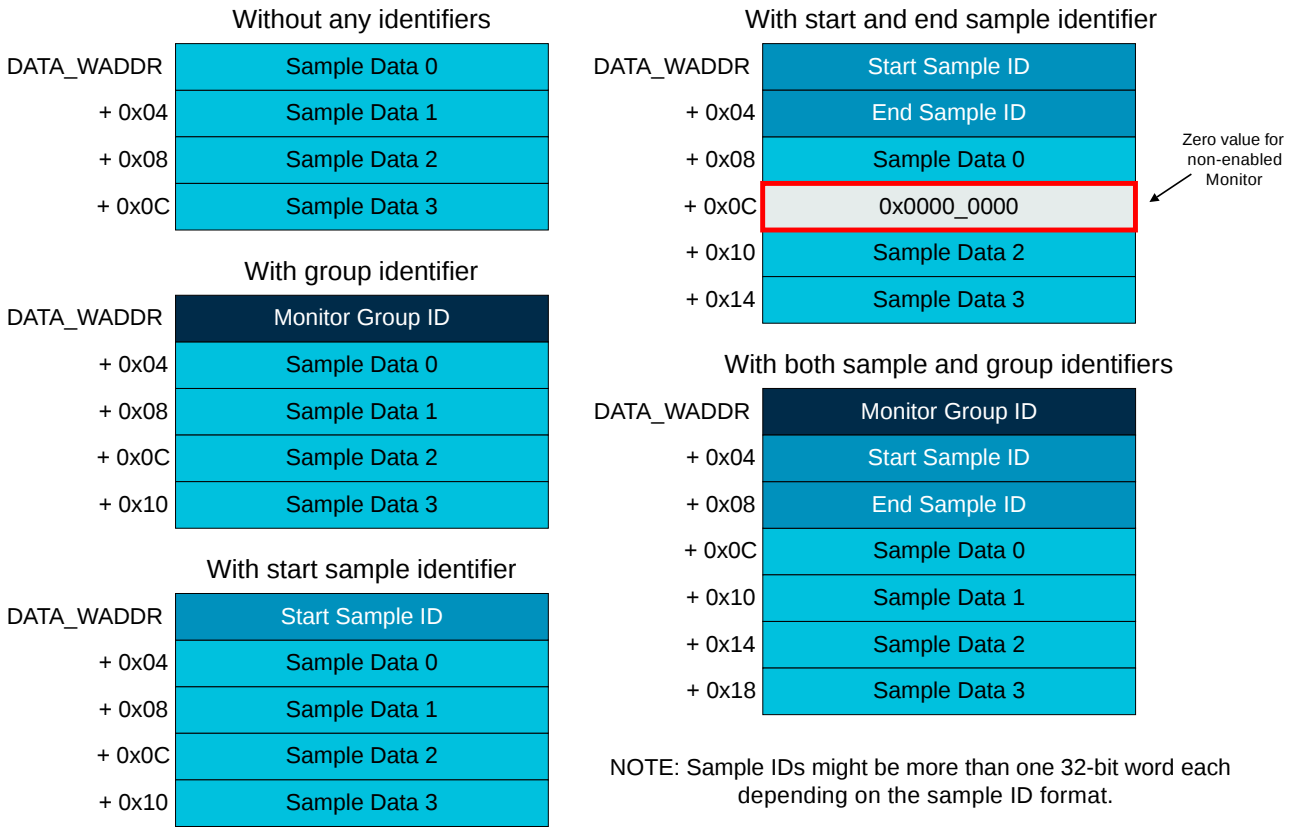


Figure 4.5: Monitor sample data set memory-mapped data structure examples

Chapter 5
Error codes

This section describes the errors that can be generated by the Monitor Group Interface (MGI) and the associated error codes that can be read from the Error Code Register (MGI_ERR_CODE).

5.1 Error overview

I Errors can be generated because of either:

- An internal MGI error.
- A Monitor Error command sent from a Monitor Local Interface (MLI) to an MGI.
- Because of a failure status in any other command sent from an MLI to its MGI.

For more information on the Monitor Error command see 3.2.12 *Monitor Error command*.

R Table 5.1 shows an overview of the error types and their associated codes.

Table 5.1: Error code overview

Error ID	Error Type
63:32	IMPLEMENTATION DEFINED
31:18	Reserved
23	Unknown Command - MGI
22:17	Reserved
16	Sample Period Warning
15:8	Reserved
7	Unknown Command - MLI
6	Reserved
5	Monitor Mode on Disabled Monitor Error
4	Monitor Mode Error
3	Monitor Disable Error
2	Monitor Enable Error
1	Sample on Disabled Monitor Error
0	Sample Error

5.1.1 Errors from the Monitor Error Command

I The following errors occur because of a Monitor Error command from an MLI.

5.1.1.1 Sample Error

I The Sample Error indicates there was an error during monitor sampling.

I This can be a result of an IMPLEMENTATION DEFINED error from the monitor itself, or if the monitor receives a Start Sample command when it has not been configured correctly.

R If an MGI receives this error in response to a Start Sample command, it counts this as received data for the purposes of monitor sample data set completion.

5.1.1.2 Sample on Disabled Monitor Error

I The Sample on Disabled Monitor Error indicates a Start Sample command was received when the monitor was not enabled.

R If an MGI receives this error in response to a Start Sample command, it counts this as received data for the purposes of monitor sample data set completion.

5.1.1.3 Monitor Enable Error

I The Monitor Enable Error indicates there was an error enabling the monitor.

R If an MGI receives this error in response to a Enable Monitor command, it keeps the monitor recorded as disabled in MGI_MON_STAT.

5.1.1.4 Monitor Disable Error

I The Monitor Disable Error indicates there was an error disabling the monitor.

R If an MGI receives this error in response to a Disable Monitor command, it keeps the monitor recorded as enabled in MGI_MON_STAT.

5.1.1.5 Monitor Mode Error

I The Sample Error indicates there was an error during monitor mode setting.

I This can be because of an IMPLEMENTATION DEFINED error from the monitor during mode setting, or because of the monitor receiving a Set Monitor Mode command with an invalid or unknown value.

I This error occurs because of a failure status in the Monitor Mode Complete command. For more information see 3.2.11 *Monitor Mode Complete command*

S In some cases, only a sub-set of the possible values that can be programmed in MGI_MODE_REQ<n> might represent valid monitor modes, therefore a Monitor Mode Error might indicate incorrect programming.

5.1.1.6 Monitor Mode on Disabled Monitor Error

I The Monitor Mode on Disabled Monitor Error indicates a Set Monitor Mode command was received when the monitor was not enabled.

I This error occurs because of a failure status in the Monitor Mode Complete command. For more information see section 3.2.11 *Monitor Mode Complete command*

R If an MGI receives a Monitor Mode Error from an MLI in response to a Set Monitor Mode command, it counts this as an acknowledgment of the command from that MLI.

5.1.1.7 Unknown Command Error - MLI

I The Unknown Command Error indicates a command sent to an MLI is not supported by that MLI.

5.1.1.8 Implementation Defined Errors

I These error IDs represent IMPLEMENTATION DEFINED errors. This allows monitor specific errors to be reported.

5.1.2 Internal MGI errors

I The following errors occur because of an internal error in an MGI.

5.1.2.1 Sample Period Error

I The Sample Period Error indicates that the programmed sample period is less than the monitor sample time.

R This error occurs when the sample period expiring would trigger a sample start while a monitor sample is still ongoing.

S This is a potential issue because it indicates that monitor data is generated less often than every programmed sample period, meaning potentially less often than is expected by the programming agent. For more information see section 2.4.1.2 *Periodic sample*

5.1.2.2 Unknown Command Error - MGI

I The Unknown Command Error indicates a command sent to an MGI is not supported.

Chapter 6
Monitor Group Interface Programmers Model

This section describes the programmer's model of the Monitor Group Interface (MGI).

6.1 Register Summary

Table 6.1 shows an overview of the registers.

Any address space not specified is reserved as RES0.

Table 6.1: Register summary

Address	Register Name	Short Name
0x000	Group Identification Register	MGI_GRP_ID
0x008	Group Data Information Register	MGI_DATA_INFO
0x010	Feature Identification Register 0	MGI_FEAT0
0x018	Feature Identification Register 1	MGI_FEAT1
0x030	Sample Enable Register	MGI_SMP_EN
0x038	Sample Configuration Register	MGI_SMP_CFG
0x040	Sample Period Register	MGI_SMP_PER
0x048	Sample Delay Register	MGI_SMP_DLY
0x060	Monitor Enable Request Register	MGI_MON_REQ
0x070	Monitor Enable Status Register	MGI_MON_STAT
0x080	Monitor Mode Broadcast Register	MGI_MODE_BCAST
0x090	Monitor Mode Request Register 0	MGI_MODE_REQ0
0x098	Monitor Mode Request Register 1	MGI_MODE_REQ1
0x0A0	Monitor Mode Request Register 2	MGI_MODE_REQ2
0x0A8	Monitor Mode Request Register 3	MGI_MODE_REQ3
0x0C0	Monitor Mode Status Register 0	MGI_MODE_STAT0
0x0C8	Monitor Mode Status Register 1	MGI_MODE_STAT1
0x0D0	Monitor Mode Status Register 2	MGI_MODE_STAT2
0x0D8	Monitor Mode Status Register 3	MGI_MODE_STAT3
0x100	Interrupt Status Register	MGI_IRQ_STAT
0x110	Interrupt Mask Register	MGI_IRQ_MASK
0x140	Output Trigger Mask Register	MGI_TRG_MASK
0x150	Error Code Register	MGI_ERR_CODE
0x160	Data Write Enable Register	MGI_WREN
0x168	Data Write Configuration Register	MGI_WRCFG
0x170	Data Write Address Register 0	MGI_WADDR0
0x174	Data Write Address Register 1	MGI_WADDR1
0x020	Data Read Address Register 0	MGI_RADDR0
0x024	Data Read Address Register 1	MGI_RADDR1

Address	Register Name	Short Name
0x190	Monitor Disconnection Identification Register	MGI_DISCON_ID
0x198	Monitor Connection Status Register	MGI_CON_STAT
0x1B0	Command Send Register 0	MGI_CMD_SEND0
0x1B8	Command Send Register 1	MGI_CMD_SEND1
0x1C0	Command Receive Register 0	MGI_CMD_RECV0
0x1C8	Command Receive Register 1	MGI_CMD_RECV1
0x200	Alert 0 Type Register	MGI_ATYP0
0x208	Alert 0 Value Register	MGI_AVAL0
0x210	Alert 1 Type Register	MGI_ATYP1
0x218	Alert 1 Value Register	MGI_AVAL1
0x220	Alert 2 Type Register	MGI_ATYP2
0x228	Alert 2 Value Register	MGI_AVAL2
0x230	Alert 3 Type Register	MGI_ATYP3
0x238	Alert 3 Value Register	MGI_AVAL3
0x240	Alert 4 Type Register	MGI_ATYP4
0x248	Alert 4 Value Register	MGI_AVAL4
0x250	Alert 5 Type Register	MGI_ATYP5
0x258	Alert 5 Value Register	MGI_AVAL5
0x260	Alert 6 Type Register	MGI_ATYP6
0x268	Alert 6 Value Register	MGI_AVAL6
0x700-0xEFF	Data Register <n>	MGI_DATA<n>
0xF00	Data Valid Register	MGI_DVLD
0xF10	Tag Start Register 0	MGI_TAG0_START
0xF14	Tag Start Register 1	MGI_TAG1_START
0xF18	Tag Start Register 2	MGI_TAG2_START
0xF1C	Tag Start Register 3	MGI_TAG3_START
0xF20	Tag End Register 0	MGI_TAG0_END
0xF24	Tag End Register 1	MGI_TAG1_END
0xF28	Tag End Register 2	MGI_TAG2_END
0xF2C	Tag End Register 3	MGI_TAG3_END
0xF40	Sample Identifier Start Register	MGI_SMPID_START
0xF48	Sample Identifier End Register	MGI_SMPID_END
0xFC0	Implementation Identification Register	MGI_IIDR
0xFC8	Architecture Identification Register	MGI_AIDR
0xFD0-0xFFF	IMPLEMENTATION DEFINED Identification Registers	N/A

6.1.1 MGI_GRP_ID, Group Identification Register

The MGI_GRP_ID characteristics are:

Purpose

This register contains information like an identifier for this MGI and the number of monitors that are supported.

Attributes

MGI_GRP_ID is a 32-bit register.

Field descriptions

The MGI_GRP_ID bit assignments are:

Bits [31:21]

Reserved, RES0.

MON_NUM, bits [20:16]

Specifies the number of Monitors in the MGI.

The number of monitors in the MGI is MON_NUM + 1.

This field is reset to the MON_NUM_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [15:12]

Reserved, RES0.

GRP_ID, bits [11:0]

Monitor Group Interface Identifier Value.

This field is reset to the GRP_ID_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_GRP_ID

MGI_GRP_ID can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x000

Access on this interface is **RO**.

6.1.2 MGI_DATA_INFO, Group Data Information Register

The MGI_DATA_INFO characteristics are:

Purpose

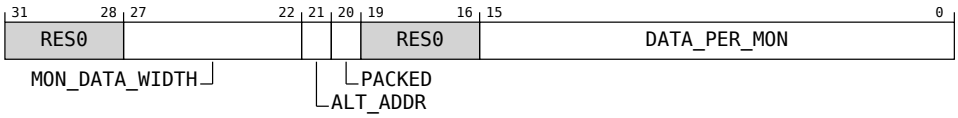
This register contains information about the data generated by the monitors in this MGI.

Attributes

MGI_DATA_INFO is a 32-bit register.

Field descriptions

The MGI_DATA_INFO bit assignments are:



Bits [31:28]

Reserved, RES0.

MON_DATA_WIDTH, bits [27:22]

The bit width of the data values generated by each monitor.

The data value width is MON_DATA_WIDTH + 1.

This field is reset to the MON_DATA_WIDTH_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

ALT_ADDR, bit [21]

Alternate Address Bit.

This field is reset to the A_CFG configuration value, see 2.11 *Configuration options*.

Value	Meaning
0b0	Monitor data is read from MGI_DATA<n>.
0b1	Monitor data is read from the address in MGI_RADDR0 / MGI_RADDR1.

This field resets to an IMPLEMENTATION DEFINED value.

PACKED, bit [20]

Packed Bit.

This indicates multiple monitor data values are placed in a single 32-bit register, see 4.1.1 *Packed Data*.

This field is reset to the P_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [19:16]

Reserved, RES0.

DATA_PER_MON, bits [15:0]

The number of data values generated per monitor for each sample.

The number of data values generated is DATA_PER_MON + 1.

This field is reset to the DATA_PER_MON_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_DATA_INFO

MGI_DATA_INFO can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x008

Access on this interface is **RO**.

6.1.3 MGI_FEAT0, Feature Identification Register 0

The MGI_FEAT0 characteristics are:

Purpose

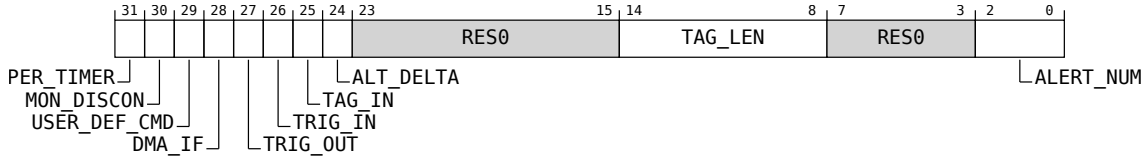
This register contains information about the features and configuration in this MGI

Attributes

MGI_FEAT0 is a 32-bit register.

Field descriptions

The MGI_FEAT0 bit assignments are:



PER_TIMER, bit [31]

Indicates if the periodic timer is present.

Value	Meaning
0b0	The periodic timer is not present.
0b1	The periodic timer is present.

If the periodic timer is not present, then MGI_SMP_PER is RES0

This field is reset to the PER_TIMER_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

MON_DISCON, bit [30]

Indicates if disconnection is supported for any monitor.

Information on disconnection support for each monitor can be read from MGI_DISCON_ID.

Value	Meaning
0b0	Disconnection is not supported on any monitors.
0b1	Disconnection is supported on one or more monitors.

This field is reset based on the MON_DISCON_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

USER_DEF_CMD, bit [29]

Indicates if User Defined commands are supported.

Value	Meaning
0b0	User Defined commands are not supported.
0b1	User Defined commands are supported.

If User Defined commands are not supported then the following registers are RES0: MGI_CMD_SEND0, MGI_CMD_SEND1, MGI_CMD_RECV0 and MGI_CMD_RECV1.

This field is reset to the USER_DEF_CMD_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

DMA_IF, bit [28]

Indicates if the DMA interface is present.

Value	Meaning
0b0	The DMA interface is not present.
0b1	The DMA interface is present.

If the DMA interface is not present, then MGI_WRCFG is RES0

If the DMA interface is not present, then MGI_WREN is RES0

This field is reset to the DMA_IF_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

TRIG_OUT, bit [27]

Indicates if the output trigger is present.

Value	Meaning
0b0	The output trigger is not present.
0b1	The output trigger is present.

This field is reset to the TRIG_OUT_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

TRIG_IN, bit [26]

Indicates if the input trigger is supported.

Value	Meaning
0b0	The input trigger is not present.
0b1	The input trigger is present.

This field is reset to the TRIG_IN_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

TAG_IN, bit [25]

Indicates if the tag input is present.

Value	Meaning
0b0	The tag input is not present.
0b1	The tag input is present.

If the tag interface is not present, then Tag Value cannot be programmed as the sample identifier configuration in MGI_SMP_CFG.SMPID_CFG.

If the tag interface is not present, then all MGI_TAG<n>_START and MGI_TAG<n>_END are RES0.

This field is reset to the TAG_IN_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

ALT_DELTA, bit [24]

Indicates if the alert rising and falling delta functions are supported.

Value	Meaning
0b0	The alert rising and falling delta functions are not present.
0b1	The alert rising and falling delta functions are present.

If these functions are not present they cannot be programmed as alert types in MGI_ATYP0.ALT_TYP.

This field is reset to the ALT_DELTA_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [23:15]

Reserved, RES0.

TAG_LEN, bits [14:8]

Indicates the bit width of the tag value if present, indicated by TAG_IN. The tag length is TAG_LEN+1.

If TAG_LEN < 32 then MGI_TAG1_START and MGI_TAG1_END are RES0.

If TAG_LEN < 64 then MGI_TAG2_START and MGI_TAG2_END are RES0.

If TAG_LEN < 96 then MGI_TAG3_START and MGI_TAG3_END are RES0.

This field is reset to the TAG_LEN_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [7:3]

Reserved, RES0.

ALERT_NUM, bits [2:0]

Indicates the number of supported alerts.

This field is reset to the ALERT_NUM_CFG configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_FEAT0

MGI_FEAT0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x010

Access on this interface is **RO**.

6.1.4 MGI_FEAT1, Feature Identification Register 1

The MGI_FEAT1 characteristics are:

Purpose

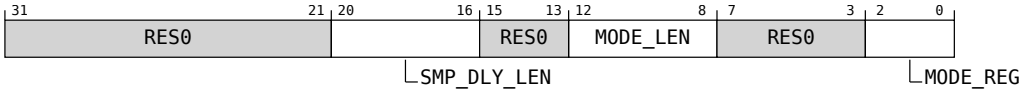
This register contains information about the features and configuration in this MGI

Attributes

MGI_FEAT1 is a 32-bit register.

Field descriptions

The MGI_FEAT1 bit assignments are:



Bits [31:21]

Reserved, RES0.

SMP_DLY_LEN, bits [20:16]

Indicates the number of bits in the Sample Delay Register.

If SMP_DLY_LEN = 0b00000 then MGI_SMP_DLY is RES0.

If SMP_DLY_LEN > 0b00000 then MGI_SMP_DLY is SMP_DLY_LEN bits wide.

This field is reset to the SMP_DLY_LEN_CFG configuration value, see 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [15:13]

Reserved, RES0.

MODE_LEN, bits [12:8]

Indicates the bit width of each of the Monitor Mode Request and Status Registers.

The number of bits in each register is MODE_LEN + 1

This field is reset to the MODE_LEN_CFG configuration value, see 2.11 *Configuration options*

This field resets to an IMPLEMENTATION DEFINED value.

Bits [7:3]

Reserved, RES0.

MODE_REG, bits [2:0]

Indicates the number of Monitor Mode Request and Status Registers.

The number of each register is MODE_REG. For instance, 0b000 means that there are not any Monitor Mode Request or Status Registers, 0b001 means there is one Monitor Mode Request Register and one Monitor Mode Status Register.

Values greater than 0b100 are reserved.

This field is reset to the MODE_REG_CFG configuration value, see 2.11 *Configuration options*

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_FEAT1

MGI_FEAT1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x018

Access on this interface is **RO**.

6.1.5 MGI_SMP_EN, Sample Enable Register

The MGI_SMP_EN characteristics are:

Purpose

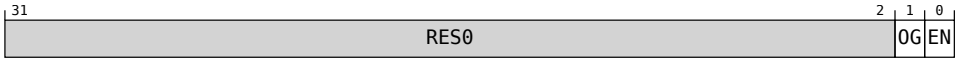
This register contains controls to enable monitor sampling.

Attributes

MGI_SMP_EN is a 32-bit register.

Field descriptions

The MGI_SMP_EN bit assignments are:



Bits [31:2]

Reserved, RES0.

OG, bit [1]

Sample Ongoing.

Indicates when monitor sampling is ongoing.

This bit is read only.

For a single monitor sample (MGI_SMP_CFG.SMP_TYP = 0b00) this is 0b1 when the monitor sample is ongoing, then goes LOW when it is complete.

For other monitor sample types is set to 0b1 when the monitor sample enable (MGI_SMPEN.EN) is set to 0b1. It is set to 0b0 when the sample enable is set to 0b0 and a monitor sample is not currently ongoing.

This field resets to `0b0`.

EN, bit [0]

Sample Enable

Enables monitor sampling.

For more information see 2.4.2 *Enabling monitor sampling*.

This field resets to `0b0`.

Accessing the MGI_SMP_EN

MGI_SMP_EN can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x030

Access on this interface is **RW**.

6.1.6 MGI_SMP_CFG, Sample Configuration Register

The MGI_SMP_CFG characteristics are:

Purpose

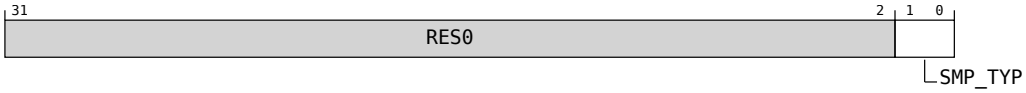
This register contains controls to configure monitor sampling.

Attributes

MGI_SMP_CFG is a 32-bit register.

Field descriptions

The MGI_SMP_CFG bit assignments are:



Bits [31:2]

Reserved, RES0.

SMP_TYP, bits [1:0]

Sample Type.

Value	Meaning
0b00	Manual Sample: A single sample is performed.
0b01	Periodic Sample: A sample is started when the sampling is enabled, then repeated every programmed sample period. If the periodic timer is not supported, MGI_FEAT0.PER_TIMER is 0b0, then this value is reserved.
0b10	Data Read: A sample is started when the sampling is enabled, then repeated every time the data from the last sample data address is read.
0b11	Trigger Input: A sample is started whenever there is a event on the input trigger. If the trigger input is not supported, MGI_FEAT0.TRIG_IN is 0b0, then this value is reserved.

This field resets to 0b00.

Accessing the MGI_SMP_CFG

MGI_SMP_CFG can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x038

Access on this interface is **RO**.

6.1.7 MGI_SMP_PER, Sample Period Register

The MGI_SMP_PER characteristics are:

Purpose

This register programs the sample period for periodic sampling.

If MGI_FEAT0.PER_TIMER is 0b0 this register is RES0.

Attributes

MGI_SMP_PER is a 32-bit register.

Field descriptions

The MGI_SMP_PER bit assignments are:

SMP_PERIOD, bits [31:0]

Length of the periodic sampling sample period in MGI clock cycles.

This field resets to `0x00000000`.

Accessing the MGI_SMP_PER

MGI_SMP_PER can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x040

Access on this interface is **RO**.

6.1.8 MGI_SMP_DLY, Sample Delay Register

The MGI_SMP_DLY characteristics are:

Purpose

This register programs a delay from a sample trigger to a sample starting, for more information see 2.4.3 *Sample delay*.

This registers bit width is MGI_FEAT1.SMP_DLY_LEN.

If MGI_FEAT0.SMP_DLY_LEN is 0b00000, this register is RES0.

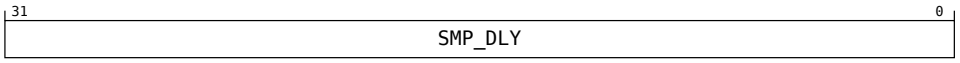
Register bits higher than the bit width supported are RES0.

Attributes

MGI_SMP_DLY is a 32-bit register.

Field descriptions

The MGI_SMP_DLY bit assignments are:



SMP_DLY, bits [31:0]

Length of the sample delay in MGI clock cycles, for more information see 2.4.3 *Sample delay*.

This field resets to `0x00000000`.

Accessing the MGI_SMP_DLY

MGI_SMP_DLY can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x048

Access on this interface is **RW**.

6.1.9 MGI_MON_REQ, Monitor Enable Request Register

The MGI_MON_REQ characteristics are:

Purpose

This register is used to request the enabling and disabling of monitors. For more information, see section 2.2 *Enabling and disabling monitors*.

The actual status of monitors can be read from the Monitor Enable Status Register.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this register width is [5:0].

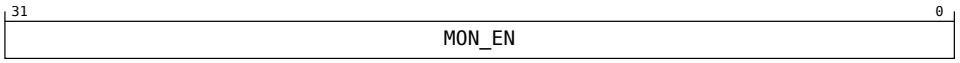
Register bits higher than the number of supported monitors as defined in MGI_GRP_ID.MON_NUM are RES0.

Attributes

MGI_MON_REQ is a 32-bit register.

Field descriptions

The MGI_MON_REQ bit assignments are:



MON_EN, bits [31:0]

Monitor Enable / Disable Request.

A write to this register sends the appropriate requests for monitors to be enabled or disabled.

For example, writing 0b1 to bit 6 is a request to enable monitor 6 if it is not already enabled.

The actual status of monitors can be read from the Monitor Enable Status Register.

Each bit value is either:

0b0: The monitor represented by this bit position is requested to be disabled.

0b1: The monitor represented by this bit position is requested to be enabled.

This field resets to `0x00000000`.

Accessing the MGI_MON_REQ

MGI_MON_REQ can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x060

Access on this interface is **RW**.

6.1.10 MGI_MON_STAT, Monitor Enable Status Register

The MGI_MON_STAT characteristics are:

Purpose

This register is used to read the enabled status of monitors, for more information see 2.2 *Enabling and disabling monitors*.

The enabling and disabled of monitors is requested using Monitor Enable Request Register.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this register width is [5:0].

Register bits higher than the number of supported monitors as defined in MGI_GRP_ID.MON_NUM are RES0.

Attributes

MGI_MON_STAT is a 32-bit register.

Field descriptions

The MGI_MON_STAT bit assignments are:

MON_STAT, bits [31:0]

Monitor Enable Status

Each bit value is either:

0b0: The monitor represented by this bit position is disabled.

0b1: The monitor represented by this bit position is enabled.

This field resets to `0x00000000`.

Accessing the MGI_MON_STAT

MGI_MON_STAT can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x070

Access on this interface is **RO**.

6.1.11 MGI_MODE_BCAST, Monitor Mode Broadcast Register

The MGI_MODE_BCAST characteristics are:

Purpose

This register is used to configure the monitors that are sent a Set Monitor Mode command when any Monitor Mode Request Register is written.

This allows different monitors to be programmed with different modes.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this register width is [5:0].

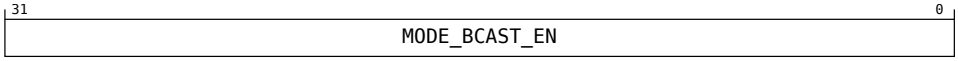
Register bits higher than the number of supported monitors as defined in MGI_GRP_ID.MON_NUM are RES0.

Attributes

MGI_MODE_BCAST is a 32-bit register.

Field descriptions

The MGI_MODE_BCAST bit assignments are:



MODE_BCAST_EN, bits [31:0]

Monitor Mode Broadcast Enable.

Each bit represents a monitor. For example, bit 0 represents monitor 0, and bit 3 represents monitor 3.

The width of this field is MGI_GRP_ID.MON_NUM + 1.

Each bit value is either:

0b0: The monitor will not be sent a Set Monitor Mode when a write to any Monitor Mode Request Register occurs.

0b1: The monitor will be sent a Set Monitor Mode when a write to any Monitor Mode Request Register occurs.

This field resets to `0xFFFFFFFF`.

Accessing the MGI_MODE_BCAST

MGI_MODE_BCAST can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x080

Access on this interface is **RW**.

6.1.12 MGI_MODE_REQ0, Monitor Mode Request Register 0

The MGI_MODE_REQ0 characteristics are:

Purpose

This register programs the monitor mode.

If MGI_FEAT1.MODE_REG < 0b001, this register is RES0.

The bit width of this register is MODE_LEN, unused bits are RES0.

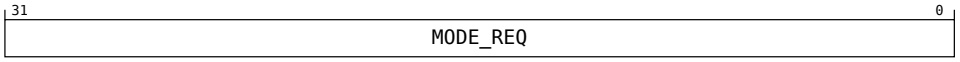
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_REQ0 is a 32-bit register.

Field descriptions

The MGI_MODE_REQ0 bit assignments are:



MODE_REQ, bits [31:0]

Monitor Mode

Writing to this register sends the written value to each connected and enabled monitor with its bit set in MGI_MODE_BCAST using a Set Monitor Mode command.

If this register is written to when its previous value is not equal to that in MGI_MODE_STAT0 then it is not guaranteed that the previous mode setting has taken place.

This field resets to `0x00000000`.

Accessing the MGI_MODE_REQ0

MGI_MODE_REQ0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x090

Access on this interface is **RW**.

6.1.13 MGI_MODE_REQ1, Monitor Mode Request Register 1

The MGI_MODE_REQ1 characteristics are:

Purpose

This register programs the monitor mode.

If MGI_FEAT1.MODE_REG < 0b010, this register is RES0.

The bit width of this register is MODE_LEN, unused bits are RES0.

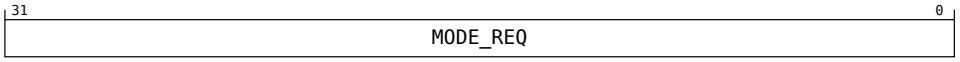
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_REQ1 is a 32-bit register.

Field descriptions

The MGI_MODE_REQ1 bit assignments are:



MODE_REQ, bits [31:0]

Monitor Mode

Writing to this register sends the written value to each connected and enabled monitor with its bit set in MGI_MODE_BCAST using a Set Monitor Mode command.

If this register is written to when its previous value is not equal to that in MGI_MODE_STAT0 then it is not guaranteed that the previous mode setting has taken place.

This field resets to `0x00000000`.

Accessing the MGI_MODE_REQ1

MGI_MODE_REQ1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x098

Access on this interface is **RW**.

6.1.14 MGI_MODE_REQ2, Monitor Mode Request Register 2

The MGI_MODE_REQ2 characteristics are:

Purpose

This register programs the monitor mode.

If MGI_FEAT1.MODE_REG < 0b011, this register is RES0.

The bit width of this register is MODE_LEN, unused bits are RES0.

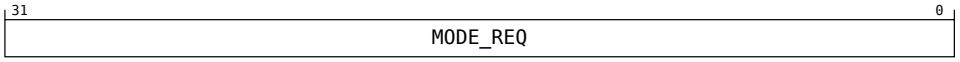
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_REQ2 is a 32-bit register.

Field descriptions

The MGI_MODE_REQ2 bit assignments are:



MODE_REQ, bits [31:0]

Monitor Mode

Writing to this register sends the written value to each connected and enabled monitor with its bit set in MGI_MODE_BCAST using a Set Monitor Mode command.

If this register is written to when its previous value is not equal to that in MGI_MODE_STAT0 then it is not guaranteed that the previous mode setting has taken place.

This field resets to `0x00000000`.

Accessing the MGI_MODE_REQ2

MGI_MODE_REQ2 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0A0

Access on this interface is **RW**.

6.1.15 MGI_MODE_REQ3, Monitor Mode Request Register 3

The MGI_MODE_REQ3 characteristics are:

Purpose

This register programs the monitor mode.

If MGI_FEAT1.MODE_REG < 0b100, this register is RES0.

The bit width of this register is MODE_LEN, unused bits are RES0.

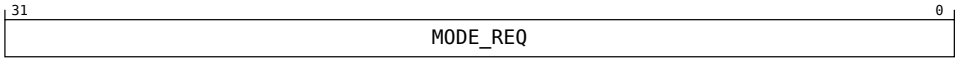
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_REQ3 is a 32-bit register.

Field descriptions

The MGI_MODE_REQ3 bit assignments are:



MODE_REQ, bits [31:0]

Monitor Mode

Writing to this register sends the written value to each connected and enabled monitor with its bit set in MGI_MODE_BCAST using a Set Monitor Mode command.

If this register is written to when its previous value is not equal to that in MGI_MODE_STAT0 then it is not guaranteed that the previous mode setting has taken place.

This field resets to `0x00000000`.

Accessing the MGI_MODE_REQ3

MGI_MODE_REQ3 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0A8

Access on this interface is **RW**.

6.1.16 MGI_MODE_STAT0, Monitor Mode Status Register 0

The MGI_MODE_STAT0 characteristics are:

Purpose

This register contains the current monitor mode.

If MGI_FEAT1.MODE_REG < 0b001, this register is RES0.

This width of this register is MODE_LEN, unused bits are RES0.

For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_STAT0 is a 32-bit register.

Field descriptions

The MGI_MODE_STAT0 bit assignments are:

MODE_STAT, bits [31:0]

Monitor Mode Status

Indication of the current monitor mode.

This can be different from the requested monitor mode if the requested mode state is not yet updated at the monitor.

This field resets to `0x00000000`.

Accessing the MGI_MODE_STAT0

MGI_MODE_STAT0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0C0

Access on this interface is **RO**.

6.1.17 MGI_MODE_STAT1, Monitor Mode Status Register 1

The MGI_MODE_STAT1 characteristics are:

Purpose

This register contains the current monitor mode.

If MGI_FEAT1.MODE_REG < 0b010, this register is RES0.

This width of this register is MODE_LEN, unused bits are RES0.

For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_STAT1 is a 32-bit register.

Field descriptions

The MGI_MODE_STAT1 bit assignments are:

MODE_STAT, bits [31:0]

Monitor Mode Status

Indication of the current monitor mode.

This can be different from the requested monitor mode if the requested mode state is not yet updated at the monitor.

This field resets to `0x00000000`.

Accessing the MGI_MODE_STAT1

MGI_MODE_STAT1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0C8

Access on this interface is **RO**.

6.1.18 MGI_MODE_STAT2, Monitor Mode Status Register 2

The MGI_MODE_STAT2 characteristics are:

Purpose

This register contains the current monitor mode.

If MGI_FEAT1.MODE_REG < 0b011, this register is RES0.

This width of this register is MODE_LEN, unused bits are RES0.

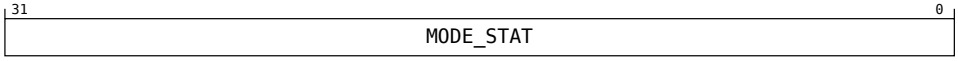
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_STAT2 is a 32-bit register.

Field descriptions

The MGI_MODE_STAT2 bit assignments are:



MODE_STAT, bits [31:0]

Monitor Mode Status

Indication of the current monitor mode.

This can be different from the requested monitor mode if the requested mode state is not yet updated at the monitor.

This field resets to `0x00000000`.

Accessing the MGI_MODE_STAT2

MGI_MODE_STAT2 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0D0

Access on this interface is **RO**.

6.1.19 MGI_MODE_STAT3, Monitor Mode Status Register 3

The MGI_MODE_STAT3 characteristics are:

Purpose

This register contains the current monitor mode.

If MGI_FEAT1.MODE_REG < 0b100, this register is RES0.

This width of this register is MODE_LEN, unused bits are RES0.

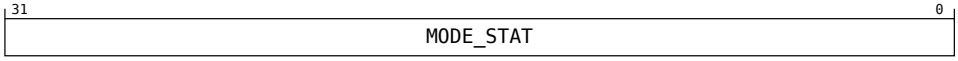
For more information on configuration see section 2.11 *Configuration options*.

Attributes

MGI_MODE_STAT3 is a 32-bit register.

Field descriptions

The MGI_MODE_STAT3 bit assignments are:



MODE_STAT, bits [31:0]

Monitor Mode Status

Indication of the current monitor mode.

This can be different from the requested monitor mode if the requested mode state is not yet updated at the monitor.

This field resets to `0x00000000`.

Accessing the MGI_MODE_STAT3

MGI_MODE_STAT3 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x0D8

Access on this interface is **RO**.

6.1.20 MGI_IRQ_STAT, Interrupt Status Register

The MGI_IRQ_STAT characteristics are:

Purpose

This register contains information on the status of interrupts and allows interrupt events to be cleared.

A read gives the interrupt event status.

A write of 0b1 to any bit clears that interrupt event status.

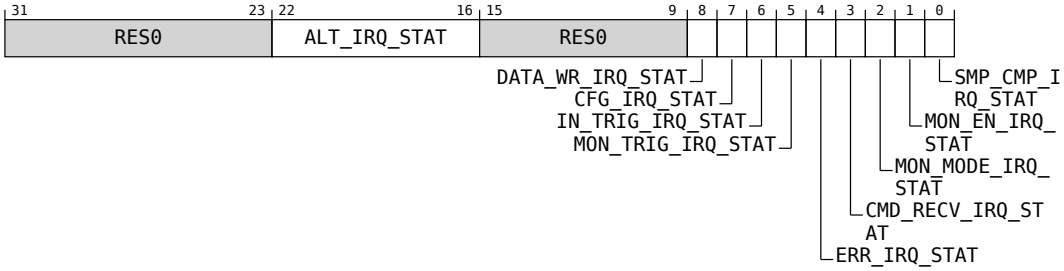
For details on the interrupt types see 2.10 *Monitor Group Interface interrupt events*.

Attributes

MGI_IRQ_STAT is a 32-bit register.

Field descriptions

The MGI_IRQ_STAT bit assignments are:



Bits [31:23]

Reserved, RES0.

ALT_IRQ_STAT, bits [22:16]

Alert Interrupt Event Status

There is an event status bit for each alert supported. The bit for alert n is, for n from 0 to 6, Alert n = ALT_IRQ_STAT[16+n]

For example, Alert 0 is bit 16 and Alert 1 is bit 17.

Bits for alerts that are not supported are RES0

The number of supported alerts is indicated in MGI_FEAT0.ALERT_NUM.

This field resets to `0b0000000`.

Bits [15:9]

Reserved, RES0.

DATA_WR_IRQ_STAT, bit [8]

Data Write Complete Interrupt Event Status.

Indicates the data write of a monitor data set has been completed by the DMA interface.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b0.

CFG_IRQ_STAT, bit [7]

Configuration Request Interrupt Event Status.

Indicates that configuration of the monitor group is required due to power on or reset deassertion.

This field resets to 0b0.

IN_TRIG_IRQ_STAT, bit [6]

Input Trigger Interrupt Event Status

Indicates that an input trigger has been received.

When MGI_FEAT0.TRIG_IN is 0b0 this field is RES0.

This field resets to 0b0.

MON_TRIG_IRQ_STAT, bit [5]

Monitor Trigger Interrupt Event Status.

Indicates that a monitor trigger has been received.

This field resets to 0b0.

ERR_IRQ_STAT, bit [4]

Error Interrupt Event Status.

Indication that an error has been received.

This field resets to 0b0.

CMD_RECV_IRQ_STAT, bit [3]

User Defined Command Received Interrupt Event Status.

Indication that an User Defined command has been received.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to 0b0.

MON_MODE_IRQ_STAT, bit [2]

Monitor Mode Request Complete Interrupt Event Status.

Indication that a set monitor mode action has been completed.

This field resets to 0b0.

MON_EN_IRQ_STAT, bit [1]

Monitor Enable Request Complete Interrupt Event Status.

Indication that a programmed monitor enable/disable action has been completed.

This field resets to 0b0.

SMP_CMP_IRQ_STAT, bit [0]

Monitor Sample Data Set Complete Interrupt Event Status.

Indication that sampling is complete for all monitors in the group.

This field resets to `0b0`.

Accessing the MGI_IRQ_STAT

MGI_IRQ_STAT can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x100

Access on this interface is **RW**.

6.1.21 MGI_IRQ_MASK, Interrupt Mask Register

The MGI_IRQ_MASK characteristics are:

Purpose

This register allows masking of interrupt events.

0b0 indicates the interrupt is enabled.

0b1 indicates the interrupt is masked.

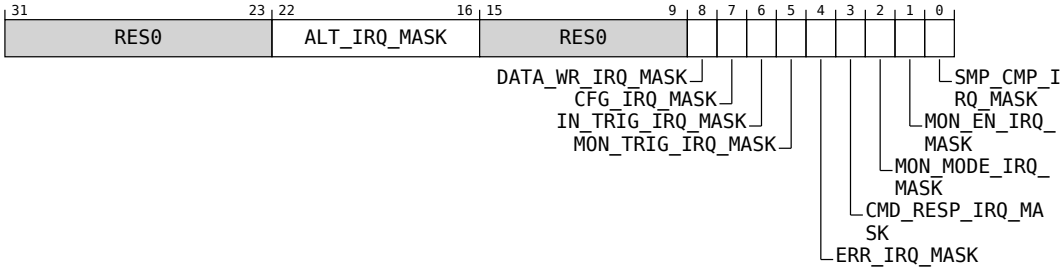
For details on the interrupt types see 2.10 *Monitor Group Interface interrupt events*.

Attributes

MGI_IRQ_MASK is a 32-bit register.

Field descriptions

The MGI_IRQ_MASK bit assignments are:



Bits [31:23]

Reserved, RES0.

ALT_IRQ_MASK, bits [22:16]

Alert Interrupt Events Mask

There is an event mask bit for each alert supported. The bit for alert n is, for n from 0 to 6, Alert n = ALT_IRQ_STAT[16+n]

For example, Alert 0 is bit 16 and Alert 1 is bit 17.

Bits for alerts that are not supported are RES0

The number of supported alerts is indicated in MGI_FEAT0.ALERT_NUM.

This field resets to `0b1111111`.

Bits [15:9]

Reserved, RES0.

DATA_WR_IRQ_MASK, bit [8]

Data Write Complete Interrupt Event Mask.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b1.

CFG_IRQ_MASK, bit [7]

Configuration Request Interrupt Event Mask.

This field is reset to the DEF_CFG_IRQ_MASK configuration value, see section 2.11 *Configuration options*.

This field resets to an IMPLEMENTATION DEFINED value.

IN_TRIG_IRQ_MASK, bit [6]

Input Trigger Interrupt Event Mask

When MGI_FEAT0.TRIG_IN is 0b0 this field is RES0.

This field resets to 0b1.

MON_TRIG_IRQ_MASK, bit [5]

Monitor Trigger Interrupt Event Mask.

This field resets to 0b1.

ERR_IRQ_MASK, bit [4]

Error Interrupt Event Mask.

This field resets to 0b1.

CMD_RESP_IRQ_MASK, bit [3]

User Defined Command Received Interrupt Event Mask.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0

This field resets to 0b1.

MON_MODE_IRQ_MASK, bit [2]

Monitor Mode Request Complete Interrupt Event Mask.

This field resets to 0b1.

MON_EN_IRQ_MASK, bit [1]

Monitor Enable Request Complete Interrupt Event Mask.

This field resets to 0b1.

SMP_CMP_IRQ_MASK, bit [0]

Monitor Sample Data Set Complete Interrupt Event Mask.

This field resets to 0b1.

Accessing the MGI_IRQ_MASK

MGI_IRQ_MASK can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x110

Access on this interface is **RW**.

6.1.22 MGI_TRG_MASK, Output Trigger Mask Register

The MGI_TRG_MASK characteristics are:

Purpose

This register allows masking of trigger events.

0b0 indicates the trigger is enabled.

0b1 indicates the trigger is masked.

For details on the interrupt types see 2.10 *Monitor Group Interface interrupt events*

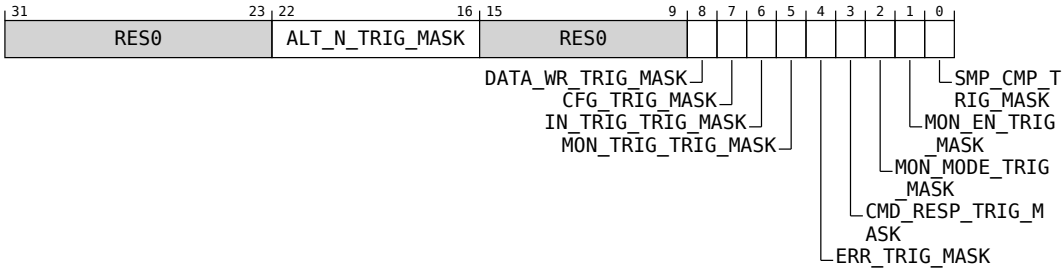
When MGI_FEAT0.TRIG_OUT is 0b0 this register is RES0.

Attributes

MGI_TRG_MASK is a 32-bit register.

Field descriptions

The MGI_TRG_MASK bit assignments are:



Bits [31:23]

Reserved, RES0.

ALT_N_TRIG_MASK, bits [22:16]

Alert Trigger Event Mask

There is an event for each alert supported.

For example Alert 0 is bit 7, Alert 1 is bit 8.

Bits for alerts which are not supported are reserved RES0

The number of supported alerts is indicated in MGI_FEAT0.ALERT_NUM.

This field resets to `0b1111111`.

Bits [15:9]

Reserved, RES0.

DATA_WR_TRIG_MASK, bit [8]

Data Write Complete Trigger Event Mask.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b1.

CFG_TRIG_MASK, bit [7]

Configuration Request Trigger Event Mask.

This field is reset to the DEF_CFG_TRIG_MASK configuration value, see 2.11 *Configuration options*

This field resets to an IMPLEMENTATION DEFINED value.

IN_TRIG_TRIG_MASK, bit [6]

'Input Trigger' Trigger Event Mask

When MGI_FEAT0.TRIG_IN is 0b0 this field is RES0.

This field resets to 0b1.

MON_TRIG_TRIG_MASK, bit [5]

'Monitor Trigger' Trigger Event Status.

This field resets to 0b1.

ERR_TRIG_MASK, bit [4]

Error Trigger Event Mask.

This field resets to 0b1.

CMD_RESP_TRIG_MASK, bit [3]

User Defined Command Response Trigger Event Mask.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to 0b1.

MON_MODE_TRIG_MASK, bit [2]

Monitor Mode Set Trigger Event Mask.

This field resets to 0b1.

MON_EN_TRIG_MASK, bit [1]

Monitor Enable Request Complete Trigger Event Mask.

This field resets to 0b1.

SMP_CMP_TRIG_MASK, bit [0]

Monitor Set Sample Complete Trigger Event Mask.

This field resets to 0b1.

Accessing the MGI_TRG_MASK

MGI_TRG_MASK can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x140

Access on this interface is **RW**.

6.1.23 MGI_ERR_CODE, Error Code Register

The MGI_ERR_CODE characteristics are:

Purpose

This register contains the error code received from a monitor error command.

The first monitor to report an error is recorded here.

While the Error interrupt event is active any subsequent other error conditions will not be reported, but the presence of additional errors is indicated by the SECOND_ERROR bit.

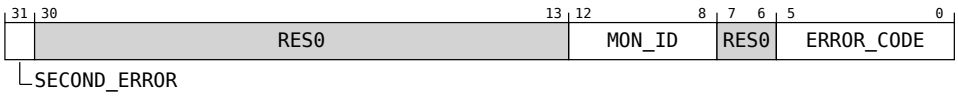
Once the interrupt event is cleared any subsequent error conditions will be reported.

Attributes

MGI_ERR_CODE is a 32-bit register.

Field descriptions

The MGI_ERR_CODE bit assignments are:



SECOND_ERROR, bit [31]

This bit set to 0b1 indicates further error(s) have occurred whilst a previously reported error interrupt event is still active.

The other fields in this register represent the first reported condition.

This field resets to `0b0`.

Bits [30:13]

Reserved, RES0.

MON_ID, bits [12:8]

The monitor number, if applicable, that returned the error.

This field resets to `0b00000`.

Bits [7:6]

Reserved, RES0.

ERROR_CODE, bits [5:0]

The reported error code.

This field resets to `0b000000`.

Accessing the MGI_ERR_CODE

MGI_ERR_CODE can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x150

Access on this interface is **RW**.

6.1.24 MGI_WREN, Data Write Enable Register

The MGI_WREN characteristics are:

Purpose

This register contains controls to enable the writing out of monitor sample data sets with the DMA interface.

When MGI_FEAT0.DMA_IF is 0b0 this register is RES0.

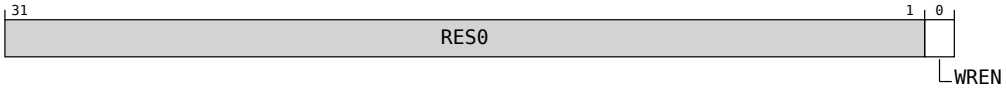
For more information see 2.8 *Direct Memory Access interface*.

Attributes

MGI_WREN is a 32-bit register.

Field descriptions

The MGI_WREN bit assignments are:



Bits [31:1]

Reserved, RES0.

WREN, bit [0]

Data Write Enable.

Enables monitor sample data set writing as configured in the Data Write Configuration Register.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

Value	Meaning
0b0	Data writing disabled.
0b1	Data writing enabled.

This field resets to 0b0.

Accessing the MGI_WREN

MGI_WREN can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x160

Access on this interface is **RW**.

6.1.25 MGI_WRCFG, Data Write Configuration Register

The MGI_WRCFG characteristics are:

Purpose

This register configures the writing out of monitor sample data sets with the DMA interface.

When MGI_FEAT0.DMA_IF is 0b0 this register is RES0.

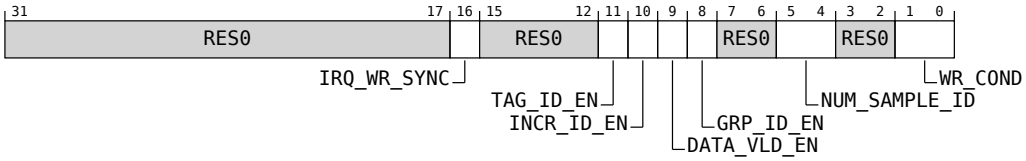
For more information see section 2.8 *Direct Memory Access interface*.

Attributes

MGI_WRCFG is a 32-bit register.

Field descriptions

The MGI_WRCFG bit assignments are:



Bits [31:17]

Reserved, RES0.

IRQ_WR_SYNC, bit [16]

Synchronize the Alert Interrupt Event with the DMA Write Complete Interrupt Event.

This bit configures if Alert interrupt events occur when the monitor sample data set is complete in the MGI, or if they are delayed until the DMA interface monitor sample data set write is complete.

Value	Meaning
0b0	Alert interrupt events occur when the monitor sample data set is complete in the MGI.
0b1	Alert interrupt events occur when the monitor sample data set is complete in the MGI and it has been written to a memory-mapped location.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

Values not specified are reserved.

This field resets to 0b0.

Bits [15:12]

Reserved, RES0.

TAG_ID_EN, bit [11]

Tag Sample ID Enable.

This bit configures if the Tag Sample ID is written with the Sample ID.

Value	Meaning
0b0	The tag sample ID is not written.
0b1	The tag sample ID is written.

If the tag input is not supported, MGI_FEAT0.TAG_IN is 0b0, then this value is RES0. See 2.11.2.4 *Tag input*

If MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b0.

INCR_ID_EN, bit [10]

Incrementing Count Sample ID Enable.

This bit configures if the Incrementing Sample ID is written with the Sample ID.

Value	Meaning
0b0	The incrementing count sample ID is not written.
0b1	The incrementing count sample ID is written.

If MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b0.

DATA_VLD_EN, bit [9]

Data Valid Enable.

This bit configures if the Data Valid bits, MGI_DVLD, are written with the monitor sample data set.

Value	Meaning
0b0	The Data Valid bits are not written.
0b1	The Data Valid bits are written.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b0.

GRP_ID_EN, bit [8]

Group ID Enable.

This bit configures if the Group ID, MGI_GRP_ID.GRP_ID, is written with the monitor sample data set.

Value	Meaning
0b0	The Group ID is not written.
0b1	The Group ID is written.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to 0b0.

Bits [7:6]

Reserved, RES0.

NUM_SAMPLE_ID, bits [5:4]

Number of Sample Identifiers written.

This field configures which of the start and end Sample Identifiers are written with the monitor sample data set.

The type of Sample Identifier is configured in MGI_WRCFG.INCR_ID_EN and MGI_WRCFG.TAG_ID_EN.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

Value	Meaning
0b00	Sample identifiers are not written.
0b01	Start sample identifiers are written.
0b10	Start and end sample identifiers are written.

Values not specified are reserved.

This field resets to 0b00.

Bits [3:2]

Reserved, RES0.

WR_COND, bits [1:0]

Write Condition.

Configures when a monitor sample data set write occurs.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

Value	Meaning
0b00	Data is written when every monitor sample data set is complete.
0b01	Data is written once when an alert is triggered.
0b10	Data writing starts when an alert is triggered and continues until data writing is disabled.

Values not specified are reserved.

This field resets to `0b00`.

Accessing the MGI_WRCFG

MGI_WRCFG can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x168

Access on this interface is **RW**.

6.1.26 MGI_WADDR0, Data Write Address Register 0

The MGI_WADDR0 characteristics are:

Purpose

This register contains the lower 32 bits of the address where monitor data is to be written.

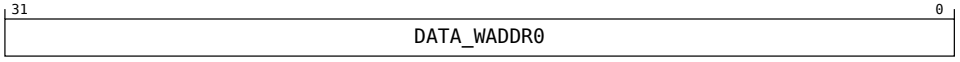
When MGI_FEAT0.DMA_IF is 0b0 this register is RES0.

Attributes

MGI_WADDR0 is a 32-bit register.

Field descriptions

The MGI_WADDR0 bit assignments are:



DATA_WADDR0, bits [31:0]

Monitor data set write address lower 32 bits.

This field is reset to the DEF_WADDR_CFG[31:0] configuration value, see section 2.11 *Configuration options*.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_WADDR0

MGI_WADDR0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x170

Access on this interface is **RW**.

6.1.27 MGI_WADDR1, Data Write Address Register 1

The MGI_WADDR1 characteristics are:

Purpose

This register contains the upper 32 bits of the address where monitor data is to be written.

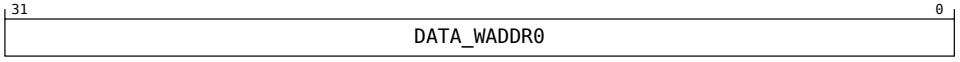
When MGI_FEAT0.DMA_IF is 0b0 this register is RES0.

Attributes

MGI_WADDR1 is a 32-bit register.

Field descriptions

The MGI_WADDR1 bit assignments are:



DATA_WADDR0, bits [31:0]

Monitor data set write address upper 32 bits.

This field is reset to the DEF_WADDR_CFG[63:32] configuration value, see section 2.11 *Configuration options*.

When MGI_FEAT0.DMA_IF is 0b0 this field is RES0.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_WADDR1

MGI_WADDR1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x174

Access on this interface is **RW**.

6.1.28 MGI_RADDR0, Data Read Address Register 0

The MGI_RADDR0 characteristics are:

Purpose

This register contains the lower 32-bits of the address where monitor data is read from if MGI_DATA_INFO.ALT_ADDR is 0b1.

When MGI_DATA_INFO.A is 0b0 this register is RES0.

Attributes

MGI_RADDR0 is a 32-bit register.

Field descriptions

The MGI_RADDR0 bit assignments are:

DATA_RADDR0, bits [31:0]

Monitor data alternate read address lower 32-bits.

This field is reset to the DEF_RADDR_CFG[31:0] configuration value, see 2.11 *Configuration options*.

When MGI_DATA_INFO.A is 0b0 this field is RES0.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_RADDR0

MGI_RADDR0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x180

Access on this interface is **RO**.

6.1.29 MGI_RADDR1, Data Read Address Register 1

The MGI_RADDR1 characteristics are:

Purpose

This register contains the upper 32 bits of the address where monitor data is read from if MGI_DATA_INFO.ALT_ADDR is 0b1.

When MGI_DATA_INFO.A is 0b0 this register is RES0.

Attributes

MGI_RADDR1 is a 32-bit register.

Field descriptions

The MGI_RADDR1 bit assignments are:

DATA_RADDR1, bits [31:0]

Monitor data alternate read address upper 32-bits.

This field is reset to the DEF_RADDR_CFG[63:32] configuration value, see 2.11 *Configuration options*.

When MGI_DATA_INFO.A is 0b0 this field is RES0.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_RADDR1

MGI_RADDR1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x184

Access on this interface is **RO**.

6.1.30 MGI_DISCON_ID, Monitor Disconnection Identification Register

The MGI_DISCON_ID characteristics are:

Purpose

This register contains information on which monitors support being disconnected.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this registers width is [5:0].

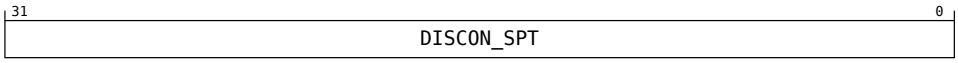
Register bits higher than the number of supported monitors, as defined in MGI_GRP_ID.MON_NUM, are RES0.

Attributes

MGI_DISCON_ID is a 32-bit register.

Field descriptions

The MGI_DISCON_ID bit assignments are:



DISCON_SPT, bits [31:0]

Monitor Disconnection Support.

Each bit represents a different monitor. For example bit 0 represents monitor 0, and bit 3 represents monitor 3.

The bit width of this field is MGI_GRP_ID.MON_NUM + 1.

Each bit value is either:

0b0: The monitor represented by this bit position does not support being disconnected, and is always connected.

0b1: The monitor represented by this bit position does support being disconnected.

The register is reset to the value of MON_DISCON_CFG.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_DISCON_ID

MGI_DISCON_ID can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x190

Access on this interface is **RO**.

6.1.31 MGI_CON_STAT, Monitor Connection Status Register

The MGI_CON_STAT characteristics are:

Purpose

This register contains information on which monitors are currently connected.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this register width is [5:0].

Register bits higher than the number of supported monitors as defined in MGI_GRP_ID.MON_NUM are RES0.

Attributes

MGI_CON_STAT is a 32-bit register.

Field descriptions

The MGI_CON_STAT bit assignments are:

CON_STAT, bits [31:0]

Monitor Connection Status.

Each bit represents a different monitor. For example, bit 0 represents monitor 0, and bit 3 represents monitor 3.

The width of this field is MGI_GRP_ID.MON_NUM + 1.

Each bit value is either:

0b0: The monitor represented by this bit position is disconnected.

0b1: The monitor represented by this bit position is connected.

Each bit for a monitor that supports connection/disconnection is reset to disconnected.

Each bit for a monitor that does not support connection/disconnection is reset to connected.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_CON_STAT

MGI_CON_STAT can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x198

Access on this interface is **RO**.

6.1.32 MGI_CMD_SEND0, Command Send Register 0

The MGI_CMD_SEND0 characteristics are:

Purpose

This register allows sending of User Defined commands to monitors.

The command type is always set to User Defined for commands sent using this register.

The command is sent because of the write to this register.

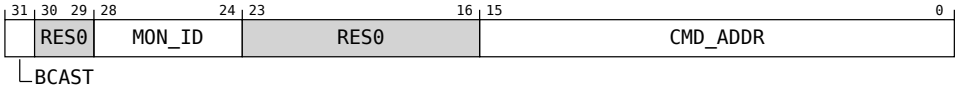
When MGI_FEAT0.USER_DEF_CMD is 0b0 this register is RES0.

Attributes

MGI_CMD_SEND0 is a 32-bit register.

Field descriptions

The MGI_CMD_SEND0 bit assignments are:



BCAST, bit [31]

Broadcast bit.

Indicates if the command is sent to a single monitor or broadcast to all monitors.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

Value	Meaning
0b0	The command is only sent to the monitor indicated in MON_ID.
0b1	The command is sent to all monitors.

This field resets to 0b0.

Bits [30:29]

Reserved, RES0.

MON_ID, bits [28:24]

Monitor ID.

Indicates the monitor to which this command is to be sent.

This field is ignored if BCAST is set to 0b1.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0

This field resets to 0b00000.

Bits [23:16]

Reserved, RES0.

CMD_ADDR, bits [15:0]

Command Address.

Indicates the value to be put in the command address/sub-ID field.

This is typically used to indicate the operation of the User Defined command.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to 0x0000.

Accessing the MGI_CMD_SEND0

MGI_CMD_SEND0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x1B0

Access on this interface is **RW**.

6.1.33 MGI_CMD_SEND1, Command Send Register 1

The MGI_CMD_SEND1 characteristics are:

Purpose

This register contains the data for User Defined commands sent to the monitors with MGI_CMD_SEND0.

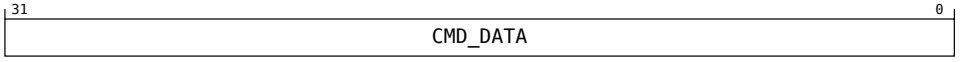
When MGI_FEAT0.USER_DEF_CMD is 0b0 this register is RES0.

Attributes

MGI_CMD_SEND1 is a 32-bit register.

Field descriptions

The MGI_CMD_SEND1 bit assignments are:



CMD_DATA, bits [31:0]

Data sent with the User Defined command.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this register is RES0.

This field resets to `0x00000000`.

Accessing the MGI_CMD_SEND1

MGI_CMD_SEND1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x1B8

Access on this interface is **RW**.

6.1.34 MGI_CMD_RECV0, Command Receive Register 0

The MGI_CMD_RECV0 characteristics are:

Purpose

This register contains the command details of User Defined commands received from the monitors.

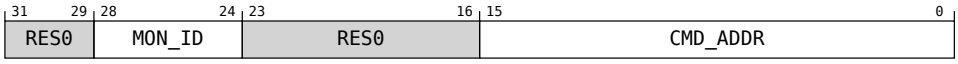
When MGI_FEAT0.USER_DEF_CMD is 0b0 this register is RES0.

Attributes

MGI_CMD_RECV0 is a 32-bit register.

Field descriptions

The MGI_CMD_RECV0 bit assignments are:



Bits [31:29]

Reserved, RES0.

MON_ID, bits [28:24]

Monitor ID.

Indicates the monitor from which this command was received.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to `0b00000`.

Bits [23:16]

Reserved, RES0.

CMD_ADDR, bits [15:0]

Command Address.

The value in the command address/sub-ID field of the received User Defined command.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to `0x0000`.

Accessing the MGI_CMD_RECV0

MGI_CMD_RECV0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x1C0

Access on this interface is **RW**.

6.1.35 MGI_CMD_RECV1, Command Receive Register 1

The MGI_CMD_RECV1 characteristics are:

Purpose

This register contains the data of User Defined commands received from the monitors.

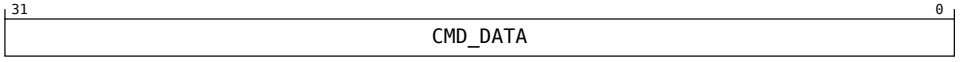
When MGI_FEAT0.USER_DEF_CMD is 0b0 this register is RES0.

Attributes

MGI_CMD_RECV1 is a 32-bit register.

Field descriptions

The MGI_CMD_RECV1 bit assignments are:



CMD_DATA, bits [31:0]

Command data received from the User Defined command.

When MGI_FEAT0.USER_DEF_CMD is 0b0 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_CMD_RECV1

MGI_CMD_RECV1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x1C8

Access on this interface is **RW**.

6.1.36 MGI_ATYP0, Alert 0 Type Register

The MGI_ATYP0 characteristics are:

Purpose

This register controls the trigger type for alert 0.

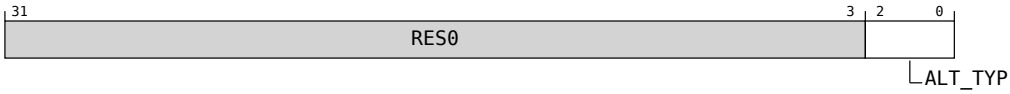
If MGI_FEAT0.ALERT_NUM < 0b001, this register is RES0.

Attributes

MGI_ATYP0 is a 32-bit register.

Field descriptions

The MGI_ATYP0 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL0. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL0. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP0

MGI_ATYP0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x200

Access on this interface is **RW**.

6.1.37 MGI_AVAL0, Alert 0 Value Register

The MGI_AVAL0 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b001, this register is RES0.

Attributes

MGI_AVAL0 is a 32-bit register.

Field descriptions

The MGI_AVAL0 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL0

MGI_AVAL0 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x208

Access on this interface is **RW**.

6.1.38 MGI_ATYP1, Alert 1 Type Register

The MGI_ATYP1 characteristics are:

Purpose

This register controls the trigger type for alert 1.

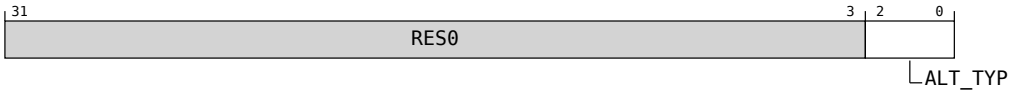
If MGI_FEAT0.ALERT_NUM < 0b010, this register is RES0.

Attributes

MGI_ATYP1 is a 32-bit register.

Field descriptions

The MGI_ATYP1 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL1. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL1. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP1

MGI_ATYP1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x210

Access on this interface is **RW**.

6.1.39 MGI_AVAL1, Alert 1 Value Register

The MGI_AVAL1 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b010, this register is RES0.

Attributes

MGI_AVAL1 is a 32-bit register.

Field descriptions

The MGI_AVAL1 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL1

MGI_AVAL1 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x218

Access on this interface is **RW**.

6.1.40 MGI_ATYP2, Alert 2 Type Register

The MGI_ATYP2 characteristics are:

Purpose

This register controls the trigger type for alert 2.

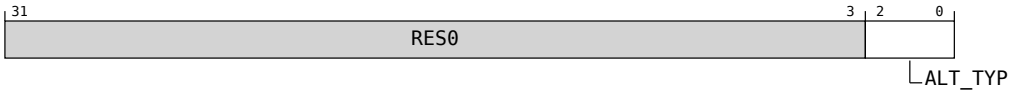
If MGI_FEAT0.ALERT_NUM < 0b011, this register is RES0.

Attributes

MGI_ATYP2 is a 32-bit register.

Field descriptions

The MGI_ATYP2 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL2. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL2. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP2

MGI_ATYP2 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x220

Access on this interface is **RW**.

6.1.41 MGI_AVAL2, Alert 2 Value Register

The MGI_AVAL2 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b011, this register is RES0.

Attributes

MGI_AVAL2 is a 32-bit register.

Field descriptions

The MGI_AVAL2 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL2

MGI_AVAL2 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x228

Access on this interface is **RW**.

6.1.42 MGI_ATYP3, Alert 3 Type Register

The MGI_ATYP3 characteristics are:

Purpose

This register controls the trigger type for alert 3.

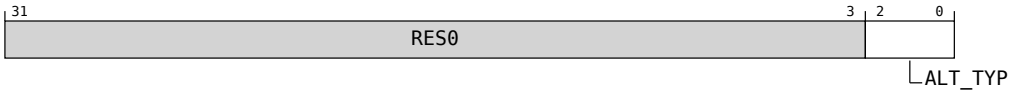
If MGI_FEAT0.ALERT_NUM < 0b100, this register is RES0.

Attributes

MGI_ATYP3 is a 32-bit register.

Field descriptions

The MGI_ATYP3 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL3. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL3. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP3

MGI_ATYP3 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x230

Access on this interface is **RW**.

6.1.43 MGI_AVAL3, Alert 3 Value Register

The MGI_AVAL3 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b100, this register is RES0.

Attributes

MGI_AVAL3 is a 32-bit register.

Field descriptions

The MGI_AVAL3 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL3

MGI_AVAL3 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x238

Access on this interface is **RW**.

6.1.44 MGI_ATYP4, Alert 4 Type Register

The MGI_ATYP4 characteristics are:

Purpose

This register controls the trigger type for alert 4.

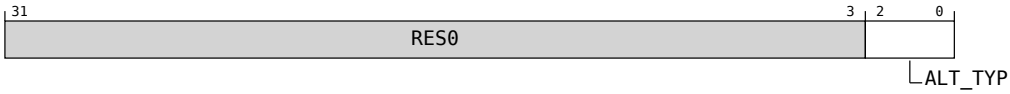
If MGI_FEAT0.ALERT_NUM < 0b101, this register is RES0.

Attributes

MGI_ATYP4 is a 32-bit register.

Field descriptions

The MGI_ATYP4 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL4. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL4. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP4

MGI_ATYP4 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x240

Access on this interface is **RW**.

6.1.45 MGI_AVAL4, Alert 4 Value Register

The MGI_AVAL4 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b101, this register is RES0.

Attributes

MGI_AVAL4 is a 32-bit register.

Field descriptions

The MGI_AVAL4 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL4

MGI_AVAL4 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x248

Access on this interface is **RW**.

6.1.46 MGI_ATYP5, Alert 5 Type Register

The MGI_ATYP5 characteristics are:

Purpose

This register controls the trigger type for alert 5.

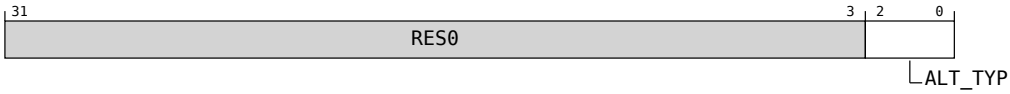
If MGI_FEAT0.ALERT_NUM < 0b110, this register is RES0.

Attributes

MGI_ATYP5 is a 32-bit register.

Field descriptions

The MGI_ATYP5 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL5. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL5. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP5

MGI_ATYP5 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x250

Access on this interface is **RW**.

6.1.47 MGI_AVAL5, Alert 5 Value Register

The MGI_AVAL5 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b110, this register is RES0.

Attributes

MGI_AVAL5 is a 32-bit register.

Field descriptions

The MGI_AVAL5 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL5

MGI_AVAL5 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x258

Access on this interface is **RW**.

6.1.48 MGI_ATYP6, Alert 6 Type Register

The MGI_ATYP6 characteristics are:

Purpose

This register controls the trigger type for alert 6.

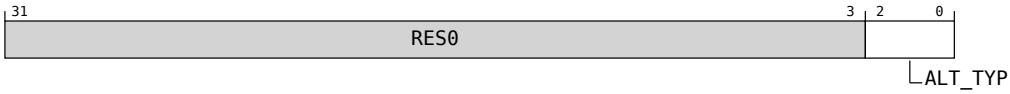
If MGI_FEAT0.ALERT_NUM < 0b111, this register is RES0.

Attributes

MGI_ATYP6 is a 32-bit register.

Field descriptions

The MGI_ATYP6 bit assignments are:



Bits [31:3]

Reserved, RES0.

ALT_TYP, bits [2:0]

Alert Type.

Value	Meaning
0b000	No alert - Alert is disabled.
0b001	Upper threshold - Alert triggered when any monitor sample is above the programmed value.
0b010	Lower threshold - Alert triggered when any monitor sample is above the programmed value.
0b101	Rising delta - Alert triggered when any monitor sample is higher than its previous sample by the value programmed in MGI_AVAL6. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.
0b110	Falling delta - Alert triggered when any monitor sample is lower than its previous sample by the value programmed in MGI_AVAL6. If the alert delta functions are not supported, MGI_FEAT0.ALT_DELTA is 0b0, then this value is reserved.

Values not listed are reserved.

This field resets to `0b000`.

Accessing the MGI_ATYP6

MGI_ATYP6 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x260

Access on this interface is **RW**.

6.1.49 MGI_AVAL6, Alert 6 Value Register

The MGI_AVAL6 characteristics are:

Purpose

This register programs the value for alert triggering.

If MGI_FEAT0.ALERT_NUM < 0b111, this register is RES0.

Attributes

MGI_AVAL6 is a 32-bit register.

Field descriptions

The MGI_AVAL6 bit assignments are:

ALT_VAL, bits [31:0]

Alert Value.

The meaning of the alert value depends on the alert type specified.

This field resets to `0x00000000`.

Accessing the MGI_AVAL6

MGI_AVAL6 can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x268

Access on this interface is **RW**.

6.1.50 MGI_DATA, Data Register <n>

The MGI_DATA characteristics are:

Purpose

This set of registers contains the monitor sample data set.

If MGI_DATA_INFO.A is 0b1 this register is RES0.

This registers width and format is determined by the data width of the monitors, and if the data is packed.

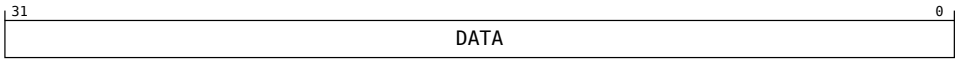
This number of monitor data registers depends upon the number of monitors, the width of the monitor data and if the data is packed. For more information see Chapter 4 *Data formats*.

Attributes

MGI_DATA is a 32-bit register.

Field descriptions

The MGI_DATA bit assignments are:



DATA, bits [31:0]

Monitor Sample Data.

This registers width and format is determined by the data width of the monitors, and if the data is packed. For more information see Chapter 4 *Data formats*.

This field resets to `0x00000000`.

Accessing the MGI_DATA

MGI_DATA can be accessed through the memory-mapped interface:

Component	Offset
MGI	0x700

Access on this interface is **RW**.

6.1.51 MGI_DVLD, Data Valid Register

The MGI_DVLD characteristics are:

Purpose

This register contains information on which of the data registers contain valid data.

This registers bit width is the number of monitors supported by the MGI.

For example, when the number of monitors is 6 this registers width is [5:0].

Register bits higher than the number of supported monitors as defined in MGI_GRP_ID.MON_NUM are RES0.

Attributes

MGI_DVLD is a 32-bit register.

Field descriptions

The MGI_DVLD bit assignments are:

DATA_VLD, bits [31:0]

Data Valid.

Each bit represents a monitor. For example, bit 0 represents monitor 0, and bit 3 represents monitor 3.

The bit width of this field is MGI_GRP_ID.MON_NUM + 1.

The data valid values are cleared to 0b0 when MGI_SMPID_START is updated.

This field resets to `0x00000000`.

Accessing the MGI_DVLD

MGI_DVLD can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF00

Access on this interface is **RO**.

6.1.52 MGI_TAG0_START, Tag Start Register 0

The MGI_TAG0_START characteristics are:

Purpose

This register contains bits [31:0] of the Start Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

Attributes

MGI_TAG0_START is a 32-bit register.

Field descriptions

The MGI_TAG0_START bit assignments are:

START_TAG_31_0, bits [31:0]

Start Sample ID tag value bits 31:0.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG0_START

MGI_TAG0_START can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF10

Access on this interface is **RO**.

6.1.53 MGI_TAG1_START, Tag Start Register 1

The MGI_TAG1_START characteristics are:

Purpose

This register contains bits [63:32] of the Start Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

When MGI_FEAT0.TAG_LEN is < 32 this register is RES0.

Attributes

MGI_TAG1_START is a 32-bit register.

Field descriptions

The MGI_TAG1_START bit assignments are:

START_TAG_63_32, bits [31:0]

Start Sample ID tag value bits 63:32.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 32 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG1_START

MGI_TAG1_START can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF14

Access on this interface is **RO**.

6.1.54 MGI_TAG2_START, Tag Start Register 2

The MGI_TAG2_START characteristics are:

Purpose

This register contains bits [95:64] of the Start Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

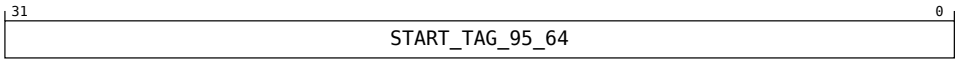
When MGI_FEAT0.TAG_LEN is < 64 this register is RES0.

Attributes

MGI_TAG2_START is a 32-bit register.

Field descriptions

The MGI_TAG2_START bit assignments are:



START_TAG_95_64, bits [31:0]

Start Sample ID tag value bits 95:64.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 64 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG2_START

MGI_TAG2_START can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF28

Access on this interface is **RO**.

6.1.55 MGI_TAG3_START, Tag Start Register 3

The MGI_TAG3_START characteristics are:

Purpose

This register contains bits [128:96] of the Start Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

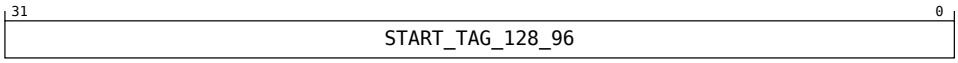
When MGI_FEAT0.TAG_LEN is < 96 this register is RES0.

Attributes

MGI_TAG3_START is a 32-bit register.

Field descriptions

The MGI_TAG3_START bit assignments are:



START_TAG_128_96, bits [31:0]

Start Sample ID tag value bits 128:96.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 96 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG3_START

MGI_TAG3_START can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF1C

Access on this interface is **RO**.

6.1.56 MGI_TAG0_END, Tag End Register 0

The MGI_TAG0_END characteristics are:

Purpose

This register contains bits [31:0] of the End Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

Attributes

MGI_TAG0_END is a 32-bit register.

Field descriptions

The MGI_TAG0_END bit assignments are:

END_TAG_31_0, bits [31:0]

End Sample ID tag value bits 31:0.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG0_END

MGI_TAG0_END can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF20

Access on this interface is **RO**.

6.1.57 MGI_TAG1_END, Tag End Register 1

The MGI_TAG1_END characteristics are:

Purpose

This register contains bits [63:32] of the End Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

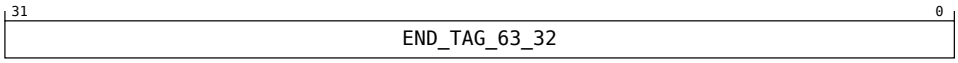
When MGI_FEAT0.TAG_LEN is < 32 this register is RES0.

Attributes

MGI_TAG1_END is a 32-bit register.

Field descriptions

The MGI_TAG1_END bit assignments are:



END_TAG_63_32, bits [31:0]

End Sample ID tag value bits 63:32.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 32 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG1_END

MGI_TAG1_END can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF24

Access on this interface is **RO**.

6.1.58 MGI_TAG2_END, Tag End Register 2

The MGI_TAG2_END characteristics are:

Purpose

This register contains bits [95:64] of the End Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

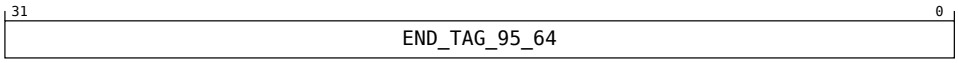
When MGI_FEAT0.TAG_LEN is < 64 this register is RES0.

Attributes

MGI_TAG2_END is a 32-bit register.

Field descriptions

The MGI_TAG2_END bit assignments are:



END_TAG_95_64, bits [31:0]

End Sample ID tag value bits 95:64.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 64 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG2_END

MGI_TAG2_END can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF28

Access on this interface is **RO**.

6.1.59 MGI_TAG3_END, Tag End Register 3

The MGI_TAG3_END characteristics are:

Purpose

This register contains bits [128:96] of the End Sample ID tag value.

When MGI_FEAT0.TAG_IN is 0b0 this register is RES0.

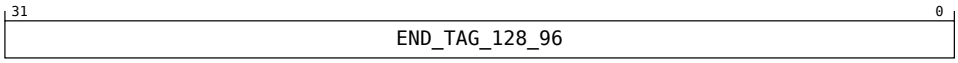
When MGI_FEAT0.TAG_LEN is < 96 this register is RES0.

Attributes

MGI_TAG3_END is a 32-bit register.

Field descriptions

The MGI_TAG3_END bit assignments are:



END_TAG_128_96, bits [31:0]

End Sample ID tag value bits 128:96.

When MGI_FEAT0.TAG_IN is 0b0 this field is RES0.

When MGI_FEAT0.TAG_LEN is < 96 this field is RES0.

This field resets to `0x00000000`.

Accessing the MGI_TAG3_END

MGI_TAG3_END can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF2C

Access on this interface is **RO**.

6.1.60 MGI_SMPID_START, Sample Identifier Start Register

The MGI_SMPID_START characteristics are:

Purpose

This register contains the monitor sample ID updated immediately before the first value of the monitor sample data set is updated in the Data Registers.

Attributes

MGI_SMPID_START is a 32-bit register.

Field descriptions

The MGI_SMPID_START bit assignments are:

SMPID_START, bits [31:0]

Monitor Sample Start ID.

This field contains the monitor sample ID updated immediately before the first value of the monitor sample data set is written to MGI_DATA<n>.

For more information on the format of this register, see section 4.2.1 *Sample identifier format*

This field resets to `0x00000000`.

Accessing the MGI_SMPID_START

MGI_SMPID_START can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF40

Access on this interface is **RO**.

6.1.61 MGI_SMPID_END, Sample Identifier End Register

The MGI_SMPID_END characteristics are:

Purpose

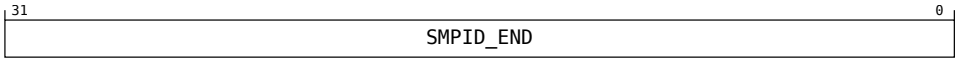
This register contains the monitor sample ID updated immediately after the last value of the monitor sample data set is updated in the Data Registers.

Attributes

MGI_SMPID_END is a 32-bit register.

Field descriptions

The MGI_SMPID_END bit assignments are:



SMPID_END, bits [31:0]

Monitor Sample End ID.

This field contains the monitor sample ID updated immediately after the last value of the monitor sample data set is updated in the Data Registers.

For more information on the format of this register, see section 4.2.1 *Sample identifier format*

This field resets to `0x00000000`.

Accessing the MGI_SMPID_END

MGI_SMPID_END can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xF48

Access on this interface is **RO**.

6.1.62 MGI_IIDR, Implementation Identification Register

The MGI_IIDR characteristics are:

Purpose

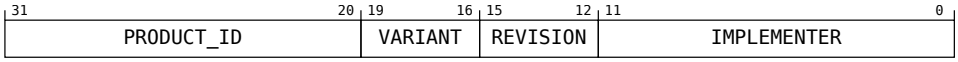
This register contains information about the implementation revision of the MGI.

Attributes

MGI_IIDR is a 32-bit register.

Field descriptions

The MGI_IIDR bit assignments are:



PRODUCT_ID, bits [31:20]

Product Identifier.

IMPLEMENTATION DEFINED value identifying the MGI part.

This field resets to an IMPLEMENTATION DEFINED value.

VARIANT, bits [19:16]

Implementation Variant.

IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product.

This field resets to an IMPLEMENTATION DEFINED value.

REVISION, bits [15:12]

Implementation Revision.

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product.

This field resets to an IMPLEMENTATION DEFINED value.

IMPLEMENTER, bits [11:0]

Implementer identification.

[11:8] The JEP106 continuation code of the implementer.

[7] Always 0.

[6:0] The JEP106 identity code of the implementer.

For an Arm implementation, bits [11:0] are 0x43B.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_IIDR

MGI_IIDR can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xFC0

Access on this interface is **RW**.

6.1.63 MGI_AIDR, Architecture Identification Register

The MGI_AIDR characteristics are:

Purpose

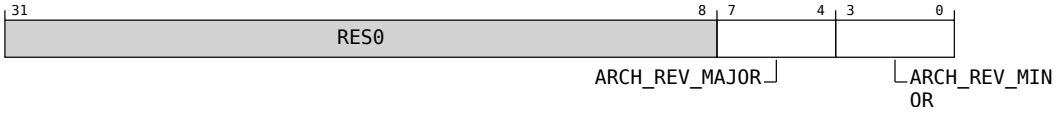
This register contains information about the implementer and the implementation of the MGI.

Attributes

MGI_AIDR is a 32-bit register.

Field descriptions

The MGI_AIDR bit assignments are:



Bits [31:8]

Reserved, RES0.

ARCH_REV_MAJOR, bits [7:4]

Architecture Major Revision.

- 0x1 – System Monitoring Control Framework Architecture Major Revision 1.
- Other Values – Reserved.

This field resets to an IMPLEMENTATION DEFINED value.

ARCH_REV_MINOR, bits [3:0]

Architecture Minor Revision.

- 0x0 – System Monitoring Control Framework Architecture Minor Revision 0.
- Other Values – Reserved.

This field resets to an IMPLEMENTATION DEFINED value.

Accessing the MGI_AIDR

MGI_AIDR can be accessed through the memory-mapped interface:

Component	Offset
MGI	0xFC8

Access on this interface is **RO**.

Part A
Appendixes

Chapter A1
Monitor Serial Interface

This section describes the recommended Monitor Serial Interface (MSI) protocol.

A1.1 Introduction

The Monitor Serial Interface (MSI) is a simple synchronous unidirectional serial protocol, intended to reduce the routing requirement between units, for example a Monitor Group Interface and a Monitor Local Interface. It is not intended to be a high speed or high bandwidth connection.

The protocol is based on the receiver detecting a Start Bit and then knowing, and counting, the number of data bits transmitted. The number of transmitted bits can either be fixed, or based upon IMPLEMENTATION DEFINED information within the transmitted MSI data.

The protocol can be pipelined to allow for crossing large distances.

A specification of the MSI data section for the System Monitoring Control Framework (SMCF) is in section 3.3 *Command format*.

A1.2 Signals

I This section describes the signals of the MSI and their functions.

Table A1.1 shows the MSI signals.

Table A1.1: MSI interface signal list

Name	Width	Source	Description
MSICLK	1	Clock Source	Clock
MSIDATA	1	Transmitter	Serial Data
MSIREADY	1	Receiver	Ready

A1.2.1 MSICLK

I **MSICLK** is the MSI clock. It is synchronous at the transmitter and receiver.

R All signals are sampled on the rising edge of **MSICLK**.

A1.2.2 MSIDATA

I **MSIDATA** is the MSI serial data.

R The serial data signal carries one bit of the data for each positive edge of *MSICLK*.

R When not transmitting **MSIDATA** is driven LOW.

A1.2.3 MSIREADY

I **MSIREADY** indicates that the receiver can accept a data bit transfer in the current clock cycle.

R The receiver asserts **MSIREADY** HIGH when it can accept a data bit on the interface.

R The receiver de-asserts **MSIREADY** LOW when it cannot accept a data bit on the interface.

A1.3 MSI packet format

I This section describes the format for packets sent on the MSI.

The packet consists of a Start Bit, a data section, and an *End Bit*.

Figure A1.1 shows the MSI packet format.

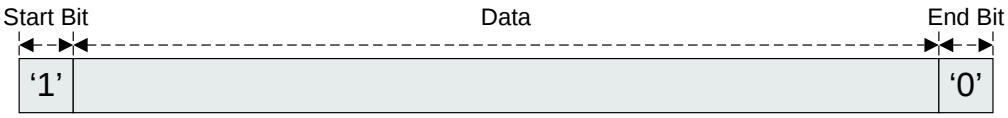


Figure A1.1: MSI packet format

A1.3.1 Start Bit

R A HIGH bit when there is not ongoing transfer indicates the start of the data packet.

A1.3.2 Data section

I This contains the data being transmitted.

R The data section minimum length is 8-bits.

R The data section maximum length is IMPLEMENTATION DEFINED.

R If the transmitter is using a variable length data section, the length of the data section must be able to be determined by the receiver from the first eight transmitted data bits.

U A transmitter and a receiver might have an agreed fixed data section length. The specification of this length is outside the specification of this protocol.

A1.3.3 End Bit

R A LOW End Bit following the data indicates the end of the packet to provide a separator between packets.

A1.4 MSI operation

The transmitter starts a transfer by putting the Start Bit on **MSIDATA**, it then places an additional data bit on **MSIDATA** for every **MSICLK** cycle that the receiver asserts **MSIREADY** HIGH until the data transfer is complete. It then puts the *End Bit* on **MSIDATA**.

Figure A1.2 shows an example basic MSI packet transfer.

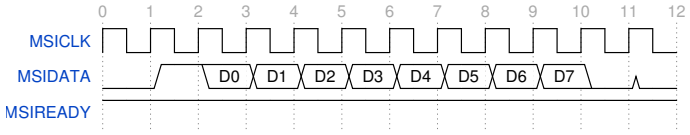


Figure A1.2: MSI operation

The sequence is:

- At t1 the transmitter sends the Start Bit by setting **MSIDATA** HIGH.
- At t2-t9, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t10 the transmitter sends the *End Bit* by setting **MSIDATA** LOW.
- At t11 the transmitter is not sending another transfer so keeps **MSIDATA** LOW.

In the **MSICLK** cycle immediately following the *End Bit* the transmitter can put another Start Bit on **MSIDATA** to send another packet.

Figure A1.3 shows consecutive MSI packet transfers.

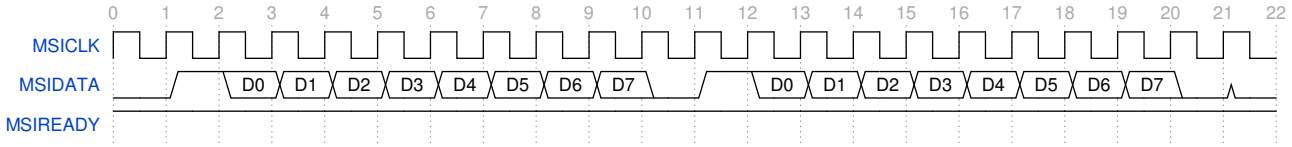


Figure A1.3: MSI operation with multiple packets

The sequence is:

- At t1 the transmitter sends the Start Bit by setting **MSIDATA** HIGH.
- At t2-t9, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t10 the transmitter sends the End Bit by setting **MSIDATA** LOW.
- At t11 the transmitter sends another packet so sends the Start Bit by setting **MSIDATA** HIGH.
- At t12-t19, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t20 the transmitter sends the End Bit by setting **MSIDATA** LOW.
- At t21 the transmitter is not sending another transfer so keeps **MSIDATA** LOW.

If a transfer is not being sent on the MSI, **MSIDATA** is set LOW.

When a transfer is not being sent the receiver can use the Start Bit as an indication that a transfer is starting. The receiver can de-assert **MSIREADY** before or at the start of the transfer to maintain this indication until it is ready to start accepting data.

Figure A1.4 shows an example MSI packet transfer delayed at the start due to the de-assertion of **MSIREADY**.

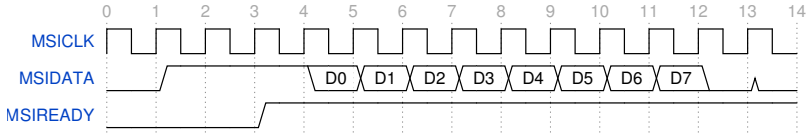


Figure A1.4: MSI initial delay by de-asserting MSIREADY

The sequence is:

- At t1 the transmitter sends the Start Bit by setting **MSIDATA** HIGH.
- At t2-t3, because **MSIREADY** is LOW the transmitter keeps the Start Bit on **MSIDATA**.
- At t4-t11, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t12 the transmitter sends the *End Bit* by setting **MSIDATA** LOW.
- At t13 the transmitter is not sending another transfer so keeps **MSIDATA** LOW.

When the receiver has seen the Start Bit, meaning that on a rising edge of **MSICLK** the Start Bit is on **MSIDATA** and **MSIREADY** is HIGH, it can use **MSIREADY** to pause the data mid-transfer. However, when the Start Bit has been accepted it cannot rely on any condition, like **MSIDATA** being HIGH when the **MSIREADY** is de-asserted, to indicate that a transfer is still ongoing. It is required to keep track of the length of the transfer, and amount of received data, for this purpose.

Figure A1.5 shows an example MSI packet transfer delayed mid transfer due to the de-assertion of **MSIREADY**.

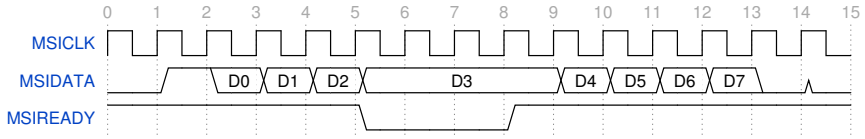


Figure A1.5: MSI mid transfer delay by de-asserting MSIREADY

The sequence is:

- At t1 the transmitter sends the Start Bit by setting **MSIDATA** HIGH.
- At t2-t5, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t6-t8, because **MSIREADY** is LOW the transmitter keeps D3 on **MSIDATA**.
- At t9-t12, because **MSIREADY** is HIGH the transmitter updates the data on **MSIDATA** every **MSICLK** rising edge.
- At t13 the transmitter sends the *End Bit* by setting **MSIDATA** LOW.
- At t14 the transmitter is not sending another transfer so keeps **MSIDATA** LOW.

If the transmitter puts the Start Bit on **MSIDATA** while **MSIREADY** is de-asserted LOW, it can be removed while the **MSIREADY** remains LOW. However, when the Start Bit has been seen at the receiver, when the Start Bit is on **MSIDATA** and **MSIREADY** is HIGH then the entire packet must be transferred.

Figure A1.6 shows an example MSI packet transfer being canceled.

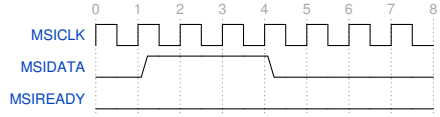


Figure A1.6: MSI transfer cancellation

The sequence is:

- At t1 the transmitter sends the Start Bit by setting **MSIDATA** HIGH.
- At t2-t3, because **MSIREADY** is LOW the transmitter keeps the Start Bit on **MSIDATA**.
- At t4, the transmitter wants to cancel the transfer, because **MSIREADY** has not yet been asserted for the transfer it can remove the Start Bit from **MSIDATA**.
- At t5 the transmitter is not sending another transfer so keeps **MSIDATA** LOW.

R If a transfer is ongoing and **MSIREADY** is de-asserted LOW the transmitter maintains the current value on **MSIDATA**.

R If a transfer is ongoing and **MSIREADY** is asserted HIGH the transmitter puts the next data bit or the *End Bit* on **MSIDATA**.

R If a transfer is not ongoing the transmitter can place the Start Bit on **MSIDATA** to start a transfer regardless of the state of **MSIREADY**.

R If the Start Bit is on **MSIDATA** it can be removed, and the transfer canceled, only if the Start Bit is on **MSIDATA** and **MSIREADY** is de-asserted LOW.

A1.5 MSI use with Monitor Group and Monitor Local Interfaces

To send data in both directions two interfaces are required.

Figure A1.7 shows the MSI connections between a group and local interface.

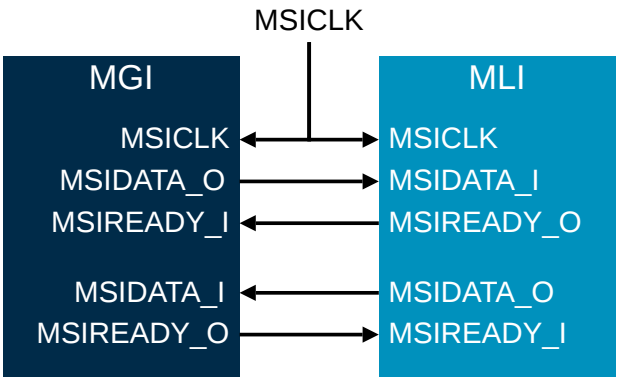


Figure A1.7: Monitor Group to Local Interface MSI connections

A1.6 Crossing asynchronous clock domains

I The protocol is designed to be sent and received with a synchronous clock. If an asynchronous domain boundary is required, the domain crossing must be done with a bridging component.

This bridging component receives the message data and then uses an IMPLEMENTATION DEFINED process to pass that across the clock domain boundary. It then transmits the data with the MSI protocol in the new clock domain. This must be done so that the message data can be sent one bit for each clock cycle, as required by the protocol, for the entire message in the new clock domain.

U The requirement to send the data one bit for each clock cycle in the new domain could imply the entire message must be transmitted across the clock domain boundary before re-transmission starts. This depends upon the method of crossing and the ratios of the clock domains.

I This type of bridge component must be able cope with the receipt of multiple subsequent messages without losing data, either with sufficient buffering or the use of **MSIREADY**.

U Because subsequent receivers can delay the receipt of message data with **MSIREADY** then this specification recommends that functionality to hold off incoming data with **MSIREADY** is implemented when buffering capacity is reached to prevent data loss.

Chapter A2
Use models

This section describes example use models for the System Monitoring Control Framework (SMCF).

A2.1 Introduction

The following section describes some example use cases and related control procedures for the SMCF. These are example use-cases for illustration and do not intend to provide a comprehensive list of potential uses.

A2.2 Power management data collection in a large core-count SoC example

In a large core-count SoC, the power management controller typically needs to collect a large amount of data relating to temperature, processor performance, and other related data. It needs to perform this on a constant and regular basis to make power management decisions. These include core DVFS points and other operating parameters related to core performance and thermal or power limits.

In this example, each core has a set of temperature sensors and a set of Activity Monitoring Units (AMUs) from which data needs to be collected.

Because there are many cores, this creates a large set of sensor and monitor data to be collected. If the power controller had to perform peripheral accesses to each monitor for sample control and data collection the absolute time taken for each access would accumulate. With increasing monitor count this becomes too large to be feasible, or even possible.

In this example, the SMCF performs the collection of this data and outputs it to a memory-mapped location that is efficient for the power controller to access. This removes the need for the power controller to perform many peripheral accesses.

The SMCF also alleviates issues of power management around having centralized data collection. For instance, if using a centralized data collector, for example a traditional DMA engine, coordination is required with the power management to ensure the collector does not access powered off cores. The SMCF alleviates this with its distributed nature allowing it to be placed inside or near power domains. Using the connection and disconnection mechanism to connect or reconnect sensors, based on the power management mechanisms which are orthogonal to the sensor and monitor management, then the coordination is managed by the SMCF.

The sample collector can initiate monitor sampling from local registers, or a shared timer, to generate a simultaneous trigger to each Monitor Group Interface (MGI). This enables samples on all enabled monitors with minimal skew between samples.

A2.2.1 Example structure

Each core has a set of temperature sensors that the central SoC power management must monitor to manage thermal limits. Additionally, it also monitors the core AMUs which provide data relating to processor activity. These two are arranged as separate groups. An additional group is included for any sensors within the core that require reading, but not on a periodic basis. This third group is not described further in this section. The structure for each core is shown in Figure A2.1.

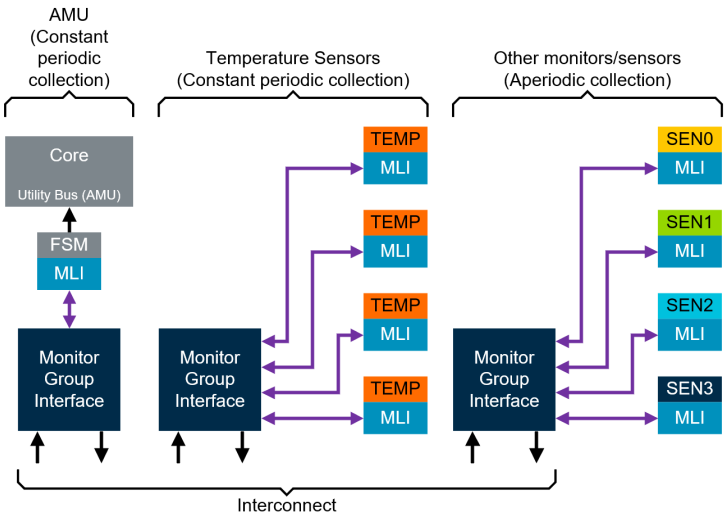


Figure A2.1: SMCF structure in each core

Figure A2.2 shows a simplified SoC structure.

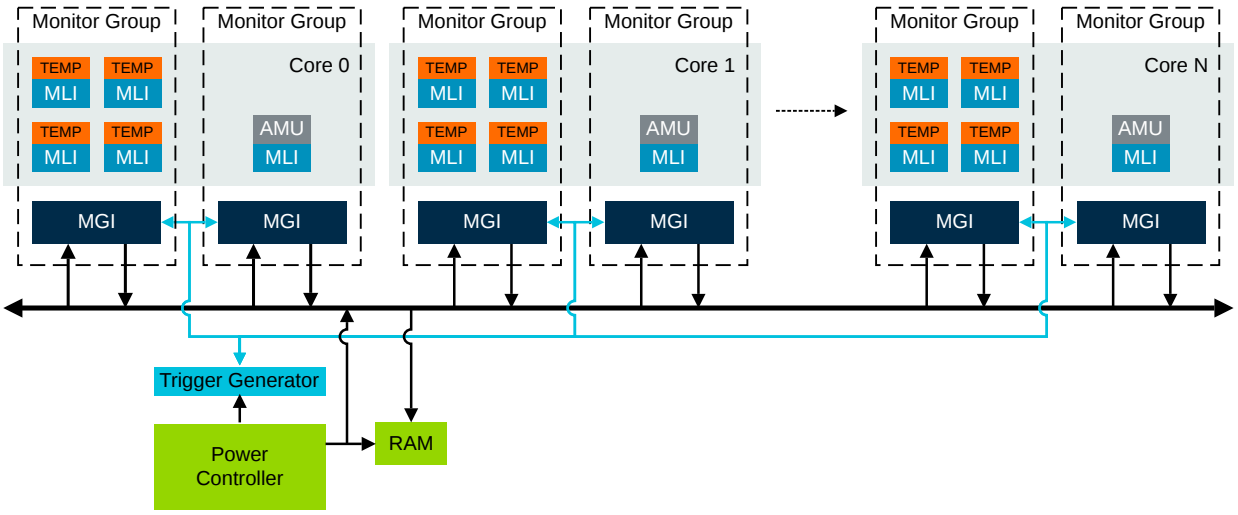


Figure A2.2: SoC SMCF structure

The trigger generator logic shown in purple indicates some required integration logic so the power controller can write to a single location to initiate an input trigger on all MGI interfaces simultaneously. It could also includes a status the power controller reads to determine when the action is complete.

An alternative to this would be to send an input trigger to one MGI, and then use the output trigger on the same MGI to send an input trigger to the next MGI, then continuing this to make a chain through all MGI. The last MGI sends a trigger back to the power controller trigger generator logic to indicate that the operation is complete. This could reduce the routing of the trigger signals. However, there are disadvantages, to maintain the chain no MGIs can be powered off, and there is potentially sample skew between the various samples. Whether these are issues depends upon the system topology and system requirements.

This structure can be easily extended to add more sensors or groups of sensors as required. It can also be extended to support multiple cores if the physical location of the MGI is amenable to providing interfaces for them.

Figure A2.3 shows an example of combining the sensors and monitors from two cores into a single set of monitor groups.

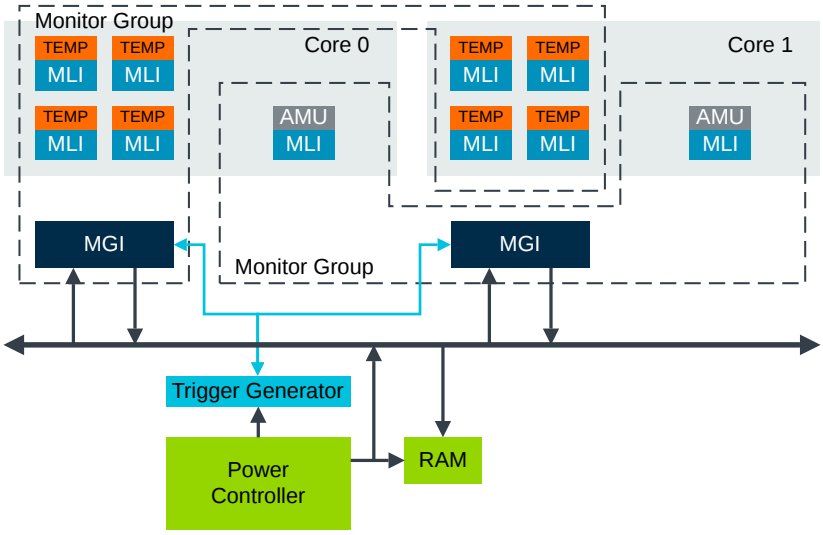


Figure A2.3: Combining groups for multiple cores

If both cores can be powered off independently, this implementation could require their MGIs to be placed external to those power domains and use the disconnection mechanism to allow the correct indication of powered off sensors.

A2.2.2 Optional features required

The following features are required in the SMCF to perform this function.

- DMA Interface.
- Input Trigger.
- Sample Delay (at a size to meet the user requirements).

A2.2.3 Control procedures

The control procedure is based on a power controller consuming the data on a periodic basis timed by the power controller itself, typically using a timer interrupt.

A2.2.3.1 Initial configuration

Each monitor group is setup with initial parameters. This includes enabling the required set of sensors and monitors, and programming the memory-mapped DMA location and monitor mode settings.

The initial setup procedure includes the following steps for each MGI to be periodically sampled.

1. Program MGI_SMP_CFG to configure samples to start on the input trigger.
2. Program MGI_WRCFG so data is written on every monitor sample data set completion.
3. Program MGI_WADDR<n> with the address where the data from this group is to be written.
4. Program MGI_WREN to enable the writing out of completed monitor sample data sets.
5. Program MGI_SMP_DLY with the required sample delay value.
6. Program MGI_MON_REQ to enable the required monitors.
7. Read MGI_MON_STAT to ensure the monitors have been enabled.
8. Program MGI_MODE_BCAST and MGI_MODE_REQ<n> to set any required monitor modes.
9. Read MGI_MODE_STAT<n> to ensure any required modes have been set.

10. Program MGI_SMP_EN to enable sampling.

When this programming is complete the MGI starts a sample whenever there is an input trigger event.

The sample delay setting is used to make the sample data more current to when the processing takes place. For instance, if sampling only takes a short time, relative to the sampling period, then the data might be somewhat out of date before the power controller comes to read and process it. The sample delay function delays the start of the sample from the input trigger by the number of MGI cycles specified in MGI_SMP_DLY, see Figure A2.4. When calculating the required sample delay value, the parameters that need to be considered are:

- The power controller's processing period
- The power controller's data processing time
- The monitor sampling time
- The required time for the writing out of monitor data

A2.2.3.2 Continuous actions

The ongoing procedure is as follows:

1. A power controller timer interrupt event occurs (this happens on a regular tick within the power controller, for instance at approximately a millisecond rate)
2. The power controller processes available data from all MGI at the memory-mapped locations where is was written.
   - If this is the first time this has occurred since the MGIs were enabled, there is not any data available yet, so this action is skipped.
3. The power controller uses its trigger generation logic to broadcast an input trigger to all MGI.
4. The power controller then waits for the next timer interrupt.

Figure A2.4 shows this process.

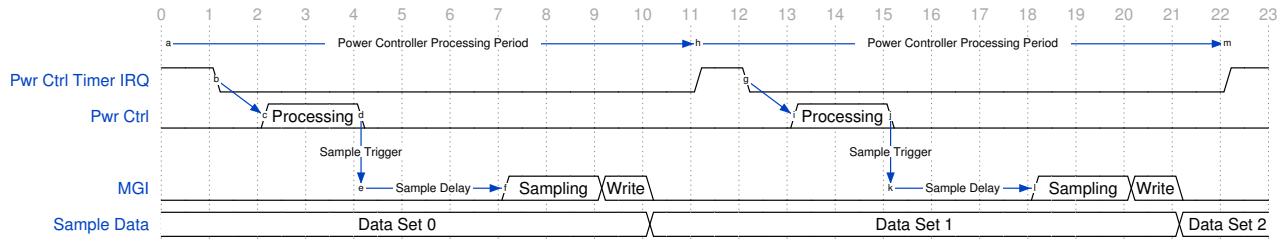


Figure A2.4: Continuous sampling timing diagram

A2.3 Self Monitoring

In a small scale SoC, like one used in the IoT space, there can be sensors that are required to be constantly monitored but the data only acted upon if the sensor gives a specific output or crosses a threshold. This might be temperature data or data from other system sensors or monitors. In this type of SoC, there might not be a separate core for managing this data, so constant processing is not required and could be a burden on a processor with other tasks.

In this case the requirement is to sample constantly but only alert when an event of interest has occurred.

A2.3.1 Example structure

The MGI is monitoring a single sensor or multiple sensors, with interrupts connected to a processor.

The timing of the sampling periods can be done by either using the MGI internal periodic timer, or with an external timer generating events for the input trigger. The latter is more useful if there are multiple MGIs that need to be timed as this prevents having a timer running in each MGI.

In this example it is assumed that the MGI internal periodic timer is being used.

A2.3.2 Optional features required

- Periodic timer.
- Alerts (as required)

A2.3.3 Control procedures

A2.3.3.1 Initial configuration

The initial setup procedure includes the following steps for each MGI to be periodically sampled.

1. Program MGI_SMP_PER to configure the interval on which the monitors are sampled.
   - This value must be larger than the time taken to perform a sample.
2. Program MGI_SMP_CFG to configure periodic sampling.
3. Program MGI_ATYP<n> to setup any required alerts.
4. Program MGI_AVAL<n> to setup the required values relating to the alert types.
5. Program MGI_IRQ_MASK to unmask the required alert interrupt events.
6. Program MGI_MON_REQ to enable the required monitors.
7. Read MGI_MON_STAT to ensure the monitors have been enabled.
8. Program MGI_MODE_BCAST and MGI_MODE_REQ<n> to set any required monitor modes.
9. Read MGI_MODE_STAT<n> to ensure any required modes have been set.
10. Program MGI_SMP_EN to enable sampling.

When this programming is complete the MGI starts a sample on every programmed period and sends an interrupt when an alert is triggered.

A2.3.3.2 Continuous actions

The only actions required are to respond to an alert interrupt when it occurs.

The sequence of actions when an interrupt is received is:

1. Read MGI_IRQ_STAT to determine the cause of the interrupt.
2. Take IMPLEMENTATION DEFINED actions based on the interrupt status.
   - This could involve reading the monitor data from the MGI and taking actions based on the values read.

3. Write 0x1 to the relevant resolved interrupt event bit in MGI_IRQ_STAT.

The interrupt might remain asserted if there are multiple interrupt events and only the status of certain events is cleared.

The most current monitor sample data set can be read from the MGI at any point regardless of the presence of an interrupt event. The data can be checked to ensure it has not been updated during a read by using MGI_SMPID_START and MGI_SMPID_END. For more information on this process see 4.2 *Sample identifiers*.

Chapter A3
Monitor Group Interfaces and Monitor Local Interfaces

This section describes implementation options for the interfaces between a Monitor Group Interface (MGI) and a Monitor Local Interface (MLI) in the System Monitoring Control Framework (SMCF). This section also describes the use of the Arm Low Power Interfaces (LPI) on MGIs and MLIs.

A3.1 MGI to MLI interfaces

A3.1.0.1 Multiple monitors for each MLI

In a typical implementation each monitor has its own MLI interface, but this is not a requirement and can depend on the type and structure of the monitor.

For instance, some monitors might be delivered as a hub to share some of the parts common to several sensors and produce a more efficient design. Alternatively, an MLI might interface to an FSM that reads data from several closely located monitors.

These instances typically only require a single MLI to support them. In this case, the assignment of monitors can be done in one of two ways, either:

- The monitors can be described as separate entities in the MGI and the commands for each monitor routed onto the same interface to the MLI.
- The monitors can be described as a single monitor in the MGI that outputs several monitor data values.

The first method has the advantage of being able to individually manage monitors, for example enabling and disabling each monitor and setting different monitor modes. However, this adds more complexity in an MLI because it has to parse each command and direct it to the relevant monitor. Also, the MGI has to be designed to route commands for separate monitors over the same MLI interface.

The second method is easier to implement but reduces flexibility because from an MGI perspective it must be enabled and disabled as a single monitor. This requires each monitor to have the same mode and generate data on each sample.

An alternate approach is to have multiple interfaces on the MGI that communicate with a single MLI that controls all the monitors. This reduces the MGI complexity but keeps the complexity in the MLI and increases the routing between the two components.

A3.1.0.2 Low Power Interface use with MGI and MLIs

The Arm Low Power Interfaces (LPI) provide a mechanism to allow components to safely enter a logical state for entry into low power modes. This section describes how these interfaces can enhance the functionality of the MGI and MLI components.

For more details on the LPIs and its use see [1] and [2].

A3.1.0.2.1 MGI LPI

When an MGI has an LPI this can be used to request it to enter a low power state. When the LPI requests a power mode where the MGI does not perform sampling, like power off, this mechanism can be used to update the sample identifier. The sample identifier is, optionally, written out by the DMA interface with the monitor data. This can indicate the MGI is powered off, so the controlling agent knows why the data is not being updated. For more information on sample identifier values see 4.2.1 *Sample identifier format*.

Also, on the exit of the MGI from a low power mode, the LPI can be used to generate the Configuration Request interrupt event.

A3.1.0.2.2 MLI LPI

When an MLI has an LPI this can be used to request it to enter a low power state. This typically only needs to generate communication with an MGI when the MLI is in a different power domain. When the LPI requests a power mode where the MLI does not perform sampling, like power off, this can be used to send a Disconnect command to the MGI. When an acknowledgment to this command is received the MLI can accept the LPI request and be powered off. For more information on MLI connection and disconnection see 2.9 *MLI connection and disconnection*.

Also, at the exit of the MLI from a low power mode the LPI can be used to generate a Connect command to reconnect the MLI to the MGI.

Chapter A4
Security Integration

This section describes any security consideration that might be required when using the System Monitoring Control Framework (SMCF) to collect sensitive data. This includes how to control access to the Monitor Group Interfaces (MGI).

A4.1 Introduction

Monitors and sensors in the SoC might provide data that is considered sensitive, because it might contain information about the system that could be used maliciously if exposed at an inappropriate level.

Therefore, access restrictions to the data might be required. This means access to SMCF MGIs should be limited to agents in the system with the required level of trust. This section describes some examples of how the SMCF MGIs can be integrated to meet limited access requirements. The specific requirements are system dependent.

A4.1.1 M-Profile System without TrustZone

This section is describes a system with an Arm M Profile core as the main processor, where the processor does not have TrustZone capabilities.

In this case, access to the SMCF MGI can be limited by only allowing access to privileged software.

In the Cortex-M3 this can be done by attaching the MGI to the External Private Peripheral Bus (EPPB), that only allows privileged access, or alternately by attaching the MGI on the main system bus and using a component to prevent accesses based on the AHB bus **HPROTS[1]** bit. This bit indicates if the access is privileged or unprivileged. When the access is unprivileged it should be treated as RAZ/WI.

For more information on the AHB signals, see [3].

A4.1.2 M-Profile System with TrustZone

This section is describes a system with an Arm M Profile core as the main processor, where the processor has TrustZone capabilities.

In this case, access to the SMCF MGI can be limited by the TrustZone functionality. Only accesses from secure software should be allowed. When the access is non-secure it should be treated as RAZ/WI.

The security status of the access is indicated on the bus. For a processor with an AHB interface, for instance the Cortex-M33, this is indicated on the AHB **HNONSECS** signal. For an M-profile processor with a AXI interface, for instance the Cortex-M55, this is indicated on the AXI **ARPROT[1]** and **AWPROT[1]** signals.

For more information on the AHB and AXI signals, see [4] and [3].

A4.1.3 A-Profile System with a System Control Processor

This section is describes a system with an Arm A-Profile core as the main processor, where there is an additional processor that performs as a System Control Processor (SCP). The SCP is a trusted agent running platform dependent firmware.

In this case, the SCP is the agent that has exclusive access to the SMCF MGI. The SCP typically uses the data for its own purposes in managing various aspect of the SoC, including power control. The application processor, or other system agents, can then request access to the data from the SCP, for example, through a Message Handling Unit (MHU) using the System Control and Management Interface (SCMI) Sensor Management protocol. The SCP can then control if access is permitted and if so, then at which level of detail of information is shared. For example, the SCP might choose to only share consolidated or averaged values of certain data, or might completely restrict access a certain class of data.

For more information on SCMI, see [5].

A4.1.4 A-Profile System with TrustZone

This section is describes a system with an Arm A-Profile core as the main processor, where it uses TrustZone to determine the correct access to monitors in the system.

In this case, access to the SMCF MGI can be limited by the TrustZone functionality. Only accesses from secure software should be allowed. When the access is non-secure it should be treated as RAZ/WI.

The security status of the access is indicated on the bus. For a processor with an CHI interface this is indicated on the NS bit in the CHI request flit. For a processor with a AXI interface this is indicated on the AXI **ARPROT[1]** and **AWPROT[1]** signals.

For more information on the AHB and AXI signals, see [4] and [6].

Glossary

AHB

Advanced High-performance Bus, see [3]

AMU

Activity Monitoring Unit

AXI

Advanced eXtensible Interface, see [4]

CHI

See [6]

DMA

Direct Memory Access

LPI

Low Power Interface, see [1]

LSB

Least Significant Bit

MGI

Monitor Group Interface

MLI

Monitor Local Interface

Monitor Data Value

A single data value produced from a monitor. A monitor can produce more than one data value for each monitor sample.

Monitor Sample Data Set

The complete set of all sampled monitor data values from all enabled and connected monitors in an MGI.

MSB

Most Significant Bit

MSI

Monitor Serial Interface

Relatively always-on

Indicates a power domain that can be powered off, but is powered on whenever the specified child domain is powered on.

RES0

A reserved bit.

SCMI

System Control and Management Interface, see [5]

SMCF

System Monitoring Control Framework

SoC

System on Chip