

# ARM® System Trace Macrocell

**Programmers' Model Architecture Specification Version  
1.1**



# ARM System Trace Macrocell

## Programmers' Model Architecture Specification Version 1.1

Copyright © 2010, 2013 ARM. All rights reserved.

### Release Information

The following changes have been made to this book.

#### Change history

Date	Issue	Confidentiality	Change
23 April 2010	A	Non-Confidential	First release for v1.0
26 September 2013	B	Non-Confidential	First release for v1.1

### Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM’s trademark usage guidelines at <http://www.arm.com/about/trademark-usage-guidelines.php>.

Copyright © 2010, 2013, ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

### Product Status

The information in this document is final, that is for a developed product.

### Web Address

<http://www.arm.com>

# Contents

## ARM System Trace Macrocell Programmers' Model Architecture Specification Version 1.1

### Preface

About this book .....	vi
Using this book .....	vii
Conventions .....	viii
Additional reading .....	ix
Feedback .....	x

### Chapter 1

#### Introduction

1.1 About the System Trace Macrocell .....	1-12
--	------

### Chapter 2

#### Configuration Registers Programmers' Model

2.1 About the configuration registers programmers' model .....	2-14
2.2 Register summary .....	2-15
2.3 Register descriptions .....	2-17
2.4 Programming the STM .....	2-45
2.5 Triggers .....	2-46
2.6 Authentication control .....	2-49

### Chapter 3

#### Extended Stimulus Ports

3.1 About extended stimulus ports .....	3-52
3.2 STM transactions .....	3-54
3.3 Address decoding .....	3-55
3.4 Grouping stimulus ports .....	3-56
3.5 More than one master .....	3-57
3.6 Data sizes .....	3-58
3.7 Bus endianness .....	3-59

3.8	Implementation options .....	3-60
3.9	Reserved locations .....	3-61
3.10	Timestamping .....	3-62
3.11	Mapping onto STPv2 .....	3-63

## **Chapter 4**

### **Implementation Defined Controls**

4.1	About implementation defined controls and registers .....	4-66
4.2	Standard hardware event tracing .....	4-67
4.3	DMA control .....	4-77

## **Appendix A**

### **Recommended Configurations**

A.1	About recommended configurations .....	A-82
-----	--	------

## **Appendix B**

### **Revisions**

### **Glossary**

# Preface

This preface introduces the *System Trace Macrocell* (STM) Programmers' Model Architecture Specification. It contains the following sections:

- *About this book* on page vi.
- *Using this book* on page vii.
- *Conventions* on page viii.
- *Additional reading* on page ix.
- *Feedback* on page x.

## About this book

This book describes the ARM *System Trace Macrocell* (STM) programmers' model architecture. Some parts of the STM programmers' model architecture are IMPLEMENTATION DEFINED. For more information see the applicable STM *Technical Reference Manual* (TRM).

### Intended audience

This book is written for the following target audiences:

- Designers of development tools providing support for STM functionality. All chapters in this book are of interest to these users.
- Advanced users of development tools providing support for STM functionality. [Chapter 2](#) is particularly relevant to these users.
- Designers of an ARM processor based product that includes an STM trace port. [Chapter 3](#) is particularly relevant to these users.
- Engineers who want to specify, design, or implement an STM to the ARM STM Architecture.

Hardware engineers who want to incorporate an ARM STM into their design must consult the applicable STM *Technical Reference Manual* listed in [Additional reading on page ix](#). ARM recommends that all users of this book have experience of the ARM architecture.

## Using this book

This book is organized into the following chapters:

### **Chapter 1** *Introduction*

Read this for an introduction to the STM.

### **Chapter 2** *Configuration Registers Programmers' Model*

Read this for information about the configuration registers, and how to program the STM. It also describes triggers and authentication control.

### **Chapter 3** *Extended Stimulus Ports*

Read this for information about the extended stimulus ports and the transaction types.

### **Chapter 4** *Implementation Defined Controls*

Read this for information about the IMPLEMENTATION DEFINED controls and registers.

### **Appendix A** *Recommended Configurations*

Read this for information about the recommendations for using the STM architecture in different implementations.

### **Appendix B** *Revisions*

Read this for a description of the technical changes between released issues of this book.

### **Glossary**

Read this for definitions of terms used in this book.

#### **Note**

ARM publishes a single glossary that relates to most ARM products, see the *ARM® Glossary* <http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-> . A definition in the glossary in this specification might be more detailed than the corresponding definition in the *ARM® Glossary*.

## Conventions

The following sections describe conventions that this book can use:

- [Typographic conventions](#).
- [Signals](#).
- [Numbers](#).

### Typographic conventions

The typographical conventions are:

<i>italic</i>	Introduces special terminology, and denotes citations.
<b>bold</b>	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 a few terms that have specific technical meanings, and are included in the [Glossary](#).

<b>Colored text</b>	Indicates a link. This can be: <ul style="list-style-type: none"><li>• A URL, for example <a href="http://infocenter.arm.com">http://infocenter.arm.com</a>.</li><li>• A cross-reference, that includes the page number of the referenced information if it is not on the current page, for example, <a href="#">About the System Trace Macrocell on page 1-12</a>.</li><li>• A link, to a chapter or appendix, or to a glossary entry, or to the section of the document that defines the colored term, for example <a href="#">STMSPSCR</a>.</li></ul>
---------------------	--

### Signals

In general this specification does not define processor signals, but it does include some signal examples and recommendations. The signal conventions are:

<b>Signal level</b>	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means: <ul style="list-style-type: none"><li>• HIGH for active-HIGH signals.</li><li>• LOW for active-LOW signals.</li></ul>
<b>Lower-case n</b>	At the start or end of a signal name denotes an active-LOW signal.

### 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.



## Additional reading

This section lists relevant publications from ARM and third parties.

See *Infocenter* <http://infocenter.arm.com>, for access to ARM documentation.

### ARM publications

This specification defines the System Trace Macrocell programmers' model architecture. See the following documents for other relevant information:

- *ARM® CoreSight™ System Trace Macrocell Technical Reference Manual* (ARM DDI 0444).
- *ARM® CoreSight™ System Trace Macrocell-500 Technical Reference Manual* (ARM DDI 0528).
- *ARM® Architecture Reference Manual, ARMv7-M edition* (ARM DDI 0403).
- *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406).
- *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).
- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (ARM IHI 0031).
- *ARM® RealView® ICE and RealView Trace User Guide* (ARM DUI 0155).

### Other publications

This section lists relevant documents published by third parties:

- *MIPI System Trace Protocol version 2 (STPv2)*.

## 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](mailto:errata@arm.com). Give:

- The title.
- The number, ARM IHI 0054B.
- The page numbers to which your comments apply.
- 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 chapter introduces the *System Trace Macrocell* (STM). It contains the following section:

- [About the System Trace Macrocell on page 1-12.](#)

## 1.1 About the System Trace Macrocell

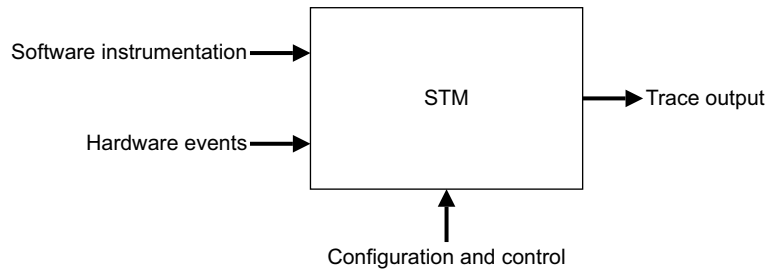
The STM enables tracing of system activity from various sources:

- Instrumented software, using memory-mapped stimulus ports.
- Hardware events.

The activity observed by the STM is packaged into a trace stream, for output to trace capture devices such as those provided by CoreSight technology.

This version of the STM architecture supports a trace stream that conforms to the *MIPI System Trace Protocol version 2* (STPv2).

Figure 1-1 shows the STM inputs and outputs.



**Figure 1-1 STM inputs and outputs**

The STM programmers' model has two main parts:

### Configuration registers

These registers are accessible both by software running on the chip and by an external debugger and are used to configure the tracing activity of the STM. The configuration registers also include optional basic stimulus port registers. For more information on the configuration registers, see [Chapter 2 Configuration Registers Programmers' Model](#).

### Extended stimulus port registers

These registers are accessible by instrumented software running on the chip, but are not necessarily accessible by an external debugger. Up to 65536 extended stimulus ports are provided. For more information on the extended stimulus port registers, see [Chapter 3 Extended Stimulus Ports](#).

The STM supports the following:

- Multiple software masters writing software instrumentation independently. Each master can use multiple stimulus ports.
- Timestamping of the system activity. The timestamp is a global timestamp which can be shared with other trace sources in the system, to enable correlation of activity from multiple trace sources.
- Interaction with DMA controllers, to manage the flow of data in the system.
- Indicating that specific events have occurred, such as the occurrence of a particular hardware event or a particular piece of software instrumentation. These events are known as triggers and can be indicated in the trace stream, or through signals to other system components.

# Chapter 2

## Configuration Registers Programmers' Model

This chapter describes the configuration registers that you can program to set up and control the STM. It contains the following sections:

- *About the configuration registers programmers' model on page 2-14.*
- *Register summary on page 2-15.*
- *Register descriptions on page 2-17.*
- *Programming the STM on page 2-45.*
- *Triggers on page 2-46.*
- *Authentication control on page 2-49.*

## 2.1 About the configuration registers programmers' model

The configuration registers occupy a 4KB block, with a CoreSight programmers' model compatible structure. The STM configuration registers are used to set up the STM implementation.

The following apply to the STM registers:

- Accesses to Reserved locations are UNK/SBZP.
- Accesses to Reserved bits in defined registers are UNK/SBZP unless otherwise stated.
- Registers reset to an UNKNOWN value unless specifically defined.

## 2.2 Register summary

Table 2-1 shows the STM registers. In the table, access type is described as follows:

<b>RW</b>	Read and write.
<b>RO</b>	Read only.
<b>WO</b>	Write only.

**Table 2-1 STM configuration register summary**

Address offset	Name	Type	Description
0x000-0x07C	Basic Stimulus Ports	RW	See <i>STMSTIMR&lt;n&gt;</i> , <i>Basic Stimulus Ports</i> on page 2-17
0x080-0x9FC	-	-	Reserved
0xA00-0xAFC	IMPLEMENTATION DEFINED Block 3		See <i>Chapter 4 Implementation Defined Controls</i>
0xB00-0xBFC	IMPLEMENTATION DEFINED Block 2		
0xC00-0xCFC	IMPLEMENTATION DEFINED Block 1		
0xD00-0xDFC	IMPLEMENTATION DEFINED Block 0		
0xE00-0xE7C	Stimulus Port Control Registers		
0xE00	Stimulus Port Enable	RW	See <i>STMSPER</i> , <i>Stimulus Port Enable Register</i> on page 2-17
0xE04-0xE1C	-	-	Reserved
0xE20	Stimulus Port Trigger Enable	RW	See <i>STMSPTER</i> , <i>Stimulus Port Trigger Enable Register</i> on page 2-18
0xE24-0xE3C	-	-	Reserved
0xE40	Trace Privilege	RW	See <i>STMPRIVMASKR</i> , <i>Trace Privilege Register</i> on page 2-19
0xE44-0xE5C	-	-	Reserved
0xE60	Stimulus Port Select Configuration	RW	See <i>STMSPSCR</i> , <i>Stimulus Port Select Configuration Register</i> on page 2-20
0xE64	Stimulus Port Master Select Configuration	RW	See <i>STMSPMSCR</i> , <i>Stimulus Port Master Select Configuration Register</i> on page 2-22
0xE68	Stimulus Port Override	RW	See <i>STMSPOVERRIDER</i> , <i>Stimulus Port Override Register</i> on page 2-24
0xE6C	Stimulus Port Master Override	RW	See <i>STMSPMOVERRIDER</i> , <i>Stimulus Port Master Override Register</i> on page 2-26
0xE70	Stimulus Port Trigger Control and Status	RW	See <i>STMSPTRIGCSR</i> , <i>Stimulus Port Trigger Control and Status Register</i> on page 2-28
0xE74-0xE7C	-	-	Reserved
0xE80-0xE9C	Primary Control and Status Registers		
0xE80	Trace Control and Status	RW	See <i>STMTCR</i> , <i>Trace Control and Status Register</i> on page 2-29
0xE84	Timestamp Stimulus	WO	See <i>STMSTSTIMR</i> , <i>Timestamp Stimulus Register</i> on page 2-31
0xE88	-	-	Reserved
0xE8C	Timestamp Frequency	RW	See <i>STMSTFREQR</i> , <i>Timestamp Frequency Register</i> on page 2-32
0xE90	Synchronization Control	RW	See <i>STMSYNCR</i> , <i>Synchronization Control Register</i> on page 2-33

**Table 2-1 STM configuration register summary (continued)**

Address offset	Name	Type	Description
0xE94	Auxiliary Control	RW	See <i>STMAUXCR, Auxiliary Control Register</i> on page 2-33
0xE94-0xE9C	-	-	Reserved
0xEA0-0xEAC	<b>Identification Registers</b>		
0xEA0	Features 1	RO	See <i>STMFEAT1R, Features 1 Register</i> on page 2-34
0xEA4	Features 2	RO	See <i>STMFEAT2R, Features 2 Register</i> on page 2-36
0xEA8	Features 3	RO	See <i>STMFEAT3R, Features 3 Register</i> on page 2-37
0xEAC-0xEFC	-	-	Reserved
0xF00-0xFFC	<b>CoreSight Management Registers</b>		
0xF00	Integration Mode Control	RW	See <i>STMITCTRL, Integration Mode Control Register</i> on page 2-38
0xF04-0xF9C	-	-	Reserved
0xFA0	Claim Tag Set	RW	See <i>STMCLAIMSET, Claim Tag Set Register</i> on page 2-39
0xFA4	Claim Tag Clear	RW	See <i>STMCLAIMCLR, Claim Tag Clear Register</i> on page 2-39
0xFA8-0xFAC	-	-	Reserved
0xFB0	Lock Access	WO	See <i>STMLAR, Lock Access Register</i> on page 2-40
0xFB4	Lock Status	RO	See <i>STMLSR, Lock Status Register</i> on page 2-41
0xFB8	Authentication Status	RO	See <i>STMAUTHSTATUS, Authentication Status Register</i> on page 2-41
0xFBC	Device Architecture	RO	See <i>STMDEVARCH, Device Architecture Register</i> on page 2-41
0xFC0-0xFC4	-	-	Reserved
0xFC8	Device Configuration	RO	See <i>STMDEVID, Device Configuration Register</i> on page 2-42
0xFCC	Device Type	RO	See <i>STMDEVTYPE, Device Type Register</i> on page 2-43
0xFD0-0xFEC	Peripheral ID	RO	See <i>STMPIDR0-7, Peripheral ID Registers</i> on page 2-43
0xFF0-0xFFC	Component ID	RO	See <i>STMCIDR0-3, Component ID Registers</i> on page 2-44



## 2.3 Register descriptions

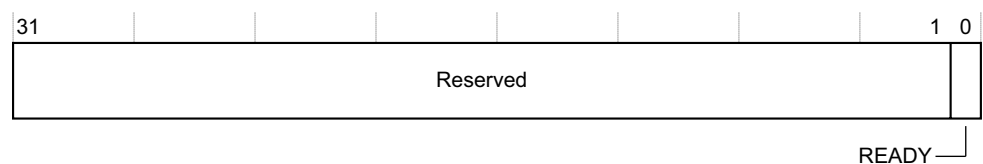
Table 2-1 on page 2-15 lists the STM registers. This section describes each of the registers.

### 2.3.1 STMSTIMR<n>, Basic Stimulus Ports

The STMSTIMR<n> characteristics are:

<b>Purpose</b>	<p>Provides up to 32 stimulus ports.</p> <p>Write accesses to these basic stimulus ports are identical to write accesses to the I_DMTS variant of the corresponding extended stimulus ports 0-31 on master 0. See <a href="#">Chapter 3 Extended Stimulus Ports</a>.</p> <p>Read accesses are used to determine if a future write to the register is accepted.</p>
<b>Usage constraints</b>	There are no usage constraints. Accesses to these registers are unaffected by the lock mechanism, see <a href="#">Lock Registers on page 2-40</a> .
<b>Configurations</b>	These registers are optional. Read <a href="#">STMFEAT2R</a> to determine if the basic stimulus ports are implemented.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-1 shows the STMSTIMR<n> bit assignments on reads.



**Figure 2-1 STMSTIMR<n> bit assignments on reads**

Table 2-2 shows the STMSTIMR<n> bit assignments on reads.

### Table 2-2 STMSTIMR<n> bit assignments on reads

Bits	Name	Description
[31:1]	-	Reserved, UNK/SBZP.
[0]	READY	<div>0b0</div> <div>A write to the stimulus port is not accepted. This value is returned when the selected stimulus port is disabled or when the STM is unable to accept a write, for example, when any buffering is full.</div> <div>0b1</div> <div>The STM can accept a write to a stimulus port.</div>

**————— Note —————**

Only supports up to 32 basic stimulus ports, even if the STM supports more than 32 extended stimulus ports.

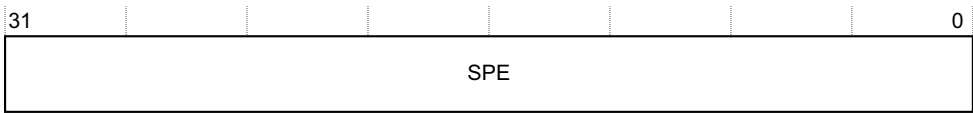
### 2.3.2 STMSPER, Stimulus Port Enable Register

The STMSPER characteristics are:

<b>Purpose</b>	Enables the stimulus port registers to generate trace. This register defines one bit per stimulus port. Writing 0b1 enables the appropriate stimulus port, writing 0b0 disables the appropriate stimulus port. This register is used in conjunction with the <a href="#">STMSPSCR</a> .
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.

**Attributes** See the register summary in [Table 2-1 on page 2-15](#).

[Figure 2-2](#) shows the STMSPER bit assignments.



**Figure 2-2 STMSPER bit assignments**

[Table 2-3](#) shows the STMSPER bit assignments.

**Table 2-3 STMSPER bit assignments**

Bits	Name	Description
[31:0]	SPE	Stimulus port enable, with one bit per stimulus port:  0b0 Stimulus port disabled. 0b1 Stimulus port enabled. The reset value of each bit is 0b0. If the number of stimulus ports is less than or equal to 32, the number of bits in the SPE field is the number of stimulus ports. If the number of stimulus ports is greater than 32, the SPE field is 32-bits wide and the <a href="#">STMSPSCR</a> controls which stimulus ports are enabled in conjunction with the SPE field.

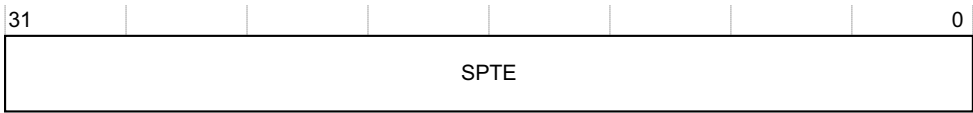
**Note**  
Bit[0] applies to the lowest-numbered port and bit[31] to the highest-numbered port.

**2.3.3 STMSPTER, Stimulus Port Trigger Enable Register**

The STMSPTER characteristics are:

- Purpose** Enables trigger generation on writes to enabled stimulus port registers.
- Usage constraints** There are no usage constraints.
- Configurations** This register is optional. Read [STMFEAT2R](#) to determine if it is implemented or write a non-zero value and read it back. If a non-zero value is returned, this register is implemented.
- Attributes** See the register summary in [Table 2-1 on page 2-15](#).

[Figure 2-3](#) shows the STMSPTER bit assignments.



**Figure 2-3 STMSPTER bit assignments**

Table 2-4 shows the STMSPTER bit assignments.

### Table 2-4 STMSPTER bit assignments

Bits	Name	Description
[31:0]	SPTE	<p>Bit mask to enable trigger generation from the stimulus port registers, with one bit per stimulus port register:</p> <p>0b0 Disabled.</p> <p>0b1 Enabled.</p> <p>The reset value of each bit is 0b0.</p> <p>If the number of stimulus ports is less than or equal to 32, the number of bits in the SPTE field is the number of stimulus ports.</p> <p>If the number of stimulus ports is greater than 32</p> <p>The SPTE field is 32-bits wide.</p> <p>The <a href="#">STMSPSCR</a> controls which stimulus ports have triggers enabled, in conjunction with the SPTE field.</p>

**————— Note —————**

Bit[0] applies to the lowest-numbered port and bit[31] to the highest-numbered port.

### 2.3.4 STMPRIVMASKR, Trace Privilege Register

The STMPRIVMASKR characteristics are:

<b>Purpose</b>	Enables an operating system to control which stimulus ports are accessible by user code.
<b>Usage constraints</b>	You can only write to this register in a privileged mode or from an external debugger.
<b>Configurations</b>	This register is optional. Read <a href="#">STMFEAT2R</a> to determine if it is implemented or write a non-zero value and read it back. If a non-zero value is returned, this register is implemented.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-4 shows the STMPRIVMASKR bit assignments.



### Figure 2-4 STMPRIVMASKR bit assignments

Table 2-5 shows the STMPRIVMASKR bit assignments.

### Table 2-5 STMPRIVMASKR bit assignments

Bits	Name	Description
[31:m]	-	Reserved, RAZ.
[m-1:0]	PRIVMASK	<p>Bit mask to control user mode access to stimulus ports. Each bit controls eight stimulus ports:</p> <p>0b0                      User mode and privileged accesses are permitted.</p> <p>0b1                      User mode accesses are ignored.</p> <p>Bit[n] controls access to stimulus ports (8n to 8n+7).</p> <p>The reset value is 0b0.</p>

———— **Note** ————

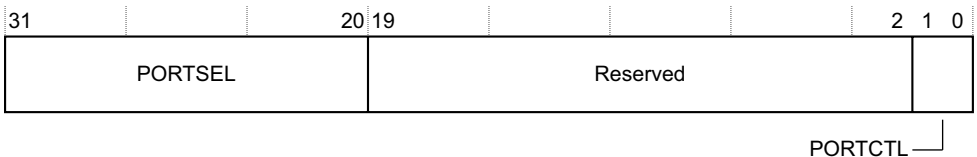
- The variable *m* is defined by the number of supported stimulus ports. For example if 32 stimulus ports are supported, *m* is 4.
- This register only supports control for up to 256 stimulus ports. The access permissions apply to the basic stimulus ports and extended stimulus ports.

**2.3.5 STMSPSCR, Stimulus Port Select Configuration Register**

The STMSPSCR characteristics are:

<b>Purpose</b>	Enables a debugger to program which stimulus ports the <a href="#">STMSPER</a> and <a href="#">STMSPTER</a> apply to.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	If 32 or fewer stimulus ports are implemented, this register is not implemented and is Reserved.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-5](#) shows the STMSPSCR bit assignments.



**Figure 2-5 STMSPSCR bit assignments**

[Table 2-6](#) shows the STMSPSCR bit assignments.

**Table 2-6 STMSPSCR bit assignments**

Bits	Name	Description
[31:20]	PORTSEL	Port Selection. This field defines which stimulus ports the <a href="#">STMSPER</a> and/or <a href="#">STMSPTER</a> apply to. The size of this field is defined by the number of implemented stimulus ports. The reset value is UNKNOWN.
[19:2]	-	Reserved, UNK/SBZP.
[1:0]	PORTCTL	This defines how the port selection is applied: 0b00 Port selection not used. 0b01 Port selection applies only to the <a href="#">STMSPTER</a> . 0b10 Reserved. 0b11 Port selection applies to both the <a href="#">STMSPER</a> and <a href="#">STMSPTER</a> . The reset value is 0b00.

**PORTCTL == 0b00**

When PORTCTL is 0b00, the [STMSPER](#) and [STMSPTER](#) apply equally to every group of 32 stimulus ports and PORTSEL is ignored. For example:

- Bit[0] of the [STMSPER](#) is 0b1.
- Bit[0] of the [STMSPTER](#) is 0b1.

This enables stimulus ports 0, 32, 64, 96, 128, and so on. Triggers are caused on writes to stimulus ports 0, 32, 64, 96, 128, and so on. All other stimulus ports are disabled and do not cause triggers.

### PORTCTL != 0b00

When PORTCTL is not 0b00, the PORTSEL field enables you to select a subset of the full stimulus ports to which the STMSPER and STMSPTER apply. PORTSEL enables you to select a single group of 32 stimulus ports or power-of-two multiples of consecutive groups to which to apply the STMSPER and STMSPTER.

To program PORTSEL, the bottom N bits which are 0 define a mask to apply to the port selection, then a 1 in bit N+1 demarks the mask from the port selection. The bits from N+2 to M select the groups to which the STMSPER and STMSPTER apply.

For example:

**PORTSEL** = bbb\_bbbb\_bbbb\_1

A single group of 32 stimulus ports bbb\_bbbb\_bbbb is selected.

**PORTSEL** = bbb\_bbb1\_0000\_0

A selection of 32 groups of 32 stimulus ports from bbb\_bbb0\_0000 to bbb\_bbb1\_1111 is selected.

**PORTSEL** = 100\_0000\_0000\_0

All stimulus ports are selected. This is equivalent to PORTCTL == 0b00.

Programming PORTCTL != 00 and PORTSEL = 000\_0000\_0000\_0 is UNPREDICTABLE.

Programming a PORTSEL value which enables more stimulus ports than are implemented results in UNPREDICTABLE behavior, for example, programming 100\_0000\_0000\_0 when only 32 stimulus ports are implemented. To enable all 32 stimulus ports, program 000\_0001\_0000\_0.

Triggers cannot be generated by writes to stimulus ports which are not enabled. Enabling a trigger on a stimulus port which is not enabled results in UNPREDICTABLE behavior.

### Using PORTCTL

Table 2-7 shows how to use PORTCTL.

**Table 2-7 Using PORTCTL**

PORTCTL	Description
0b00	<p>Port selection select is not used.</p> <p>STMSPER and STMSPTER apply equally to every group of 32 stimulus ports. PORTSEL is ignored. For example:</p> <ul style="list-style-type: none"> <li>Only bit[0] of the STMSPER is 0b1.</li> <li>Only bit[0] of the STMSPTER is 0b1.</li> </ul> <p>This enables stimulus ports 0, 32, 64, 96, 128, and so on. Triggers are caused on writes to stimulus ports 0, 32, 64, 96, 128, and so on. All other stimulus ports are disabled and do not cause triggers.</p>

Table 2-7 Using PORTCTL (continued)

PORTCTL	Description
0b01	<p>Port selection only applies to the <a href="#">STMSPTER</a>.  <a href="#">STMSPTER</a> applies equally to every group of 32 stimulus ports.  <a href="#">STMSPTER</a> only applies to the groups of 32 stimulus ports selected by PORTSEL and other groups do not cause triggers.  For example:</p> <ul style="list-style-type: none"> <li>• PORTSEL is b000_0000_0001_1 (select group 1).</li> <li>• Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> <li>• Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> </ul> <p>This enables stimulus ports 0, 32, 64, 96, 128, and so on. Triggers are only caused on writes to stimulus port 32. All other stimulus ports are disabled and do not cause triggers.</p>
0b10	Reserved.
0b11	<p>Port selection applies to <a href="#">STMSPTER</a> and <a href="#">STMSPTER</a>.  <a href="#">STMSPTER</a> and <a href="#">STMSPTER</a> only apply to the groups selected by PORTSEL. Other groups are not enabled and do not cause triggers.  For example:</p> <ul style="list-style-type: none"> <li>• PORTSEL is b000_0000_0001_1 (select group 1).</li> <li>• Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> <li>• Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> </ul> <p>This enables only stimulus port 32 and triggers are only caused on writes to stimulus port 32. All other stimulus ports are disabled and do not cause triggers.</p>

### 2.3.6 STMSPMSCR, Stimulus Port Master Select Configuration Register

The STMSPMSCR characteristics are:

<b>Purpose</b>	Enables a debugger to program which masters the <a href="#">STMSPSCR</a> applies to.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	If only one master is implemented, this register is not implemented and is Reserved.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-6](#) shows the STMSPMSCR bit assignments.

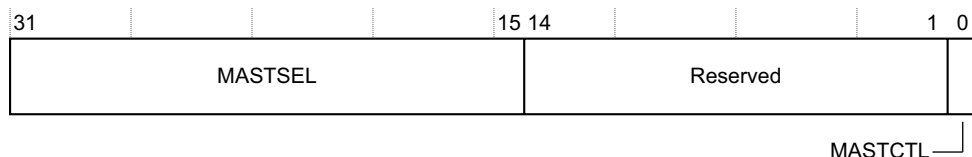


Figure 2-6 STMSPMSCR bit assignments

Table 2-8 shows the STMSPMSCR bit assignments.

**Table 2-8 STMSPMSCR bit assignments**

Bits	Name	Description
[31:15]	MASTSEL	Master Selection. This field defines which master the <a href="#">STMSPSCR</a> applies to. The size of this field is defined by the number of implemented masters. The reset value is UNKNOWN.
[14:1]	-	Reserved, UNK/SBZP.
[0]	MASTCTL	This bit defines how the master is applied: 0b0 Master selection not used. 0b1 Master selection applies to the <a href="#">STMSPSCR</a> . The reset value is 0b0.

### MASTCTL == 0b0

When MASTCTL is 0b0 the port selection used by the [STMSPSCR](#) applies equally to all masters and MASTSEL is ignored.

### MASTCTL == 0b1

When MASTCTL is 0b1, the MASTSEL field enables you to select a subset of the full masters to which the [STMSPSCR](#) applies. MASTSEL enables you to select a single master or power-of-two multiples of consecutive masters to which to apply the [STMSPSCR](#).

To program MASTSEL, the bottom N bits which are 0 define a mask to apply to the master selection, then a 1 in bit N+1 demarks the mask from the master selection. The bits from N+2 to M select the master to which the [STMSPSCR](#) applies.

For example:

**MASTSEL** = bbbb\_bbbb\_bbbb\_bbbb\_1

A single master bbbb\_bbbb\_bbbb\_bbbb is selected.

**MASTSEL** = bbbb\_bbbb\_bbb1\_0000\_0

A selection of 32 masters from bbbb\_bbbb\_bbb0\_0000 to bbbb\_bbbb\_bbb1\_1111 is selected.

**MASTSEL** = 1000\_0000\_0000\_0000\_0

All masters are selected. This is equivalent to MASTCTL == 0b0.

Programming MASTCTL == 1 and MASTSEL = 0000\_0000\_0000\_0000\_0 is UNPREDICTABLE.

Programming a MASTSEL value which enables more masters than are implemented results in UNPREDICTABLE behavior. For example, programming 1000\_0000\_0000\_0000\_0 when only 32 masters are implemented. To enable all 32 masters program 0000\_0000\_0001\_0000\_0.

## Using MASTCTL

Table 2-9 shows how to use MASTCTL.

Table 2-9 Using MASTCTL

MASTCTL	Description
0b0	<p>Master selection select is not used.</p> <p><a href="#">STMSPSCR</a> applies equally to every master. MASTSEL is ignored.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>MASTCTL is 0b0.</li> <li><a href="#">STMSPSCR</a>.PORTSEL is 0b00.</li> <li>Only bit[0] of the <a href="#">STMSPER</a> is 0b1.</li> <li>Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> </ul> <p>This enables stimulus ports 0, 32, 64, 96, 128, and so on, on all masters. Triggers are caused on writes to stimulus ports 0, 32, 64, 96, 128, and so on, on all masters. All other stimulus ports on all masters are disabled and do not cause triggers.</p>
0b1	<p>Master selection applies to <a href="#">STMSPSCR</a>.</p> <p><a href="#">STMSPSCR</a> only applies to the masters selected by MASTSEL. Other masters are not enabled and do not cause triggers.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>MASTCTL is 0b1.</li> <li>MASTSEL is b0000_0000_0000_0001_1 (select master 1).</li> <li><a href="#">STMSPSCR</a>.PORTCTL is 0b11.</li> <li><a href="#">STMSPSCR</a>.PORTSEL is b000_0000_0001_1 (select group 1).</li> <li>Only bit[0] of the <a href="#">STMSPER</a> is 0b1.</li> <li>Only bit[0] of the <a href="#">STMSPTER</a> is 0b1.</li> </ul> <p>This enables only stimulus port 32 on master 1 and triggers are only caused on writes to stimulus port 32 on master 1. All other stimulus ports on all masters are disabled and do not cause triggers.</p>

### 2.3.7 STMSPOVERRIDER, Stimulus Port Override Register

The STMSPOVERRIDER characteristics are:

<b>Purpose</b>	Enables a debugger to override various features of the STM. This register is used in conjunction with <a href="#">STMSPMOVERRIDER</a> .
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is optional. Read <a href="#">STMFEAT2R</a> to determine if it is implemented.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-7 shows the STMSPOVERRIDER bit assignments.

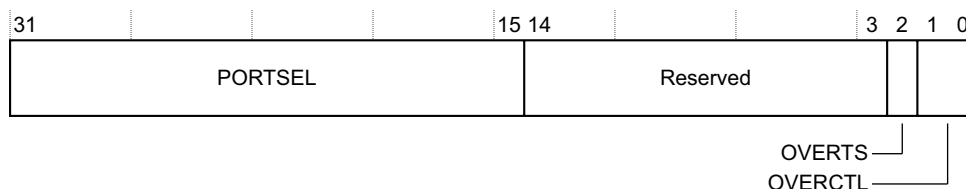


Figure 2-7 STMSPOVERRIDER bit assignments



Table 2-10 shows the STMSPOVERRIDER bit assignments.

**Table 2-10 STMSPOVERRIDER bit assignments**

Bits	Name	Description
[31:15]	PORTSEL	Port selection. This field defines which stimulus ports the override controls apply to. The size of this field is defined by the number of implemented stimulus ports. The reset value is UNKNOWN.
[14:3]	-	Reserved, UNK/SBZP.
[2]	OVERTS	Timestamping override. This override requests all stimulus port writes that cause trace to be traced with a timestamp (where possible). As with normal operation, this does not ensure all packets are generated with timestamps. This field is independent of OVERCTL and PORTSEL and <a href="#">STMSPMOVERRIDER</a> . 0b0 Override not enabled. 0b1 Override enabled. The reset value is 0b0.
[1:0]	OVERCTL	This defines how the port selection is applied: 0b00 Override controls disabled. 0b01 Ports selected by PORTSEL always behave as guaranteed transactions. 0b10 Ports selected by PORTSEL always behave as invariant timing transactions. 0b11 Reserved. The reset value is 0b00.

### OVERCTL != 0b00

When OVERCTL is not 0b00, the PORTSEL field enables you to select a subset of the full stimulus ports to which the override controls apply. PORTSEL enables you to select a single stimulus ports or power-of-two multiples of consecutive stimulus ports to which to apply the override controls.

To program PORTSEL, the bottom N bits which are 0 define a mask to apply to the port selection, then a 1 in bit N+1 delimits the mask from the port selection. The bits from N+2 to M select the ports to which the override controls apply.

For example:

**PORTSEL** = pppp\_pppp\_pppp\_pppp\_1

A single port pppp\_pppp\_pppp\_pppp is selected.

**PORTSEL** = pppp\_pppp\_ppp1\_0000\_0

A selection of 32 ports from pppp\_pppp\_ppp0\_0000 to pppp\_pppp\_ppp1\_1111 are selected.

**PORTSEL** = 1000\_0000\_0000\_0000\_0

All ports are selected.

Programming OVERCTL != 00 and PORTSEL = 0000\_0000\_0000\_0000\_0 is UNPREDICTABLE.

Programming a PORTSEL value which enables more stimulus ports than are implemented results in UNPREDICTABLE behavior. For example, programming 1000\_0000\_0000\_0000\_0 when only 32 stimulus ports are implemented. To enable all 32 stimulus ports, program 0000\_0000\_0001\_0000\_0.

Using OVERCTL

Table 2-11 shows how to use OVERCTL.

Table 2-11 Using OVERCTL

OVERCTL	Description
0b00	Override controls disabled. PORTSEL is ignored.
0b01	Ports selected by PORTSEL always behave as guaranteed transactions. For example, PORTSEL is b0000_0000_0000_0000_1, selecting port 0. All stimulus port writes to stimulus port 0 behave as guaranteed transactions. Writes to other stimulus ports are treated as they would normally behave. For example, PORTSEL is b0000_0000_0011_0000_0, selecting ports 32-63. All stimulus port writes to stimulus ports 32-63 behave as guaranteed transactions. Writes to other stimulus ports are treated as they would normally behave.
0b10	Ports selected by PORTSEL always behave as invariant timing transactions. For example, PORTSEL is b0000_0000_0000_0000_1, selecting port 0. All stimulus port writes to stimulus port 0 behave as invariant timing transactions. Writes to other stimulus ports are treated as they would normally behave. For example, PORTSEL is b0000_0000_0011_0000_0, selecting ports 32-63. All stimulus port writes to stimulus ports 32-63 behave as invariant timing transactions. Writes to other stimulus ports are treated as they would normally behave.
0b11	Reserved.

2.3.8 STMSPMOVERRIDER, Stimulus Port Master Override Register

The STMSPMOVERRIDER characteristics are:

- Purpose** Enables a debugger to select which masters the STMSPMOVERRIDER applies to.
- Usage constraints** There are no usage constraints.
- Configurations** This register is optional. Read STMFEAT2R to determine if it is implemented.
- Attributes** See the register summary in Table 2-1 on page 2-15.

Figure 2-8 shows the STMSPMOVERRIDER bit assignments.

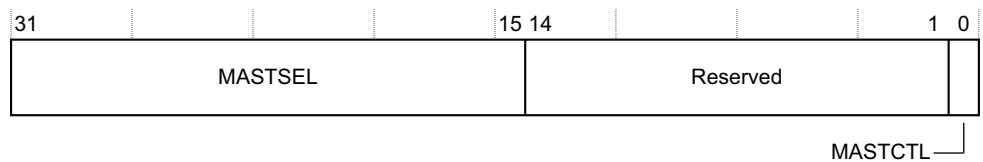


Figure 2-8 STMSPMOVERRIDER bit assignments

Table 2-12 shows the STMSPMOVERRIDER bit assignments.

**Table 2-12 STMSPMOVERRIDER bit assignments**

Bits	Name	Description
[31:15]	MASTSEL	Master selection. This field defines which master the override controls apply to. The size of this field is defined by the number of implemented masters. The reset value is UNKNOWN.
[14:1]	-	Reserved, UNK/SBZP.
[0]	MASTCTL	This bit defines how the master selection is applied: 0b0 Master selection not enabled. <a href="#">STMSPOVERRIDER</a> applies equally to all masters. 0b1 Master selection enabled. <a href="#">STMSPOVERRIDER</a> applies to the masters selected by MASTSEL. The reset value is 0b0.

### MASTCTL == 0b0

When MASTCTL is 0b0 the override controls used by the [STMSPOVERRIDER](#) apply equally to all masters and MASTSEL is ignored.

### MASTCTL == 0b1

When MASTCTL is 0b1, the MASTSEL field enables you to select a subset of the full masters to which the [STMSPOVERRIDER](#) applies. MASTSEL enables you to select a single master or power-of-two multiples of consecutive masters to which to apply the [STMSPOVERRIDER](#).

To program MASTSEL, the bottom N bits which are 0 define a mask to apply to the master selection, then a 1 in bit N+1 demarks the mask from the master selection. The bits from N+2 to M select the master to which the [STMSPOVERRIDER](#) applies.

For example:

**MASTSEL** = bbbb\_bbbb\_bbbb\_bbbb\_1

A single master bbbb\_bbbb\_bbbb\_bbbb is selected.

**MASTSEL** = bbbb\_bbbb\_bbb1\_0000\_0

A selection of 32 masters from bbbb\_bbbb\_bbb0\_0000 to bbbb\_bbbb\_bbb1\_1111 is selected.

**MASTSEL** = 1000\_0000\_0000\_0000\_0

All masters are selected. This is equivalent to MASTCTL == 0b0.

Programming MASTCTL == 1 and MASTSEL = 0000\_0000\_0000\_0000\_0 is UNPREDICTABLE.

Programming a MASTSEL value which enables more masters than are implemented results in UNPREDICTABLE behavior. For example, programming 1000\_0000\_0000\_0000\_0 when only 32 masters are implemented. To enable all 32 masters, program 0000\_0000\_0001\_0000\_0.

## Using MASTCTL

Table 2-13 shows how to use MASTCTL.

**Table 2-13 Using MASTCTL**

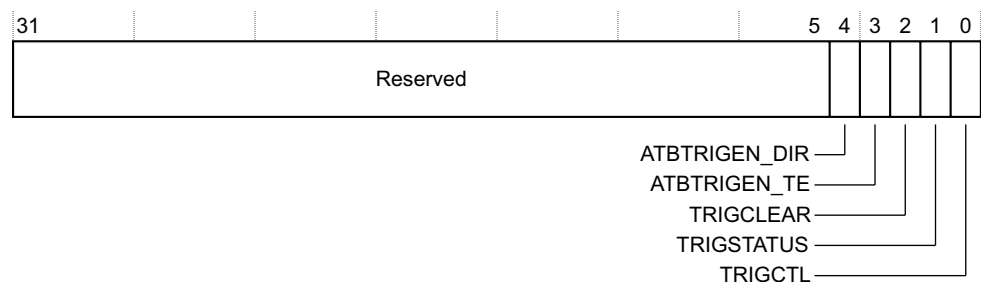
MASTCTL	Description
0b0	Master selection for override controls disabled and <a href="#">STMSPOVERRIDER</a> applies equally to all masters. MASTSEL is ignored.
0b1	<p>The <a href="#">STMSPOVERRIDER</a> applies to the masters selected by MASTSEL.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>MASTSEL is b0000_0000_0000_0001_1, selecting master 1.</li> <li><a href="#">STMSPOVERRIDER.OVERCTL</a> is 0b01.</li> <li><a href="#">STMSPOVERRIDER.PORTSEL</a> is b0000_0000_0000_0001_1, selecting port 1.</li> </ul> <p>All stimulus port writes to stimulus port 1 on master 1 behave as guaranteed transactions. Writes to other stimulus ports on all other masters are treated as they would normally behave.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>MASTSEL is b0000_0000_0000_0011_0, selecting masters 2-3.</li> <li><a href="#">STMSPOVERRIDER.OVERCTL</a> is 0b10.</li> <li><a href="#">STMSPOVERRIDER.PORTSEL</a> is b0000_0000_0011_0001_1, selecting ports 32-63.</li> </ul> <p>All stimulus port writes to stimulus ports 32-63 on masters 2 and 3 behave as invariant timing transactions. Writes to other stimulus ports on all other masters are treated as they would normally behave.</p>

### 2.3.9 STMSPTRIGCSR, Stimulus Port Trigger Control and Status Register

The STMSPTRIGCSR characteristics are:

<b>Purpose</b>	Controls the STM triggers caused by the <a href="#">STMSPTER</a> .
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is optional. Read <a href="#">STMFEAT1R</a> to determine if it is implemented.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-9 shows the STMSPTRIGCSR bit assignments.



**Figure 2-9 STMSPTRIGCSR bit assignments**

Table 2-14 shows the STMSPTRIGCSR bit assignments.

**Table 2-14 STMSPTRIGCSR bit assignments**

Bits	Type	Name	Description
[31:5]	-	-	Reserved, UNK/SBZP.
[4]	RW	ATBTRIGEN_DIR	ATB trigger enable on direct writes to TRIG locations in an Extended Stimulus Port. When set, this bit enables the STM to use the ATID value of 0x7D when software writes to the TRIG locations. See <a href="#">Triggers on page 2-46</a> for more information. The reset value is 0b0.
[3]	RW	ATBTRIGEN_TE	ATB trigger enable on writes to Stimulus Ports being monitored using the <a href="#">STMSPTER</a> . When set, this bit enables the STM to use the ATID value of 0x7D when software writes to an enabled Stimulus Port. See <a href="#">Triggers on page 2-46</a> and <a href="#">STMSPTER, Stimulus Port Trigger Enable Register on page 2-18</a> for more information. The reset value is 0b0.
[2]	WO	TRIGCLEAR	When TRIGCTL indicates single-shot mode, this bit is used to clear TRIGSTATUS: 0b0 No effect. 0b1 Clears TRIGSTATUS if TRIGSTATUS is 0b1. Writing a 0b1 to this bit when in multi-shot mode is Unpredictable.
[1]	RO	TRIGSTATUS	When TRIGCTL indicates single-shot mode, this bit indicates whether the single trigger has occurred: 0b0 Trigger has not occurred. 0b1 Trigger has occurred. In multi-shot mode this bit is always UNK/SBZP.
[0]	RW	TRIGCTL	Trigger control: 0b0 Triggers are multi-shot. 0b1 Triggers are single-shot. See <a href="#">Triggers on page 2-46</a> for more information. The reset value is 0b0.

### 2.3.10 STMTCSR, Trace Control and Status Register

The STMTCSR characteristics are:

<b>Purpose</b>	Controls the STM settings.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-10 on page 2-30](#) shows the STMTCSR bit assignments.

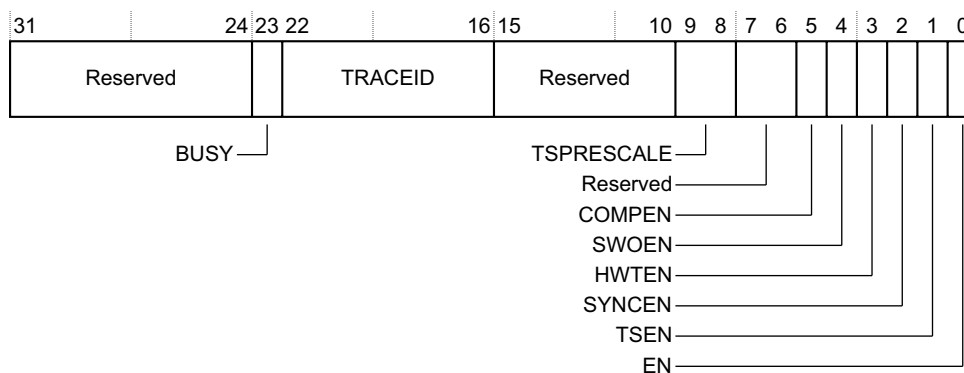


Figure 2-10 STMTCSR bit assignments

Table 2-15 shows the STMTCSR bit assignments.

Table 2-15 STMTCSR bit assignments

Bits	Type	Name	Description
[31:24]	-	-	Reserved, UNK/SBZP.
[23]	RO	BUSY	STM is busy, for example the STM trace FIFO is not empty. The reset value is IMPLEMENTATION SPECIFIC.
[22:16]	RW <sup>a</sup>	TRACEID	<b>TRACEID[6:0]</b> value. The reset value is UNKNOWN.
[15:10]	-	-	Reserved, UNK/SBZP.
[9:8]	RW <sup>a</sup>	TSPRESCALE	Timestamp prescaler. The reference clock source is selected by SWOEN: <div> <div>0b00</div> <div>No prescaling.</div> <div>0b01</div> <div>Divide by 4.</div> <div>0b10</div> <div>Divide by 16.</div> <div>0b11</div> <div>Divide by 64.</div> </div> The reset value is 0b00.
[7:6]	-	-	Reserved, UNK/SBZP.
[5]	RW <sup>b</sup>	COMPEN	Compression enable for stimulus ports: <div> <div>0b0</div> <div>Compression disabled, data transfers are transmitted at the size of the transaction.</div> <div>0b1</div> <div>Compression enabled, data transfers are compressed to save bandwidth.</div> </div> The reset value is 0b0.
[4]	RW <sup>a</sup>	SWOEN	Enables asynchronous-specific usage model for timestamps, when TSEN == 0b1: <div> <div>0b0</div> <div>Timestamp counter uses a system clock and counts continuously.</div> <div>0b1</div> <div>Timestamp counter uses a clock from an external trace output interface. The timestamp counter is held in reset while the trace output line is idle.</div> </div> The reset value is 0b0.
[3]	RW <sup>a</sup>	HWTEN	Enable hardware event trace packet emission. The reset value is 0b0.

**Table 2-15 STMTCSR bit assignments (continued)**

Bits	Type	Name	Description
[2]	RW <sup>ac</sup>	SYNCEN	Enable synchronization packets. Synchronization period is defined by the <a href="#">STMSYNCR</a> , if implemented, or by another IMPLEMENTATION DEFINED mechanism. The reset value is 0b0 <sup>c</sup> .
[1]	RW <sup>a</sup>	TSEN	Enable timestamps. Timestamp behavior might be qualified by SWOEN. When this bit is zero no timestamps are generated and, when using STPv2, FREQ packets are not generated. The reset value is 0b0.
[0]	RW	EN	Global STM enable. Always present. The reset value is 0b0.

- These bits are optional. To determine which bits are implemented, read [STMFEAT1R](#), or write each bit with a value of 0b1 and read back. If the value returned is 0b1, the bit is implemented. Only perform this when [STMTCSR.EN](#) is 0b0. For more information on recommended configurations, see [Appendix A](#).
- These bits are optional. The [STMFEAT1R](#) and [STMFEAT2R](#) identify the presence of these bits.
- The [STMTCSR.SYNCEN](#) bit is not always implemented as RW. When the [STMSYNCR](#) register is implemented, this bit is RO and always reads as 0b1.

To avoid trace stream corruption, the STM must be disabled with [STMTCSR.EN](#) == 0b0 and the [STMTCSR.BUSY](#) bit polled until it is 0b0 before [STMTCSR.TRACEID](#) is modified.

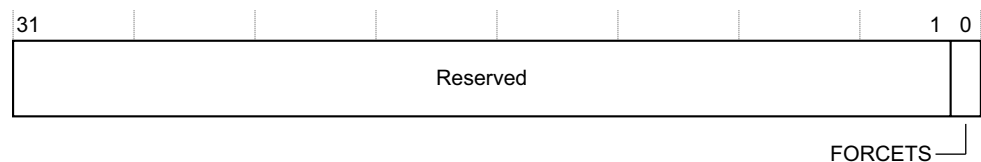
To ensure that all writes to the STM stimulus ports are traced before disabling the STM, ARM recommends that software writes to the stimulus port then reads from any stimulus port before clearing [STMTCSR.EN](#). This is only required if the same piece of software is writing to the stimulus ports and disabling the STM.

### 2.3.11 STMTSSTIMR, Timestamp Stimulus Register

The STMTSSTIMR characteristics are:

<b>Purpose</b>	Forces the next packet caused by a stimulus port write to have a timestamp output.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is only implemented if the <a href="#">STMFEAT1R.FORCETS</a> bit is set, otherwise it ignores writes.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-11](#) shows the STMTSSTIMR bit assignments.



**Figure 2-11 STMTSSTIMR bit assignments**

Table 2-16 shows the STMTSSTIMR bit assignments.

### Table 2-16 STMTSSTIMR bit assignments

Bits	Type	Name	Description
[31:1]	-	-	Reserved, UNK/SBZP.
[0]	WO	FORCETS	Force timestamp stimulus. A write to this register with this bit as 0b1 requests the next stimulus port write which causes trace to be upgraded to have a timestamp. Writes with this bit 0b0 are ignored.

If timestamping is not enabled, that is, when `STMTCSR.TSEN == 0b0`, writes to this register are ignored.

Implementations are allowed to ignore the timestamp indication on a stimulus port write, for example, if there is insufficient buffer space to trace the timestamp. However, the timestamp request initiated by writes to this register is persistent until a trace packet with a timestamp is generated.

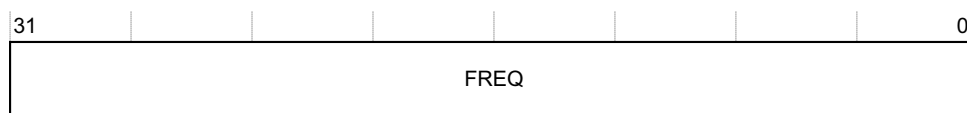
The timestamp request initiated by writes to this register is persistent except through a reset of the STM. This means that disabling and re-enabling the STM using [STMTCR.EN](#) does not clear this request.

### 2.3.12 STMTSFREQR, Timestamp Frequency Register

The STMTSFREQR characteristics are:

<b>Purpose</b>	Indicates the frequency of the timestamp counter. The unit of measurement is increments per second.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is only implemented when <a href="#">STMFEAT1R.PROT</a> indicates STPv2 is implemented.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-12 shows the STMTSFREOR bit assignments.



### Figure 2-12 STMTSFREQR bit assignments

Table 2-17 shows the STMTSFREQR bit assignments.

### Table 2-17 STMTSFREQR bit assignments

Bits	Type	Name	Description
31:0	IMPDEF	FREQ	The timestamp frequency in Hz. The reset value is IMPLEMENTATION DEFINED.

If timestamping is enabled, writing to this register causes a FREQ or FREQ\_TS packet to be generated, indicating the new timestamp frequency. A value of zero indicates the timestamp frequency is not known.

This register might be read-only in some implementations. In read-only implementations, the reset value indicates the timestamp frequency. In read/write implementations software must program this with the frequency of the timestamp clock, although the reset value might also indicate the initial value of the timestamp frequency.

The presence and configuration of this register is defined in the [STMFEAT1R](#) register.



### 2.3.13 STMSYNCR, Synchronization Control Register

The STMSYNCR characteristics are:

<b>Purpose</b>	Controls the interval between synchronization packets, in terms of the number of bytes of trace generated. This register only provides a hint of the desired synchronization frequency, because implementations are permitted to be inaccurate.
----------------	---

Writing a value of 0x00000000 to this register disables the synchronization counter, however any other IMPLEMENTATION DEFINED synchronizations mechanism continue to operate independently.

When this register is written, the STM must perform synchronization immediately if enabled, and reset the count value to the newly programmed value immediately, ensuring subsequent synchronization occurs in the desired period.

<b>Usage constraints</b>	There are no usage constraints.
--------------------------	---------------------------------

**Configurations** This register is optional. Read [STMFEAT1R](#) to determine if it is implemented.

**Attributes** See the register summary in [Table 2-1 on page 2-15](#).

Figure 2-13 shows the STMSYNCR bit assignments.



**Figure 2-13 STMSYNCR bit assignments**

Table 2-18 shows the STMSYNCR bit assignments.

### Table 2-18 STMSYNCR bit assignments

Bits	Name	Description
[31:13]	-	Reserved, UNK/SBZP
[12]	MODE	<p>Mode control:</p> <p>0b0      COUNT[11:0] defines a value N. Synchronization period is N bytes.</p> <p>0b1      COUNT[11:7] defines a value N. Synchronization period is 2<sup>N</sup> bytes. N must be in the range of 12 to 27 inclusive and other values are UNPREDICTABLE.</p> <p>The reset value is 0b0.</p>
[11:0]	COUNT	<p>Counter value for the number of bytes between synchronization packets. Reads return the value of this register.</p> <p>The reset value is IMPLEMENTATION DEFINED.</p>

To determine if this register is implemented, read the [STMFEAT1R.SYNC](#) field. If [STMFEAT1R.SYNC](#) returns 0b00, write the value 0x00001FFF to this register and read it back. If the returned value is zero this register is not implemented, otherwise the register is implemented.

Some lower-order bits of **STMSYNCR.COUNT** might not be implemented. This can be determined when reading back the value after writing 0x00001FFF.

### 2.3.14 STMAUXCR, Auxiliary Control Register

The STMAUXCR characteristics are:

**Purpose** Used for IMPLEMENTATION DEFINED STM controls. The contents of the register are IMPLEMENTATION DEFINED. Setting any bits in this register to anything other than 0b0 might result in behavior which contravenes this architecture.

<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Table 2-19](#) shows the STMAUXCR bit assignments.

**Table 2-19 STMAUXCR bit assignments**

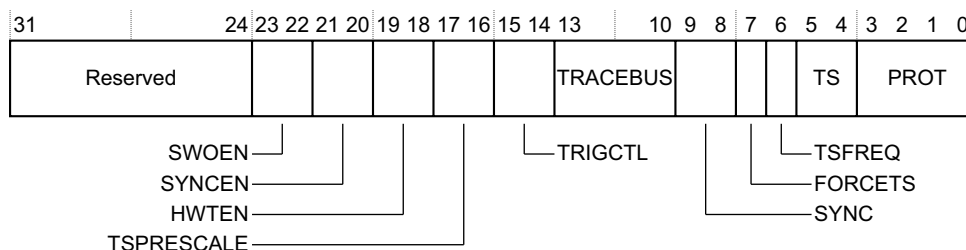
Bits	Name	Description
[31:0]	-	IMPLEMENTATION DEFINED. The reset value is 0b0.

### 2.3.15 STMFEAT1R, Features 1 Register

The STMFEAT1R characteristics are:

<b>Purpose</b>	Indicates the features of the STM.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-14](#) shows the STMFEAT1R bit assignments.



**Figure 2-14 STMFEAT1R bit assignments**

[Table 2-20](#) shows the STMFEAT1R bit assignments.

**Table 2-20 STMFEAT1R bit assignments**

Bits	Name	Description
[31:24]	-	Reserved, RAZ.
[23:22]	SWOEN	<a href="#">STMTCR.SWOEN</a> support: 0b00 Support not defined here. Support for <a href="#">STMTCR.SWOEN</a> can be detected by direct access to the <a href="#">STMTCR</a> . 0b01 <a href="#">STMTCR.SWOEN</a> not implemented. 0b10 <a href="#">STMTCR.SWOEN</a> implemented.
[21:20]	SYNCEN	<a href="#">STMTCR.SYNCEN</a> support: 0b00 Support not defined here. Support for <a href="#">STMTCR.SYNCEN</a> can be detected by direct access to the <a href="#">STMTCR</a> . 0b01 <a href="#">STMTCR.SYNCEN</a> not implemented and always reads as 0b0. 0b10 <a href="#">STMTCR.SYNCEN</a> implemented but always reads as 0b1. 0b11 <a href="#">STMTCR.SYNCEN</a> implemented and is writeable.

**Table 2-20 STMFEAT1R bit assignments (continued)**

Bits	Name	Description
[19:18]	HWTEN	<a href="#">STMTCSR.HWTEN</a> support:
		0b00 Support not defined here. Support for <a href="#">STMTCSR.HWTEN</a> can be detected by direct access to the <a href="#">STMTCSR</a> .
		0b01 <a href="#">STMTCSR.HWTEN</a> not implemented.
		0b10 <a href="#">STMTCSR.HWTEN</a> implemented.
[17:16]	TSPRESCALE	Timestamp prescale support:
		0b00 Support not defined here. Support for timestamp prescaling can be detected by direct access to the <a href="#">STMTCSR</a> .
		0b01 Timestamp prescale not implemented.
		0b10 Timestamp prescale implemented.
[15:14]	TRIGCTL	Trigger control support:
		0b00 Trigger support not defined here.
		0b01 Multi-shot triggers supported only.
		0b10 Multi-shot and single-shot triggers supported. <a href="#">STMSPTRIGCSR.TRIGCTL</a> implemented.
[13:10]	TRACEBUS	Trace bus support:
		0b0000 CoreSight ATB implemented. <a href="#">STMTCSR.TRACEID</a> implemented.
		0b0001 CoreSight ATB plus ATB trigger support implemented. <a href="#">STMTCSR.TRACEID</a> and <a href="#">STMSPTRIGCSR.ATBTRIGEN_DIR</a> and <a href="#">STMSPTRIGCSR.ATBTRIGEN_TE</a> implemented.
[9:8]	SYNC	STMSYNCR support:
		0b00 Support not defined here. Support for the <a href="#">STMSYNCR</a> can be detected by direct access to the <a href="#">STMSYNCR</a> .
		0b01 <a href="#">STMSYNCR</a> not implemented.
		0b10 <a href="#">STMSYNCR</a> implemented without MODE control.
		0b11 <a href="#">STMSYNCR</a> implemented with MODE control.
[7]	FORCETS	<a href="#">STMTSSTIMR</a> support:
		0b0 <a href="#">STMTSSTIMR</a> bit[0] not implemented.
		0b1 <a href="#">STMTSSTIMR</a> bit[0] implemented.
[6]	TSFREQ	Timestamp frequency indication configuration:
		0b0 <a href="#">STMTSFREQR</a> is read-only.
		0b1 <a href="#">STMTSFREQR</a> is read-write.
[5:4]	TS	Timestamp support:
		0b00 Differential timestamps implemented.
		0b01 Absolute timestamps implemented.
		0b10 Timestamping not implemented.
[3:0]	PROT	Protocol type:
		0b0001 STPv2.

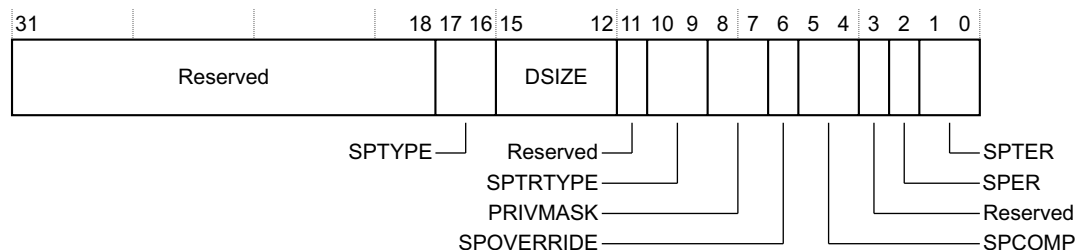
Unspecified encodings of fields in this register are Reserved.

### 2.3.16 STMFEAT2R, Features 2 Register

The STMFEAT2R characteristics are:

<b>Purpose</b>	Indicates the features of the STM.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-15](#) shows the STMFEAT2R bit assignments.



**Figure 2-15 STMFEAT2R bit assignments**

[Table 2-21](#) shows the STMFEAT2R bit assignments.

**Table 2-21 STMFEAT2R bit assignments**

Bits	Name	Description
[31:18]	-	Reserved, RAZ.
[17:16]	SPTYPE	Stimulus Port type support: 0b00 Only Basic Stimulus Ports implemented. 0b01 Only Extended Stimulus Ports implemented. 0b10 Both Basic and Extended Stimulus Ports implemented.
[15:12]	DSIZE	Fundamental data size: 0b0000 32-bit data. 0b0001 64-bit data.
[11]	-	Reserved, RAZ.
[10:9]	SPTRTYPE	Stimulus Port Transaction Type support: 0b00 Only invariant timing transactions are supported. 0b01 Only guaranteed transactions are supported. 0b10 Both invariant timing and guaranteed transactions are supported.
[8:7]	PRIVMASK	<a href="#">STMPRIVMASKR</a> support: 0b00 <a href="#">STMPRIVMASKR</a> support not defined here. Support for the <a href="#">STMPRIVMASKR</a> can be detected by direct access to the <a href="#">STMPRIVMASKR</a> . 0b01 <a href="#">STMPRIVMASKR</a> not implemented. 0b10 <a href="#">STMPRIVMASKR</a> implemented.
[6]	SPOVERRIDE	<a href="#">STMSPOVERRIDER</a> and <a href="#">STMSPMOVERRIDER</a> support: 0b0 <a href="#">STMSPOVERRIDER</a> and <a href="#">STMSPMOVERRIDER</a> not implemented. 0b1 <a href="#">STMSPOVERRIDER</a> and <a href="#">STMSPMOVERRIDER</a> implemented.

**Table 2-21 STMFEAT2R bit assignments (continued)**

Bits	Name	Description
[5:4]	SPCOMP	Data compression on stimulus ports support: 0b00      Data compression support is not defined here. Use the part number of the device to determine if data compression is supported. 0b01      No data compression supported. 0b10      Data compression always enabled. 0b11      Data compression support is programmable. STMTCSR.COMPEN is implemented.
[3]	-	Reserved, RAZ.
[2]	SPER	<a href="#">STMSPER</a> presence: 0b0 <a href="#">STMSPER</a> is implemented.
[1:0]	SPTER	<a href="#">STMSPTER</a> support: 0b00 <a href="#">STMSPTER</a> presence is not indicated here, check the <a href="#">STMSPTER</a> . 0b01 <a href="#">STMSPTER</a> is not implemented. 0b10 <a href="#">STMSPTER</a> is implemented.

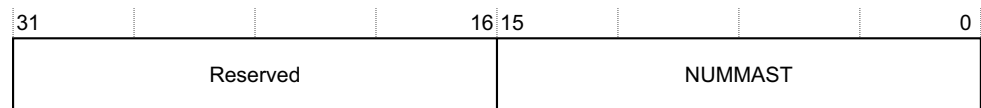
Unspecified encodings of fields in this register are Reserved.

### 2.3.17 STMFEAT3R, Features 3 Register

The STMFEAT3R characteristics are:

<b>Purpose</b>	Indicates the features of the STM.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-16](#) shows the STMFEAT3R bit assignments.



**Figure 2-16 STMFEAT3R bit assignments**

[Table 2-22](#) shows the STMFEAT3R bit assignments.

**Table 2-22 STMFEAT3R bit assignments**

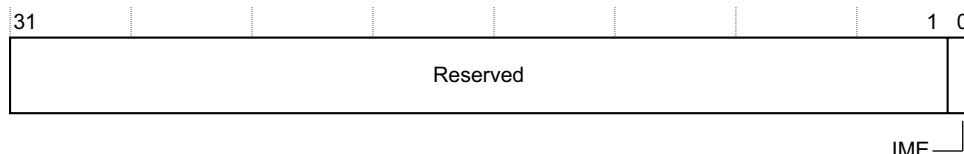
Bits	Name	Description
[31:16]	-	Reserved, UNK/SBZP
[15:0]	NUMMAST	The number of stimulus port masters implemented, minus 1. For example: 0x0000      1 master implemented. 0x00FF      256 masters implemented.

### 2.3.18 STMITCTRL, Integration Mode Control Register

The STMITCTRL characteristics are:

<b>Purpose</b>	Controls whether the STM is in integration mode.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-17](#) shows the STMITCTRL bit assignments.



**Figure 2-17 STMITCTRL bit assignments**

[Table 2-23](#) shows the STMITCTRL Register bit assignments.

**Table 2-23 STMITCTRL bit assignments**

Bits	Name	Description
[31:1]	-	Reserved, UNK/SBZP.
[0]	IME	When 0b1, the STM is in integration mode. The reset value is 0b0.

This register must only be programmed with a value of 0b1 when [STMTCSR.EN](#) is 0b0.

This presence of this register is IMPLEMENTATION DEFINED. Writing 0b1 to [STMITCTRL.IME](#) and reading the value back can determine the presence. If the returned value has [STMITCTRL.IME](#) 0b1, the register is present.

### 2.3.19 Claim Tag Registers

The claim tag mechanism enables multiple agents to arbitrate over access control to the STM configuration registers. For example, in a system where multiple processors all use the same STM and each processor has separate hardware events which are connected to the STM, each processor might need to independently control the configuration of its hardware events. The claim tag mechanism enables each processor to attempt to claim access to the STM configuration registers so that it can reconfigure the STM without risk of other processors corrupting the configuration.

#### **Note**

The claim tag mechanism does not prevent access to any registers, it merely acts as an arbitration mechanism.

The claim tag registers have an IMPLEMENTATION DEFINED number of claim tag bits, typically one per agent. If an agent requires access to the configuration registers, the agent must set its relevant claim tag bit using the [STMCLAIMSET](#) register. It must then read the status of the claim tag and, if its own bit is the only bit which is set, it has then claimed access. If any other bits are set, this agent has not necessarily claimed access and must clear its bit using the [STMCLAIMCLR](#) register and attempt the process again.

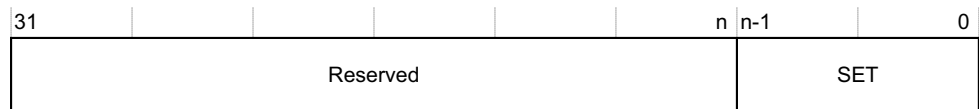
At least four claim tag bits are implemented.

## STMCLAIMSET, Claim Tag Set Register

The STMCLAIMSET characteristics are:

<b>Purpose</b>	On writes this register sets bits of the claim tag. On reads it indicates the number of claim tag bits implemented.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-18](#) shows the STMCLAIMSET bit assignments.



**Figure 2-18 STMCLAIMSET bit assignments**

[Table 2-24](#) shows the STMCLAIMSET Register bit assignments.

**Table 2-24 STMCLAIMSET bit assignments**

Bits	Name	Description
[31:n]	-	Reserved, UNK/SBZP.
[n-1:0]	SET	On reads, each bit reads as 0b1 if the claim tag bit is implemented. For example if four claim tag bits are implemented, this register reads as 0xF. On writes, a 0b1 in a bit position causes the corresponding claim tag bit to be set.

## STMCLAIMCLR, Claim Tag Clear Register

The STMCLAIMCLR characteristics are:

<b>Purpose</b>	On writes this register clears bits of the claim tag. On reads it indicates the current status of the claim tag.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-19](#) shows the STMCLAIMCLR bit assignments.



**Figure 2-19 STMCLAIMCLR bit assignments**

Table 2-25 shows the STMCLAIMCLR register bit assignments.

### Table 2-25 STMCLAIMCLR bit assignments

Bits	Name	Description
[31:n]	-	Reserved, UNK/SBZP.
[n-1:0]	CLR	On reads, each bit reads as one if the claim tag bit is set. On writes, a 0b1 in a bit position causes the corresponding claim tag bit to be cleared. On a reset the claim tags are reset to 0b0.

### 2.3.20 Lock Registers

The lock mechanism controls memory-mapped software access to all configuration registers except for the [STMLAR](#).

If you lock the STM using this feature, it ignores memory-mapped software writes to configuration registers. Memory-mapped debugger accesses and all reads are unaffected. The basic stimulus ports and extended stimulus ports are not affected by the lock mechanism.

To write to the configuration registers, the on-chip software that accesses the STM must access the STM registers as follows:

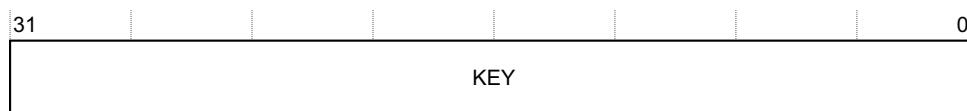
1. Unlock the STM by writing 0xC5ACCE55 to the [STMLAR](#).
2. Access the other STM configuration registers.
3. Lock the STM by writing any other value, for example 0x0, to the [STMLAR](#).

## STMLAR, Lock Access Register

The STMLAR characteristics are:

<b>Purpose</b>	Locks or unlocks write access to the other configuration registers.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-20 shows the STMLAR bit assignments.



### Figure 2-20 STMLAR bit assignments

Table 2-26 shows the STMLAR bit assignments.

### Table 2-26 STMLAR bit assignments

Bits	Name	Description
[31:0]	KEY	Write a value of 0xC5ACCE55 to unlock access to the configuration registers. Write a value which is not 0xC5ACCE55 to lock access to the configuration registers.

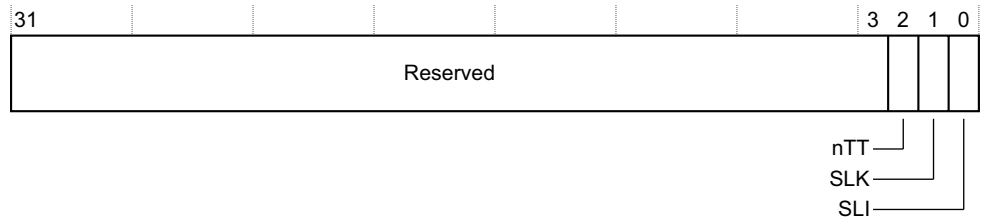


## STMLSR, Lock Status Register

The STMLSR characteristics are:

<b>Purpose</b>	Indicates the status of the lock mechanism.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Figure 2-21](#) shows the STMLSR bit assignments.



**Figure 2-21 STMLSR bit assignments**

[Table 2-27](#) shows the STMLSR bit assignments.

**Table 2-27 STMLSR bit assignments**

Bits	Name	Description
[31:3]	-	Reserved, UNK/SBZP
[2]	nTT	RAZ. Indicates that the <a href="#">STMLAR</a> is 32 bits.
[1]	SLK	Indicates whether the STM configuration registers are locked: 0b0 Writes to the configuration registers are permitted. 0b1 STM is locked. Writes to the configuration registers are ignored. If this register is accessed from an interface where the lock mechanism is ignored, for example, an external debugger, this field reads as 0b0 regardless of whether the STM is locked. The reset value of this bit is 0b1 for accesses from interfaces where the lock mechanism is required.
[0]	SLI	Indicates whether the lock mechanism is implemented for this interface: 0b0 This access is from an interface that ignores the lock mechanism. The Locked bit reads as 0b0 and writes to the <a href="#">STMLAR</a> are ignored. 0b1 This access is from an interface that requires the STM to be unlocked.

### 2.3.21 STMAUTHSTATUS, Authentication Status Register

This read-only register returns the authentication status values for the four different debug types. This register is defined by the CoreSight Architecture.

See [Authentication control on page 2-49](#) for more information.

### 2.3.22 STMDEVARCH, Device Architecture Register

The STMDEVARCH characteristics are:

<b>Purpose</b>	Identifies the architect and architecture of a CoreSight component. The architect might differ from the designer of a component, for example ARM defines the architecture but another company designs and implements the component.
----------------	---

Usage constraints	There are no usage constraints.
Configurations	This register is either: <ul style="list-style-type: none"><li>Not present, and all bits read as zero.</li><li>Present, as indicated by PRESENT, bit[20].</li></ul>
Attributes	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-22 shows the STMDEVARCH bit assignments.

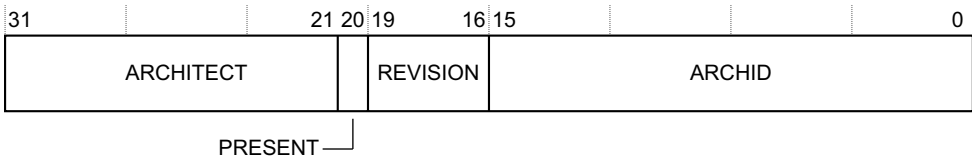


Figure 2-22 STMDEVARCH bit assignments

Table 2-28 shows the STMDEVARCH bit assignments.

Table 2-28 STMDEVARCH bit assignments

Bits	Name	Description
[31:21]	ARCHITECT	Defines the architect of the component: Always takes the value 0x23B, because ARM is the architect of this architecture.
[20]	PRESENT	Indicates the presence of this register: 0b0 STMDEVARCH is not present so bits[31:0] read as zero. 0b1 STMDEVARCH is present.
[19:16]	REVISION	Architecture revision: 0b0000 STMv1.0. 0b0001 STMv1.1.
[15:0]	ARCHID	Architecture ID: Always takes the value 0x0A63, indicating STM Architecture version 1.

2.3.23 STMDEVID, Device Configuration Register

The STMDEVID characteristics are:

Purpose	Controls the number of stimulus ports implemented.
Usage constraints	There are no usage constraints.
Configurations	This register is available in all implementations.
Attributes	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-23 shows the STMDEVID bit assignments.

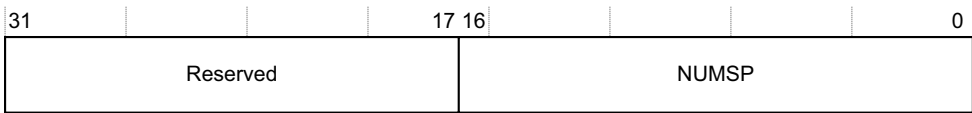


Figure 2-23 STMDEVID bit assignments

Table 2-29 shows the STMDEVID bit assignments.

### Table 2-29 STMDEVID bit assignments

Bits	Name	Description
[31:17]	-	Reserved, UNK/SBZP.
[16:0]	NUMSP	The number of stimulus ports implemented. For example: <div> <div>0x00020</div> <div>32 stimulus ports implemented</div> </div> <div> <div>0x10000</div> <div>65536 stimulus ports implemented.</div> </div>

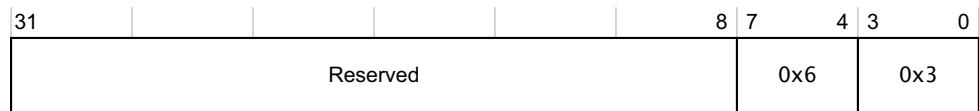
There are 32 stimulus ports if STMDEVID.NUMSP == 0x0000.

### 2.3.24 STMDEVTYPE, Device Type Register

The STMDEVTYPE characteristics are:

<b>Purpose</b>	Returns the device type identifier value.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

Figure 2-24 shows the STMDEVTYPE bit assignments.



**Figure 2-24 STMDEVTYPE bit assignments**

Table 2-30 shows the STMDEVTYPE bit assignments.

### Table 2-30 STMDEVTYPE bit assignments

Bits	Name	Description
[31:8]	-	Reserved, UNK/SBZP
[7:4]	SUB	0x6, indicating the trace is derived from software activity
[3:0]	MAJOR	0x3, indicating the device is a trace source

### 2.3.25 STMPIDR0-7, Peripheral ID Registers

The STMPIDR0-7 characteristics are:

<b>Purpose</b>	Returns the Peripheral ID value. See the <i>ARM Debug Interface v5 Architecture Specification</i> for more information on these registers.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	These registers are available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

### 2.3.26 STMCIDR0-3, Component ID Registers

The STMCIDR0-3 characteristics are:

<b>Purpose</b>	Returns the Component ID value. See the <i>ARM Debug Interface v5 Architecture Specification</i> for more information on these registers.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	These registers are available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 2-1 on page 2-15</a> .

[Table 2-31](#) shows the values for the STMCIDR0-3 registers.

**Table 2-31 STMCIDR0-3 values**

Register	Offset	Value
STMCIDR0	0xFF0	0x0D
STMCIDR1	0xFF4	0x90
STMCIDR2	0xFF8	0x05
STMCIDR3	0xFFC	0xB1

## 2.4 Programming the STM

You do not have to disable the STM to reprogram it. You can modify the following registers while the `STMTCR.EN` bit is 0b1:

- `STMSPER`.
- `STMSPTER`.
- `STMSPSCR`.
- `STMSPMSCR`.
- `STMPRIVMASKR`.
- `STMSPTRIGCSR`.
- `STMSPOVERRIDER`.
- `STMSPMOVERRIDER`.
- `STMTCR`, except `STMTCR.TRACEID` field.
- `STMSYNCR`.
- `STMTSFREQR`.
- CoreSight Management registers.

### 2.4.1 Modifying the `STMSPSCR` and `STMSPMSCR`

Take care when changing the `STMSPSCR` and `STMSPMSCR`, because changes to the `STMSPSCR`, `STMSPMSCR`, `STMSPER`, and `STMSPTER` are not atomic. Certain sequences of changes might result in some stimulus ports being enabled or disabled during the reprogramming process.

For example, when switching from enabling stimulus port 0 to stimulus port 63, both the `STMSPSCR` and `STMSPER` must be modified:

- `STMSPER` from 0x00000001 to 0x80000000.
- `STMSPSCR` from 0x00100001 to 0x00300001.

If you change the `STMSPSCR` first, stimulus port 32 is enabled until the `STMSPER` is modified. Similarly, if you change the `STMSPER` first, stimulus port 31 is enabled until the `STMSPSCR` is modified. ARM recommends that you clear the `STMSPER` to 0x00000000, modify the `STMSPSCR`, and finally modify the `STMSPER` to the required final value.

### 2.4.2 Modifying the `STMSYNCR`

Modifying the `STMSYNCR` when `STMTCR.EN` is 0b1 might not immediately change the synchronization period. The STM might wait until the current synchronization period has finished before recognizing the change to the `STMSYNCR`.

## 2.5 Triggers

Triggers are used to identify points of interest in the trace stream. STPv2 has packets which indicate a trigger has occurred.

The following mechanisms are provided for generating triggers:

- The *Stimulus Port Trigger Enable Register* ([STMSPTER](#)).
- The *Hardware Event Trigger Enable Register* ([STMHETER](#)), if hardware event tracing is implemented.
- Dedicated trigger locations in each extended stimulus port.

Triggers are indicated using one or more of the following mechanisms:

- Dedicated output signals for each cause.
- Insertion of specific trigger packets into the trace stream.
- Insertion of the trigger ATID on an ATB interface.

[Table 2-32](#) shows a summary of trigger generation.

**Table 2-32 Trigger generation summary**

Cause	Outcome		
	Dedicated output asserted	Trigger on ATB	Trigger packet
Match using <a href="#">STMSPTER</a> <sup>a</sup>	Yes <sup>b</sup>	Yes <sup>bc</sup>	No
Match using <a href="#">STMHETER</a> <sup>d</sup>	Yes <sup>e</sup>	Yes <sup>ef</sup>	No
Write to TRIG location <sup>a</sup>	Yes	Yes <sup>c</sup>	Yes

- a. Only on stimulus ports which are enabled for tracing.
- b. In single-shot mode only the first match, controlled by the [STMSPTRIGCSR](#).
- c. Controlled using the [STMSPTRIGCSR](#).
- d. Only on hardware events which are enabled for tracing.
- e. In single-shot mode only the first match, controlled by the [STMHEMCR](#).
- f. Controlled using the [STMHEMCR](#).

The following sections describe triggers in more detail:

- [Triggers caused by matches using the STMSPTER](#).
- [Triggers caused by matches using the STMHETER on page 2-47](#).
- [Triggers caused by writes to TRIG locations in the extended stimulus port on page 2-47](#).

### 2.5.1 Triggers caused by matches using the STMSPTER

For more information on how these triggers are caused, see [STMSPTER, Stimulus Port Trigger Enable Register on page 2-18](#). This mechanism only generates trigger events on a channel which is enabled for tracing.

These triggers operate in one of two modes, single-shot or multi-shot, controlled by the [STMSPTRIGCSR](#).

- In single-shot mode, only the first detected trigger causes a trigger event.
- In multi-shot mode, every detected trigger causes a trigger event.

#### Dedicated output signal

Each trigger event caused by a match using the [STMSPTER](#) asserts a dedicated output signal:

- In single-shot mode, only the first match causes this output signal to be asserted.
- If multiple writes occur in close succession, this signal might not be asserted for every write.

This signal is usually connected to a CoreSight cross trigger network.

Writes to both guaranteed and invariant timing locations cause the output signal to be asserted, regardless of whether the data for that transaction is successfully traced.

### Insertion of trigger packets into the trace stream

Trigger events caused by a match using the [STMSPTER](#) do not cause trigger packets to be inserted into the trace stream.

### Insertion of trigger ATID on an ATB interface

Each trigger event caused by a match using the [STMSPTER](#) causes insertion of the trigger ATID on the ATB interface. This functionality can be controlled using the [STMSPTTRIGCSR](#). In single-shot mode only the first match causes the trigger ATID to be inserted.

Writes to both guaranteed and invariant timing locations cause the trigger ATID to be generated, regardless of whether the data for that transaction is successfully traced.

When this feature is enabled, the STM outputs a single byte ATB transaction with the ATID encoding of 0x7D. The payload of this transaction is always the [STMTCR](#).TRACEID in the lower seven bits. Bit[7] is SBZ.

## 2.5.2 Triggers caused by matches using the STMHETER

For more information on how these triggers are caused, see [STMHETER, Hardware Event Trigger Enable Register on page 4-68](#). This mechanism only generates trigger events on a hardware event which is enabled for tracing.

These triggers operate in one of two modes, single-shot or multi-shot, controlled by the [STMHEMCR](#):

- In single-shot mode, only the first detected trigger causes a trigger event.
- In multi-shot mode, every detected trigger causes a trigger event.

### Dedicated output signal

Each trigger event caused by a match using the [STMHETER](#) asserts a dedicated output signal:

- In single-shot mode, only the first match causes this output signal to be asserted.
- If multiple events occur in close succession, this signal might not be asserted for every event.

This signal is usually connected to a CoreSight cross trigger network.

### Insertion of trigger packets into the trace stream

Trigger events caused by a match using the [STMHETER](#) do not cause trigger packets to be inserted into the trace stream.

### Insertion of trigger ATID on an ATB interface

Each trigger event caused by a match using the [STMHETER](#) causes insertion of the trigger ATID on the ATB interface. This functionality can be controlled using the [STMHEMCR](#). In single-shot mode only the first match causes the trigger ATID to be inserted.

When this feature is enabled, the STM outputs a single byte ATB transaction with the ATID encoding of 0x7D. The payload of this transaction is always the [STMTCR](#).TRACEID in the lower seven bits. Bit[7] is SBZ.

## 2.5.3 Triggers caused by writes to TRIG locations in the extended stimulus port

This section describes triggers generated by writes to the TRIG locations in the extended stimulus ports. See [Chapter 3 Extended Stimulus Ports](#) for more information.

### Dedicated output signal

Each write to a TRIG location asserts a dedicated output signal, if that stimulus port is enabled using the [STMSPER](#). If multiple writes occur in close succession, this signal might not be asserted for every write.

This signal is usually connected to a CoreSight cross trigger network.

### Insertion of trigger packets into the trace stream

Each write to a TRIG location inserts a trigger packet into the trace stream, if that stimulus port is enabled using the [STMSPER](#). All explicit writes to TRIG locations generate a separate trigger packet.

If the write is not traced because the STM cannot produce trace for the transaction, the trigger packet is not generated and a MERR or GERR packet must be generated to indicate this loss.

### Insertion of trigger ATID on an ATB interface

Each write to a TRIG location causes insertion of the trigger ATID on the ATB interface, if that stimulus port is enabled using the [STMSPER](#). This functionality is controlled using [STMSPTRIGCSR](#).

When this feature is enabled, the STM outputs a single byte ATB transaction with the ATID encoding of 0x7D. The payload of this transaction is always the [STMTCSR](#).TRACEID in the lower seven bits. Bit[7] is SBZ.



## 2.6 Authentication control

The CoreSight architecture defines an authentication interface for controlling the permitted level of debug capabilities for a device. It defines three levels of control:

- No debug permitted.
- Only non-invasive debug permitted.
- Invasive and non-invasive debug permitted.

These levels are duplicated for secure and non-secure states, permitting different levels of debug for secure and non-secure states.

The STM is generally considered a non-invasive debug component despite guaranteed transfers causing invasion, because system software chooses the level of invasion. When non-invasive debug is disabled, the STM:

- Treats all stimulus port writes as invariant timing.
- Ignores all stimulus port writes.
- Does not generate any trace.
- Does not generate any triggers.

The *STMSPVERRIDER*, *Stimulus Port Override Register* on page 2-24 and *STMSPMOVERRIDER*, *Stimulus Port Master Override Register* on page 2-26 enable tools to override what the software chooses. When overriding transactions to be guaranteed, this is considered invasive debug. This override mode does not operate when invasive debug is disabled.

Table 2-33 shows the behavior of the STM override functions based on the permitted level of debug.

**Table 2-33 Authentication control with guaranteed override selected**

Permitted debug level	Request type	Override selected	Request treated as
None	-	-	Invariant timing, write ignored
Non-invasive	Guaranteed	None	Guaranteed
Non-invasive	Invariant timing	None	Invariant timing
Non-invasive	Guaranteed	Guaranteed	Guaranteed
Non-invasive	Invariant timing	Guaranteed	Invariant timing
Non-invasive	-	Invariant timing	Invariant timing
Invasive	Guaranteed	None	Guaranteed
Invasive	Invariant timing	None	Invariant timing
Invasive	-	Guaranteed	Guaranteed
Invasive	-	Invariant timing	Invariant timing



## Chapter 3

# Extended Stimulus Ports

This chapter describes the extended stimulus ports. It contains the following sections:

- *About extended stimulus ports on page 3-52.*
- *STM transactions on page 3-54.*
- *Address decoding on page 3-55.*
- *Grouping stimulus ports on page 3-56.*
- *More than one master on page 3-57.*
- *Data sizes on page 3-58.*
- *Bus endianness on page 3-59.*
- *Implementation options on page 3-60.*
- *Reserved locations on page 3-61.*
- *Timestamping on page 3-62.*
- *Mapping onto STPv2 on page 3-63.*

## 3.1 About extended stimulus ports

Each extended stimulus port occupies 256 consecutive bytes in the memory map.

The STM extended stimulus ports must be marked as Device memory. This ensures writes to the STM occur in program order.

Multiple locations are available for each stimulus port. Each location allows software to choose the type of trace packet to be generated.

Data accesses can be optionally marked, for example to indicate the start or end of messages consisting of multiple transactions. Data accesses can also optionally request a timestamp to be generated with the trace packet.

Non-data accesses can generate the following types of trace packet:

**Flag** This is a simple marker with no data payload and can be used to indicate messages consisting of multiple packets.

**Trigger** This can be used to indicate a significant event in the trace.

Non-data accesses can be optionally timestamped.

All locations are write-only. Read accesses return zero, but software must not rely on this value.

Unaligned accesses are not supported. All accesses must be aligned to the access size.

Data accesses must be aligned to the bottom of the 8-byte window for each access type and, therefore, every data packet access must have address bits[2:0] == 0b000. Accesses with address bits[2:0] != 0b000 are UNPREDICTABLE. See [Data sizes on page 3-58](#) for more information on data accesses.

Non-data accesses must be written as zero and the implementation must ignore the data value.

[Table 3-1](#) shows the address map for a single stimulus port.

**Table 3-1 Address map for a single stimulus port**

Address offset	Short name	Description
<b>Guaranteed data accesses</b>		
0x00-0x04	G_DMTS	Data, marked with timestamp, guaranteed
0x08-0x0C	G_DM	Data, marked, guaranteed
0x10-0x14	G_DTS	Data, with timestamp, guaranteed
0x18-0x1C	G_D	Data, guaranteed
0x20-0x5C	-	Reserved
<b>Guaranteed non-data accesses</b>		
0x60-0x64	G_FLAGTS	Flag with timestamp, guaranteed
0x68-0x6C	G_FLAG	Flag, guaranteed
0x70-0x74	G_TRIGTS	Trigger with timestamp, guaranteed
0x78-0x7C	G_TRIG	Trigger, guaranteed
<b>Invariant Timing data accesses</b>		
0x80-0x84	I_DMTS	Data, marked with timestamp, invariant timing
0x88-0x8C	I_DM	Data, marked, invariant timing
0x90-0x94	I_DTS	Data, with timestamp, invariant timing

**Table 3-1 Address map for a single stimulus port (continued)**

Address offset	Short name	Description
0x98-0x9C	I_D	Data, invariant timing
0xA0-0xDC	-	Reserved
<b>Invariant Timing non-data accesses</b>		
0xE0-0xE4	I_FLAGTS	Flag with timestamp, invariant timing
0xE8-0xEC	I_FLAG	Flag, invariant timing
0xF0-0xF4	I_TRIGTS	Trigger with timestamp, invariant timing
0xF8-0xFC	I_TRIG	Trigger, invariant timing

## 3.2 STM transactions

The STM supports the following transactions:

- [Guaranteed transactions](#).
- [Invariant timing transactions](#).

### 3.2.1 Guaranteed transactions

Guaranteed transactions are guaranteed to be traced. This might involve stalling the bus or system to ensure the transaction is accepted by the STM, for example when the STM trace buffer is full.

When a guaranteed transaction is performed, the following aspects of the transaction are guaranteed to be traced if specified:

- Data.
- Mark.
- Timestamp.
- Flag.
- Trigger.

---

**Note**

Guaranteed transactions are also known as blocking transactions.

---

### 3.2.2 Invariant timing transactions

Invariant timing transactions are not guaranteed to be traced. These transactions will take an invariant amount of time regardless of the state of the STM.

When an invariant timing transaction is traced, the following aspects of the transaction are traced if specified:

- Data.
- Mark.
- Flag.
- Trigger.

If the transaction is dropped because the STM cannot accept it, none of these aspects is traced, except a trigger. For more information on triggers, see [Triggers on page 2-46](#).

When a write to an invariant timing location in a stimulus port requests a timestamp, this does not guarantee a timestamp is traced. The STM might choose to omit the timestamp, or assign the timestamp to a later packet if there is insufficient trace buffering or bandwidth.

Other system behavior might affect the timing of invariant timing transactions. In addition, mixing guaranteed and invariant timing transactions might cause the invariant timing transactions to take a variable amount of time to complete, because a guaranteed transaction might change the timing on the system bus which affects a subsequent invariant timing transaction.

If only invariant timing transactions are used, the STM responds identically to these transactions regardless of its state.

---

**Note**

Invariant timing transactions are also known as non-blocking transactions.

---

### 3.3 Address decoding

The address bits are used to define the type of packet.

Table 3-2 shows the address bit meanings for accesses where address bit[6] == 0b0.

**Table 3-2 Address bit meanings for data accesses**

Address bit	Function if clear	Function if set
[7]	The transaction is guaranteed	The transaction is invariant timing
[4]	This packet is marked	This packet is not marked
[3]	This packet is timestamped	This packet is not timestamped

Table 3-3 shows the address bit meanings for accesses where address bits[6:5] == 0b11.

**Table 3-3 Address bit meanings for non-data accesses**

Address bit	Function if clear	Function if set
[7]	The transaction is guaranteed	The transaction is invariant timing
[4]	This transaction causes a flag packet to be traced	This transaction causes a trigger event
[3]	This packet is timestamped	This packet is not timestamped

## 3.4 Grouping stimulus ports

Stimulus ports are grouped, where 16 stimulus ports occupy a 4KB page in memory, as [Table 3-4](#) shows.

**Table 3-4 Address map for a group of 16 stimulus ports**

Address offset	Description
0x000-0x0FF	Stimulus port 0
0x100-0x1FF	Stimulus port 1
.	.
.	.
.	.
0xE00-0xEFF	Stimulus port 14
0xF00-0xFFF	Stimulus port 15

An integer number of stimulus ports are supported. Where more than 16 stimulus ports are required, additional 4KB blocks are required for each additional full or partial group of 16 stimulus ports. These 4KB blocks are contiguous in the physical address space. The number of stimulus ports supported is IMPLEMENTATION DEFINED, up to 65536 in a memory map requiring 16MB of address space.



## 3.5 More than one master

Where more than 65536 stimulus ports are required, or where multiple independent system masters are required, the STM architecture supports extending the memory map to up to 65536 groups of stimulus ports, each group known as a master.

Each master supports the same number of stimulus ports, as defined by the [STMDEVID](#) register.

The number of masters is defined in the [STMFEAT3R](#) register.

Each master requires up to 16MB of address space. Each of these 16MB blocks are aligned to a 16MB boundary, even if the number of stimulus ports per master is fewer than 65536.

An implementation might support more than one master, but not all address spaces for every master are necessarily accessible by all masters in a system. For example, each processor in a system might be assigned a different master block, but might not be able to access the blocks for any another master.

## 3.6 Data sizes

An STM implementation supports a maximum fundamental data size, from one of the following:

- 32-bit.
- 64-bit.

---

**Note**

---

An STM does not generate a packet with a data size greater than its maximum fundamental data size.

---

Table 3-5 shows how many packets are generated for each transaction size, based on the fundamental data size of the implementation. The transaction size is dependent on the source of the transaction, for example, a processor, and the bus infrastructure used to transmit the transaction. For example, if a processor writes a 64-bit value over a 32-bit bus to an STM with a 32-bit fundamental data size, this might be presented as two STM packets because the bus might have separated the 64-bit value into two 32-bit transfers.

**Table 3-5 Expected packets based on fundamental data size**

Transaction size	Fundamental data size	
	32	64
8	1	1
16	1	1
32	1	1
64	2	1

If compression is enabled, the packet might be smaller than the transaction size. When analyzing the trace protocol and when compression is used to reduce the size of a trace packet, the trace packet must not be expanded to more than the maximum fundamental data size.

To ensure that code is portable between processor micro-architectures and system implementations, ARM recommends that only the native data size of the machine is used, and smaller sizes. For the 32-bit ARMv7 architecture, only 8, 16, and 32-bit transfers are recommended. For an ARMv8 processor that supports the AArch64 Execution state, it is recommended that the fundamental data size of 64-bits is implemented.

Generally, the data width of the interconnect driving the STM is at least as large as the fundamental data size of the STM. Where this is not the case, the interconnect must be able to indicate multiple parts of a single transaction so that they can be reconstituted atomically. For example, where the fundamental data size is 64 bits and the interconnect is 32 bits, the interconnect must be able to indicate that two halves of a 64-bit transaction must be combined to create a 64-bit transaction, and this must be performed atomically.

Although software stimulus must not perform data accesses where address bits[2:0] != 0b000, an implementation must support accesses aligned to its fundamental data size. For example, if the implementation has a fundamental data size of 32 bits, it must accept accesses where address bits[2:0] == 0b100. These accesses might occur in systems where a 64-bit transaction is downsized by the bus fabric to 2x32-bit transactions, and therefore the second access is to address 0x004 and the STM must accept this as a write to location 0x000.

## 3.7 Bus endianness

As a memory-mapped implementation, the endianness is determined by the system in which the STM is implemented. For example, a write of a 32-bit register containing the value 0x11223344 must be presented in the trace stream with 0x44 in the least significant byte.

If the STM is little-endian but the system is big-endian, hardware byte-swizzling must be implemented to ensure the value written into the STM has the least-significant byte at the bottom of the access.

For example, for an STM supporting up to 32-bit transactions, a big-endian byte write to 0x00 results in the byte of data being located in bits[31:24] of the value presented to the STM. A little-endian STM expects the data in bits[7:0], so the value must be swizzled.

---

**Note**

---

This refers to bus endianness and not processor endianness, for example, the endianness defined by the CPSR.E bit in the ARM Architecture.

---

## 3.8 Implementation options

[Table 3-6](#) shows the implementation options.

**Table 3-6 Implementation options**

Feature	Options
Data types	All implementations which implement STPv2 support the following basic data types: <ul style="list-style-type: none"><li>• D, DTS, DM, DMTS, FLAG, FLAG_TS, TRIG, and TRIG_TS</li></ul>
Fundamental data size	It is IMPLEMENTATION DEFINED what data sizes are supported. The fundamental data size is indicated in the <a href="#">STMFEAT1R</a> .
Invariant timing and guaranteed transactions	Invariant timing transactions and guaranteed transactions are optional, but at least one of the transaction types must be supported: <ul style="list-style-type: none"><li>• When not supported, the invariant timing locations in the extended stimulus port memory map behave as guaranteed transactions</li><li>• When not supported, the guaranteed locations in the extended stimulus port memory map behave as invariant timing transactions.</li></ul>

## 3.9 Reserved locations

The STM does not permit transactions to Reserved locations in the stimulus port memory map. The operation of the STM is UNPREDICTABLE on writes to these locations.

## 3.10 Timestamping

When a write to an invariant timing location in a stimulus port requests a timestamp, this does not always guarantee a timestamp is traced. The STM might omit the timestamp or assign the timestamp to a later packet if there is insufficient trace buffering or bandwidth.

The STM might also choose to timestamp a guaranteed or invariant timing transaction which was not requested to have a timestamp.

Timestamps are not generated when timestamping is disabled using the [STMTCSR.TSEN](#) control.

Timestamps are only guaranteed to be generated for a transaction which is requested to have a timestamp and:

- The transaction is marked as guaranteed.
- The [STMTCSR.TSEN](#) field is set.

Software must not rely on timestamps being generated for any messaging protocol.

## 3.11 Mapping onto STPv2

All stimulus ports are mapped onto an STPv2 channel with the same number as the stimulus port. The mapping onto STPv2 masters is IMPLEMENTATION DEFINED. An example is where all the masters are mapped into contiguous 16MB blocks and the upper address bits are used to define the master number.

If the STM drops a write to a invariant timing stimulus port, an error packet is generated which indicates that trace has been lost before tracing resumes. The packet might indicate that trace has been lost from a single specific master, or that the master which lost trace cannot be determined.

Synchronization of the trace stream generates the following packets:

- ASYNC.
- VERSION.
- FREQ, if [STMTCSR.TSEN](#) is set.





# Chapter 4

## Implementation Defined Controls

This chapter describes the IMPLEMENTATION DEFINED controls and registers. It contains the following sections:

- *About implementation defined controls and registers on page 4-66.*
- *Standard hardware event tracing on page 4-67.*
- *DMA control on page 4-77.*

## 4.1 About implementation defined controls and registers

Two blocks of 64 locations at 0xC00-0xCFC and 0xD00-0xDFC are reserved for IMPLEMENTATION DEFINED controls. This functionality might include:

- Hardware event tracing.
- DMA communication and configuration.

Each of these two blocks of 64 locations has an identification mechanism to enable identification of common functionality that might be present in multiple STMs. Location 0xFC in each block identifies any common function.

Figure 4-1 shows the Implementation Defined Controls Identification Register bit assignments.



Figure 4-1 Implementation Defined Controls Identification Register bit assignments

Table 4-1 shows the Implementation Defined Controls Identification Register bit assignments.

Table 4-1 Implementation Defined Controls Identification Register bit assignments

Bits	Name	Description
[31:12]	-	Reserved, UNK/SBZP.
[11:8]	VENDSPEC	The contents of this field are IMPLEMENTATION DEFINED.
[7:4]	CLASSREV	This field depends on the value of the Class field.
[3:0]	CLASS	The type of controls implemented. This defines the programmer's model of this block of controls: 0b0000 No controls implemented here. All other fields are UNK/SBZP. 0b0001 Hardware Event Control. 0b0010 DMA control. 0b1111 Unknown controls implemented.

You can interpret this register in the following order:

1. The CLASS field identifies the programmer's model.
2. The CLASSREV field identifies the revision of the programmer's model.
3. The VENDSPEC field identifies any vendor-specific modifications or mappings.

## 4.2 Standard hardware event tracing

A value of 0b0001 in the CLASS field of register 0xFC identifies standard hardware event tracing. This functionality provides a simple mechanism to trace simple signals in a system. Up to 256 signals are supported.

### 4.2.1 Hardware event control registers

The hardware event control registers operate simultaneously on a bank of 32 hardware events. If more than 32 hardware events are implemented, selection of the currently controlled bank is performed using the Hardware Event Bank Select Register.

Table 4-2 shows the standard hardware event tracing control registers, in register order. In the table, access type is described as follows:

**RW** Read and write.

**RO** Read only.

**Table 4-2 Standard hardware event tracing control register summary**

Register	Name	Type	Description
0x00	Event Enable	RW	See <i>STMHEER, Hardware Event Enable Register</i>
0x04-0x1C	-	-	Reserved
0x20	Trigger Enable	RW	See <i>STMHETER, Hardware Event Trigger Enable Register</i> on page 4-68
0x24-0x5C	-	-	Reserved
0x60	Bank Select	RW	See <i>STMHEBSR, Hardware Event Bank Select Register</i> on page 4-69
0x64	Main Control	RW	See <i>STMHEMCR, Hardware Event Main Control Register</i> on page 4-69
0x68	Hardware Event External Multiplex Control	RW	See <i>STMHEEXTMUXR, Hardware Event External Multiplex Control Register</i> on page 4-71
0x6C-0xF0	-	-	Reserved
0xF4	Master Number	RO or RW <sup>a</sup>	See <i>STMHEMASTR, Hardware Event Master Number Register</i> on page 4-72
0xF8	Features 1	RO	See <i>STMHEFEAT1R, Hardware Event Features 1 Register</i> on page 4-72
0xFC	ID	RO	See <i>STMHEIDR, Hardware Event ID Register</i> on page 4-73

a. Read the STMHEFEAT1R to determine if this register is RO or RW.

### STMHEER, Hardware Event Enable Register

The STMHEER characteristics are:

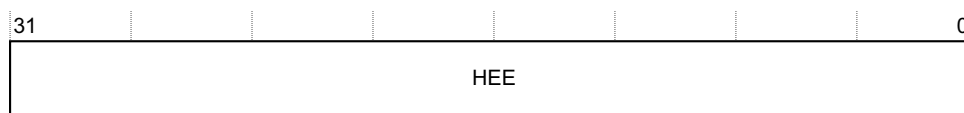
**Purpose** This register is used to enable hardware events to generate trace.

**Usage constraints** There are no usage constraints.

**Configurations** This is a banked register. Bank selection is done using the *STMHEBSR*.

**Attributes** See the register summary in Table 4-2.

Figure 4-2 on page 4-68 shows the STMHEER bit assignments.



**Figure 4-2 STMHEER bit assignments**

Table 4-3 shows the STMHEER bit assignments.

**Table 4-3 STMHEER bit assignments**

Bits	Name	Type	Description
[31:0]	HEE	RW	Hardware event enable, with one bit per hardware event: 0b0 Hardware event disabled. 0b1 Hardware event enabled. Reset value is UNKNOWN.

This register must always be initialized for each bank before enabling event tracing in the [STMHEMCR](#).

### STMHETER, Hardware Event Trigger Enable Register

The STMHETER characteristics are:

<b>Purpose</b>	Enables trigger generation on hardware events.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This is a banked register. Bank selection is done using the <a href="#">STMHEBSR</a> .
<b>Attributes</b>	See the register summary in <a href="#">Table 4-2 on page 4-67</a> .

Figure 4-3 shows the STMHETER bit assignments.



**Figure 4-3 STMHETER bit assignments**

Table 4-4 shows the STMHETER bit assignments.

**Table 4-4 STMHETER bit assignments**

Bits	Name	Type	Description
[31:0]	HETE	RW	Bit mask to enable trigger generation from the hardware events, with one bit per hardware event: 0b0 Disabled. 0b1 Enabled. Reset value is UNKNOWN.

This register must always be initialized for each bank before enabling event tracing in the [STMHEMCR](#).

## STMHEBSR, Hardware Event Bank Select Register

The STMHEBSR characteristics are:

<b>Purpose</b>	<p>Select a bank of 32 hardware events to control. For example:</p> <ul style="list-style-type: none"> <li>When this register is set to 0x0, reads from and writes to the <a href="#">STMHEER</a> and <a href="#">STMHETER</a> correspond to hardware event 0-31.</li> <li>When this register is set to 0x1, reads from and writes to the <a href="#">STMHEER</a> and <a href="#">STMHETER</a> correspond to hardware event 32-63.</li> </ul> <p>The size of this register is IMPLEMENTATION DEFINED but is based on the number of implemented hardware events as indicated in the <a href="#">STMHEFEAT1R</a>. If 32 or fewer hardware events are implemented, this register ignores writes, and reads as zero.</p>
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-2 on page 4-67</a> .

[Figure 4-4](#) shows the STMHEBSR bit assignments.



**Figure 4-4 STMHEBSR bit assignments**

[Table 4-5](#) shows the STMHEBSR bit assignments.

**Table 4-5 STMHEBSR bit assignments**

Bits	Name	Type	Description
[31:n]	-	-	Reserved, UNK/SBZP.
[n-1:0]	HEBS	RW	Selects the bank of 32 hardware events to control. Reset value of each bit is 0b0.

## STMHEMCR, Hardware Event Main Control Register

The STMHEMCR characteristics are:

<b>Purpose</b>	Controls the primary functions of the hardware event tracing.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-2 on page 4-67</a> .

[Figure 4-5 on page 4-70](#) shows the STMHEMCR bit assignments.

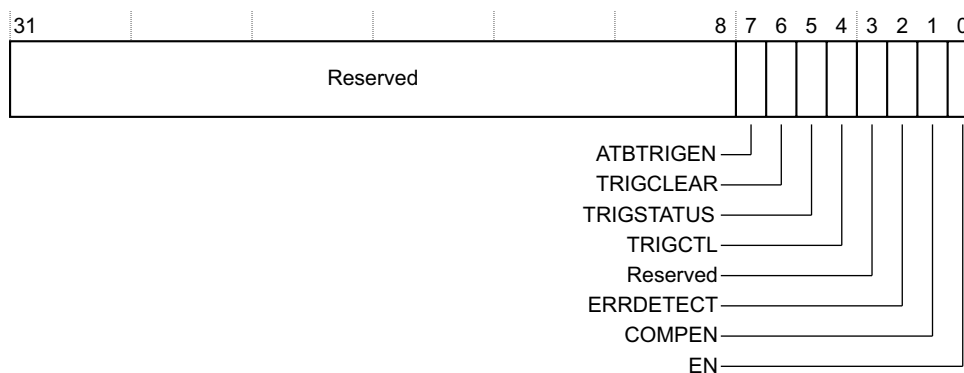


Figure 4-5 STMHEMCR bit assignments

Table 4-6 shows the STMHEMCR bit assignments.

Table 4-6 STMHEMCR bit assignments

Bits	Name	Type	Description
[31:8]	-	-	Reserved, UNK/SBZP.
[7]	ATBTRIGEN	RW	ATB trigger enable on events being monitored using the <a href="#">STMHETER</a> . When set, this bit enables the STM to use the ATID value of 0x7D. For more information, see <a href="#">Triggers on page 2-46</a> and <a href="#">STMHETER, Hardware Event Trigger Enable Register on page 4-68</a> . Reset value is UNKNOWN. This bit is implemented only when the STMFEAT1R.TRACEBUS is 0b0001.
[6]	TRIGCLEAR	WO	When TRIGCTL indicates single-shot mode, this bit is used to clear TRIGSTATUS: 0b0 No effect. 0b1 Clears TRIGSTATUS if TRIGSTATUS is 0b1. Writing a 0b1 to this bit when in multi-shot mode is UNPREDICTABLE.
[5]	TRIGSTATUS	RO	When TRIGCTL indicates single-shot mode, this indicates whether the single trigger has occurred: 0b0 Trigger has not occurred. 0b1 Trigger has occurred. In multi-shot mode this bit is always UNKNOWN.
[4]	TRIGCTL	RW	Trigger Control: 0b0 Triggers are multi-shot. 0b1 Triggers are single-shot. Reset value is UNKNOWN. For more information see <a href="#">Triggers on page 2-46</a> . This bit is implemented only when the <a href="#">STMHEFEAT1R</a> .TRIGCTL is 0b10.
[3]	-	-	Reserved

**Table 4-6 STMHEMCR bit assignments (continued)**

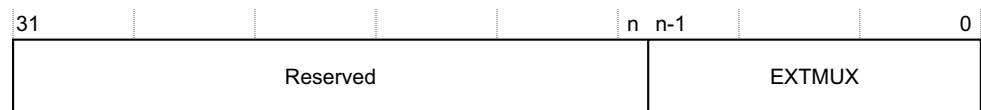
Bits	Name	Type	Description
[2]	ERRDETECT	RW	Enable error detection on the hardware event tracing: 0b0 Disabled. 0b1 Enabled. If an event cannot be traced, this bit enables indication of the lost information. Reset value is UNKNOWN.
[1]	COMPEN	RW	Enable leading zero suppression of hardware event data values in the trace stream: 0b0 Disabled. 0b1 Enabled. Reset value is UNKNOWN.
[0]	EN	RW	Enable Hardware Event Tracing: 0b0 Disabled. 0b1 Enabled. To enable hardware event tracing, the <a href="#">STMTCSSR.EN</a> bit must also be 0b1. Reset value is 0b0.

### STMHEEXTMUXR, Hardware Event External Multiplex Control Register

The STMHEEXTMUXR characteristics are:

<b>Purpose</b>	Control the multiplexing of many hardware events on the available hardware event inputs to the STM.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is implemented if <a href="#">STMHEFEAT1R.HEEXTMUXSIZE</a> is not zero.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-2 on page 4-67</a> .

[Figure 4-6](#) shows the STMHEEXTMUXR bit assignments.



**Figure 4-6 STMHEEXTMUXR bit assignments**

[Table 4-7](#) shows the STMHEEXTMUXR bit assignments.

**Table 4-7 STMHEEXTMUXR bit assignments**

Bits	Name	Type	Description
[31:n]	-	-	Reserved.
[n-1:0]	EXTMUX	RW <sup>a</sup>	Provides a value to optional multiplexing logic, to control which hardware events are connected to the STM. The behavior of this multiplexing logic is IMPLEMENTATION DEFINED. This field is reset to zero.

a. The size of this field is defined by [STMHEFEAT1R.HEEXTMUXSIZE](#).

STMHEMASTR, Hardware Event Master Number Register

The STMHEMASTR characteristics are:

- Purpose** Indicate the master number of hardware event trace. This number is the master number presented in the trace protocol.
- Usage constraints** There are no usage constraints.
- Configurations** This register is available in all implementations.
- Attributes** See the register summary in [Table 4-2 on page 4-67](#).

Figure 4-7 shows the STMHEMASTR bit assignments.

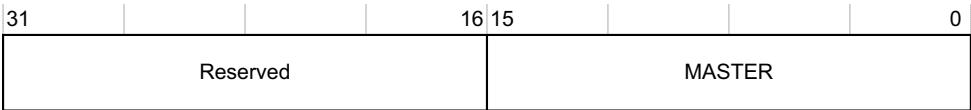


Figure 4-7 STMHEMASTR bit assignments

Table 4-8 shows the STMHEMASTR bit assignments.

Table 4-8 STMHEMASTR bit assignments

Bits	Name	Type	Description
[31:16]	-	-	Reserved, UNK/SBZP
[15:0]	MASTER	RO or RW <sup>a</sup>	The master number for hardware event trace. Reset value is IMPLEMENTATION DEFINED.

a. Read the STMHEFEAT1R to determine if this register is RO or RW.

STMHEFEAT1R, Hardware Event Features 1 Register

The STMHEFEAT1R characteristics are:

- Purpose** Indicates the hardware event tracing features of the STM.
- Usage constraints** There are no usage constraints.
- Configurations** This register is available in all implementations.
- Attributes** See the register summary in [Table 4-2 on page 4-67](#).

Figure 4-8 shows the STMHEFEAT1R bit assignments.

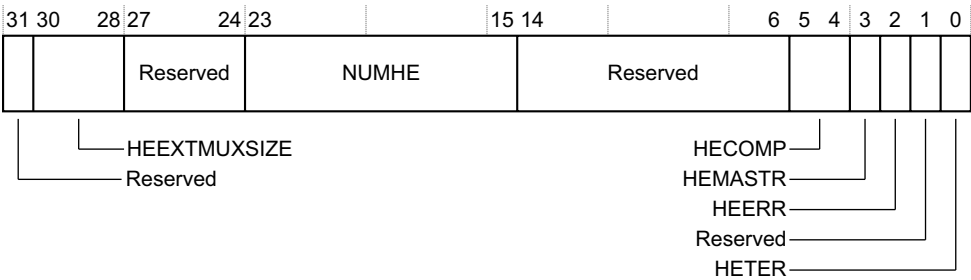


Figure 4-8 STMHEFEAT1R bit assignments



Table 4-9 shows the STMHEFEAT1R bit assignments.

**Table 4-9 STMHEFEAT1R bit assignments**

Bits	Name	Description
[31]	-	Reserved, RAZ.
[30:28]	HEEXTMUXSIZE	Indicates size of <a href="#">STMHEEXTMUXR</a> : 0b000 <a href="#">STMHEEXTMUXR</a> is not implemented. 0b001 <a href="#">STMHEEXTMUXR</a> is implemented and <a href="#">STMHEEXTMUXR.EXTMUX</a> is 2-bits wide. 0b010 <a href="#">STMHEEXTMUXR</a> is implemented and <a href="#">STMHEEXTMUXR.EXTMUX</a> is 4-bits wide. 0b011 <a href="#">STMHEEXTMUXR</a> is implemented and <a href="#">STMHEEXTMUXR.EXTMUX</a> is 8-bits wide. 0b100 <a href="#">STMHEEXTMUXR</a> is implemented and <a href="#">STMHEEXTMUXR.EXTMUX</a> is 16-bits wide. 0b101 <a href="#">STMHEEXTMUXR</a> is implemented and <a href="#">STMHEEXTMUXR.EXTMUX</a> is 32-bits wide. 0b110 Reserved. 0b111 Reserved. This field is always 0b000 if <a href="#">STMHEIDR.CLASSREV</a> is 0b0000.
[27:24]	-	Reserved, RAZ.
[23:15]	NUMHE	Number of hardware events supported. 0 to 256 events are supported.
[14:6]	-	Reserved, RAZ.
[5:4]	HECOMP	Data compression on hardware event tracing support: 0b00 Data compression support is not defined here. Use the part number of the device to determine if data compression is supported. 0b01 No data compression supported. 0b10 Data compression always enabled. 0b11 Data compression support is programmable. <a href="#">STMHEMCR.COMPEN</a> is implemented.
[3]	HEMASTR	<a href="#">STMHEMASTR</a> support: 0b0 <a href="#">STMHEMASTR</a> is RO. 0b1 <a href="#">STMHEMASTR</a> is RW.
[2]	HEERR	Hardware event error detection support: 0b0 Hardware event error detection not implemented. 0b1 Hardware event error detection implemented. <a href="#">STMHEMCR.ERRDETECT</a> is implemented.
[1]	-	Reserved, RAZ.
[0]	HETER	<a href="#">STMHETER</a> support: 0b0 <a href="#">STMHETER</a> is not implemented. 0b1 <a href="#">STMHETER</a> is implemented.

If 32 or fewer hardware events are supported, [STMHEBSR](#) is not implemented.

### STMHEIDR, Hardware Event ID Register

This register uses the 0b0001 encoding of the CLASS field.

There are two possible values of the CLASSREV field:

0b0000 Hardware event controls version 1.

0b0001 Hardware event controls version 2.

Version 2 adds the following features:

- [STMHEFEAT1R.HEEXTMUXSIZE](#).
- [STMHEEXTMUXR](#).

For more information about this register, see [About implementation defined controls and registers](#) on page 4-66.

## 4.2.2 Changing the STM programming

Hardware event tracing is only enabled when both the [STMTCSSR.EN](#) and [STMHEMCR.EN](#) bits are both 0b1.

The STM does not have to be disabled to be reprogrammed. The following registers can be modified while the [STMTCSSR.EN](#) and [STMHEMCR.EN](#) bits are 0b1:

- [STMHEER](#).
- [STMHETER](#).
- [STMHEBSR](#).
- [STMHEEXTMUXR](#).

## 4.2.3 Tracing hardware events

Data packets are output using channels 0-7 for tracing the hardware events. Events are encoded as a function of the channel number and the payload of the data packet. [STMHEMASTR](#) specifies the master number. The data is output in two formats:

- DM/DMTS where the payload indicates the number of the event.
- D/DTS where the payload is a bit field with 1 bit per event. A bit is set for each event which occurred.

When a timestamp is included, the events indicated in the data packet occurred at the time indicated. When a timestamp is not included, the STM was unable to add an accurate timestamp. There will be a subsequent packet with a timestamp to indicate the approximate time of the events.

[Table 4-10](#) shows hardware event tracing using STPv2.

**Table 4-10 Hardware event tracing**

Packet	Payload	Meaning
M8/M16	8-bit/16-bit master identifier	The STPv2 master number for hardware event tracing.
C8	8-bit channel identifier	Used in combination with data packets to indicate which events have occurred.
DxMTS	Up to 8 bits of data, timestamp	The data payload indicates the event number from 0 to 255: $\text{Event number} = (\text{floor}(\text{channel} / 8) * 256) + \text{payload}$ The timestamp represents the time the event indicated occurred.
DxM	Up to 8 bits of data	The data payload indicates the event number from 0 to 255: $\text{Event number} = (\text{floor}(\text{channel} / 8) * 256) + \text{payload}$ An accurate timestamp was not available for this event.

**Table 4-10 Hardware event tracing (continued)**

Packet	Payload	Meaning
DxTS	Up to 64 bits of data, timestamp	The data payload is encoded with 1 bit per hardware event: Event number = (channel * 32) + bit position The timestamp represents the time the events indicated occurred. When multiple events are indicated, they all occurred at the same time.
Dx	Up to 64 bits of data	The data payload is encoded with 1 bit per hardware event: Event number = (channel * 32) + bit position An accurate timestamp was not available for these events. When multiple events are indicated, they did not necessarily occur at the same time
FLAG_TS	Timestamp	The timestamp indicates the time the FLAG_TS packet was generated. All events traced before this packet occurred on or before this timestamp. This is typically output soon after a D/DM packet to indicate the approximate time of those non-timestamped events. The channel number is irrelevant for FLAG_TS packets.

If the same event occurs multiple times before a data packet is output indicating the event, an error packet is traced indicating an event has been lost when [STMHEMCR.ERRDETECT](#) is 0b1.

The payload might be leading-zero suppressed. This is enabled using the [STMHEMCR.COMPEN](#) field. When enabled, if the higher-order bits of the data value to be traced are zero, a smaller packet might be output. For example, if only event 5 is to be traced, a D4MTS packet might be output with a payload of 0x5. Similarly, if events 0 and 3 occurred simultaneously, a D4TS packet might be output with a payload of 0x9.

### Hardware event tracing examples

All of these examples assume that [STMHEMCR.COMPEN](#) is b1, enabling leading-zero suppression of the payload values. Also, these examples assume that the current master is the value in [STMHEMASTR.MASTER](#) and therefore do not include the trace required for changing to that master number.

#### Example 1

Only event 4 is asserted.

- A D4MTS packet is generated with a payload of 0x4, indicating event 4 was asserted. The timestamp value is the time the event was asserted.

#### Example 2

Event 4 and event 0 are asserted.

- A D8TS packet is generated with a payload of 0x11, with the bitfield indicating events 0 and 4 were asserted. The timestamp value is the time the events were asserted.

#### Example 3

Event 31 is asserted, however the STM cannot output this immediately.

On the next cycle, this can be output.

- A D8M packet is output with a payload of 0x1F, indicating event 31 was asserted. There is no timestamp in this packet because the packet output was delayed from the time the event occurred.
- Later, a FLAG\_TS packet is output with the current timestamp value. This can be used to determine that event 31 was asserted approximately near this timestamp.

#### **Example 4**

Event 31 is asserted, however the STM cannot output this immediately.

On the next cycle, event 0 is asserted and a packet can be output.

- A D32 packet is output with a payload of 0x80000001, with the bitfield indicating events 31 and 0 were asserted. There is no timestamp in this packet because the packet output was delayed from the time that event 31 occurred.
- Later, a FLAG\_TS packet is output with the current timestamp value. This can be used to determine that events 31 and 0 were asserted approximately near this timestamp.

#### **Example 5**

Event 17 is asserted, however the STM cannot output this immediately.

On the next cycle, this can be output.

- A D8M packet is output with a payload of 0x11, indicating event 17 was asserted. There is no timestamp in this packet because the packet output was delayed from the time the event occurred.

On the following cycle, event 5 is asserted, and again the STM cannot output this immediately. On the next cycle this can be output.

- A D4M packet is output with a payload of 0x5, indicating event 5 was asserted. Again, there is no timestamp in this packet because the packet output was delayed from the time the event occurred.
- Later, a FLAG\_TS packet is output with the current timestamp value. This can be used to determine that events 17 and 5 were asserted approximately near this timestamp.

### 4.3 DMA control

This section describes registers for basic control of *Direct Memory Access* (DMA) transfers to and from the STM. These controls are implemented when the CLASS field of the STMDMAIDR is 0b0010.

### 4.3.1 DMA control registers

Table 4-11 shows the example DMA control registers, in register order. In the table, access type is described as follows:

**RW** Read and write.

**RO** Read only.

**WO** Write only.

### Table 4-11 Example DMA control registers

Register	Name	Type	Description
0x00	-	-	Reserved
0x04	Transfer Start	WO	See <i>STMDMASTARTR, DMA Transfer Start Register</i>
0x08	Transfer Stop	WO	See <i>STMDMASTOPR, DMA Transfer Stop Register</i> on page 4-78
0x0C	Transfer Status	RO	See <i>STMDMASTATR, DMA Transfer Status Register</i> on page 4-78
0x10	Control	RW	See <i>STMDMACTLR, DMA Control Register</i> on page 4-79
0x14-0xF8	-	-	Reserved
0xFC	ID	RO	See <i>STMDMAIDR, DMA ID Register</i> on page 4-80

### STMDMASTARTR, DMA Transfer Start Register

The STMDMASTARTR characteristics are:

<b>Purpose</b>	Starts a DMA transfer:
----------------	------------------------

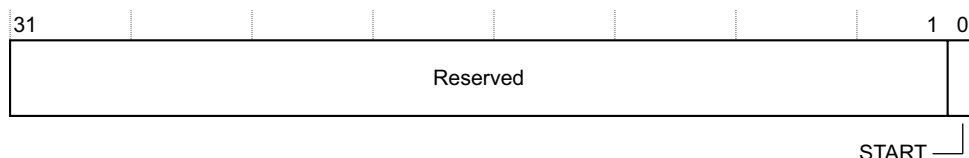
- A write of `0b1` when the DMA peripheral request interface is idle starts a DMA transfer.
- A write of `0b0` has no effect.
- A write of `0b1` when the DMA peripheral request interface is active has no effect.

<b>Usage constraints</b>	There are no usage constraints.
--------------------------	---------------------------------

**Configurations** This register is available in all implementations.

**Attributes** See the register summary in [Table 4-11](#).

Figure 4-9 shows the STMDMASTARTR bit assignments.



### Figure 4-9 STMDMASTARTR bit assignments

Table 4-12 shows the STMDMASTARTR bit assignments.

**Table 4-12 STMDMASTARTR bit assignments**

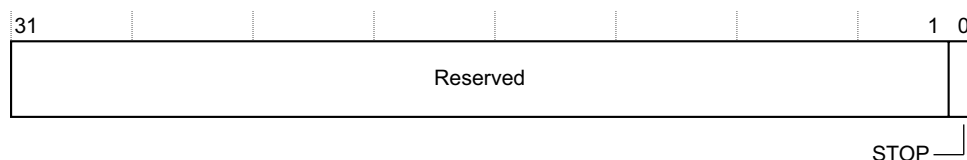
Bits	Name	Description
[31:1]	-	Reserved, UNK/SBZP
[0]	START	Start a DMA transfer

## STMDMASTOPR, DMA Transfer Stop Register

The STMDMASTOPR characteristics are:

<b>Purpose</b>	Stops a DMA transfer: <ul style="list-style-type: none"> <li>A write of 0b1 stops an active DMA transfer.</li> <li>A write of 0b0 has no effect.</li> <li>A write of 0b1 when the DMA peripheral request interface is idle has no effect.</li> </ul>
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-11 on page 4-77</a> .

Figure 4-10 shows the STMDMASTOPR bit assignments.



**Figure 4-10 STMDMASTOPR bit assignments**

Table 4-13 shows the STMDMASTOPR bit assignments.

**Table 4-13 STMDMASTOPR bit assignments**

Bits	Name	Description
[31:1]	-	Reserved, UNK/SBZP.
[0]	STOP	Stop a DMA transfer

## STMDMASTATR, DMA Transfer Status Register

The STMDMASTATR characteristics are:

<b>Purpose</b>	Indicates whether a DMA transfer is in progress.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-11 on page 4-77</a> .

Figure 4-11 on page 4-79 shows the STMDMASTATR bit assignments.

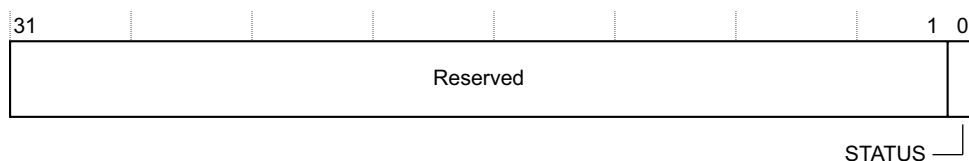
**Figure 4-11 STMDMASTATR bit assignments**

Table 4-14 shows the STMDMASTATR bit assignments.

**Table 4-14 STMDMASTATR bit assignments**

Bits	Name	Description
[31:1]	-	Reserved, UNK/SBZP.
[0]	STATUS	Status of the DMA peripheral request interface: 0b0 Interface is idle. 0b1 Interface is active.

### STMDMACTLR, DMA Control Register

The STMDMACTLR characteristics are:

<b>Purpose</b>	Controls the DMA transfer request mechanism.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	This register is available in all implementations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-11 on page 4-77</a> .

Figure 4-12 shows the STMDMACTLR bit assignments.

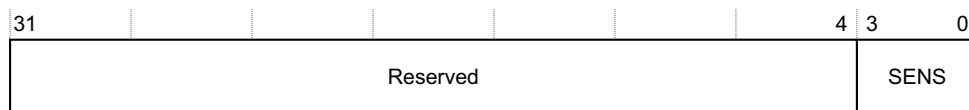
**Figure 4-12 STMDMACTLR bit assignments**

Table 4-15 shows the STMDMACTLR bit assignments.

**Table 4-15 STMDMACTLR bit assignments**

Bits	Name	Description
[31:4]	-	Reserved, UNK/SBZP.
[3:0]	SENS	Determines the sensitivity of the DMA request to the current buffer level in the STM. A smaller value indicates that the STM waits for a large amount of buffer space to be available before requesting a DMA transfer. Not all bits of this field might be implemented. Lower order bits might not be implemented. To detect the implemented bits, write 0b1111 to this field and read it back. The bits that return 0b1 are implemented. If no bits are implemented, there is no control over the sensitivity. Reset value is 0b0000.

The [STMDMACTLR.SENS](#) field is a hint to the hardware and does not necessarily correspond to any specific buffer levels. This field is intended to be used to balance the usage of the STM to ensure there is sufficient buffer space and appropriate throughput.

### STMDMAIDR, DMA ID Register

This register uses the 0b0010 encoding of the CLASS field. For more information about this register, see [About implementation defined controls and registers on page 4-66](#).



# Appendix A

## Recommended Configurations

This appendix describes the recommended configurations for using the STM architecture. It contains the following section:

- [\*About recommended configurations on page A-82.\*](#)

## A.1 About recommended configurations

The STM architecture has many IMPLEMENTATION DEFINED options. [Table A-1](#) shows the recommended configurations.

**Table A-1 Recommended configurations**

Feature	Recommended configuration 1	Recommended configuration 2
Trace protocol	STPv2	STPv2
Timestamping	Absolute	Absolute
STMTSFREQR	Read-write	Read-write
STMTSSTIMR	Implemented	Implemented
STMSYNCR	Implemented	Implemented
Claim tags	4	4
TRACEID	CoreSight ATB plus ATB trigger	CoreSight ATB plus ATB trigger
Trigger control	Multi-shot and single-shot	Multi-shot and single-shot
<a href="#">STMTCSR.TSPRESCALE</a>	Not implemented	Not implemented
<a href="#">STMTCSR.HWTEN</a>	Not implemented	Not implemented
<a href="#">STMTCSR.SYNCEN</a>	Always reads as 0b1	Always reads as 0b1
<a href="#">STMTCSR.SWOEN</a>	Not implemented	Not implemented
Number of stimulus ports	65536	65536
Number of masters	Minimum of 2	Minimum of 2
Stimulus port types	Extended only	Extended only
Fundamental data size	32	64
Transaction types	Invariant timing and guaranteed	Invariant timing and guaranteed
<a href="#">STMSPER</a>	Implemented	Implemented
<a href="#">STMSPTER</a>	Implemented	Implemented
<a href="#">STMPRIVMASKR</a>	Not implemented	Not implemented
<a href="#">STMSPOVERRIDER</a> and <a href="#">STMSPMOVERRIDER</a>	Implemented	Implemented
<a href="#">STMSPSCR</a> and <a href="#">STMSPMSCR</a>	Implemented	Implemented
Data compression on stimulus ports	Programmable	Programmable
Hardware event tracing	See <a href="#">Table A-2 on page A-83</a>	See <a href="#">Table A-2 on page A-83</a>

For systems with an ARMv7 processor, ARM recommends configuration 1 or configuration 2.

For systems with an ARMv8-A processor, ARM recommends configuration 2.

Table A-2 shows the *Hardware Event* (HE) tracing recommended configuration.

**Table A-2 Hardware event tracing recommended configuration**

Feature	Recommended configuration
Number of HW events	0-256
<a href="#">STMHETER</a>	Implemented
HW error detection	Implemented
<a href="#">STMHEMASTR</a>	Read only
Data compression on HW trace	Programmable



# Appendix B

## Revisions

This appendix describes the main technical changes between released issues of this book.

**Table B-1 Differences between issue A and issue B**

Change	Location
Added support for the STMv1.1 architecture:	
Added <a href="#">STMDEVARCH</a> register	<a href="#">STMDEVARCH, Device Architecture Register on page 2-41</a>
Added support for version 2 of the Standard hardware event tracing controls:	
• Added the <a href="#">STMHEEXTMUXR</a> register.	<a href="#">STMHEEXTMUXR, Hardware Event External Multiplex Control Register on page 4-71</a>
• Added the <a href="#">STMHEFEAT1R.HEEXTMUXSIZE</a> field.	<a href="#">STMHEFEAT1R, Hardware Event Features 1 Register on page 4-72</a>
• Updated the possible values of <a href="#">STMHEIDR.CLASSREV</a> .	<a href="#">STMHEIDR, Hardware Event ID Register on page 4-73</a>
Renamed some fields in the CoreSight Management registers to be consistent with the CoreSight Architecture specification.	Entire document
Corrected the address decoding for Extended Stimulus Ports for non-data accesses.	<a href="#">Address decoding on page 3-55</a>
Added some examples for hardware event tracing.	<a href="#">Tracing hardware events on page 4-74</a>
Added a new recommended configuration for ARMv8-A processors.	<a href="#">Table A-1 on page A-82</a>



# Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

<b>CoreSight</b>	The infrastructure for monitoring, tracing, and debugging a complete system on chip.
<b>Debugger</b>	<p>A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.</p> <p>An application that monitors and controls the operation of a second application. Usually used to find errors in the application program flow.</p>
<b>IMPLEMENTATION DEFINED</b>	The behavior is not architecturally defined, but must be defined and documented by individual implementations.
<b>IMPLEMENTATION SPECIFIC</b>	The exact behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.
<b>Macrocell</b>	A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.
<b>RAZ</b>	<i>See</i> Read-As-Zero fields.
<b>Read-As-Zero fields (RAZ)</b>	Appear as zero when read.
<b>Reserved</b>	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces UNPREDICTABLE results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are IMPLEMENTATION SPECIFIC. All reserved bits not used by the implementation must be written as zero and are Read-As-Zero.
<b>SBZP</b>	<i>See</i> Should-Be-Zero-or-Preserved

**Should-Be-Zero-or-Preserved (SZBP)**

Must be written as 0, or all 0s for a bit field, by software if the value is being written without having been previously read, or if the register has not been initialized. Where the register was previously read on the same processor, since the processor was last reset, the value in the field should be preserved by writing the value that was previously read.

Hardware must ignore writes to these fields.

If a value is written to the field that is neither 0 (or all 0s for a bit field), nor a value previously read for the same field on the same processor, the result is UNPREDICTABLE.

**TPA**

*See* Trace Port Analyzer.

**Trace port**

A port on a device, such as a processor or ASIC, that is used to output trace information.

**Trace Port Analyzer (TPA)**

A hardware device that captures trace information output on a trace port. This can be a low-cost product designed specifically for trace acquisition, or a logic analyzer.

**UNDEFINED**

Indicates an instruction that generates an Undefined Instruction exception.

**UNKNOWN**

An UNKNOWN value does not contain valid data, and can vary from moment to moment, instruction to instruction, and implementation to implementation. An UNKNOWN value must not be a security hole. UNKNOWN values must not be documented or promoted as having a defined value or effect.

**UNPREDICTABLE**

Means that the behavior of the STM cannot be relied on. Such conditions have not been validated. UNPREDICTABLE behavior can affect the behavior of the entire system.