

`cortex_a76_trm`
Arm[®] Cortex[®]-A76 Core
Technical Reference Manual

Release Information

Release Information

Issue	Date	Confidentiality	Change
[this text will be replaced automatically with content from: <bookevent><revisionid> in <bookchangehistory>]	[this text will be replaced automatically with content from: <bookevent> dates in <bookchangehistory>]	[this text will be replaced automatically with content from: <data name="permission" value="nonconfidential.restricted"/> in the <bookmap>...<bookchangehistory>]	[this text will be replaced automatically with content from: @reys <revisionid> in the <bookmap>...<bookchangehistory>

Proprietary Notice

This document is CONFIDENTIAL and any use by you is subject to the terms of the agreement between you and Arm or the terms of the agreement between you and the party authorised by Arm to disclose this document to you.

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: (i) for the purposes of determining whether implementations infringe any third party patents;(ii) for developing technology or products which avoid any of Arm's intellectual property; or (iii) as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or (iv) for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arm technology described in this document with any other products created by you or a third party, without obtaining Arm's prior written consent.

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 click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US 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/company/policies/trademarks>.

Copyright © Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

機密著作権情報

本書は機密情報であり、お客様による使用は、お客様と Arm との間で締結した契約の条項、またはお客様と本書をお客様に開示することを Arm が認めた当事者との間で締結した契約の条項に従うものとします。

本書は著作権などの権利により保護されており、本書に含まれる手順または実装に関する情報は、1 つ以上の特許または申請中の特許により保護されている可能性があります。本書のいかなる部分も、Arm から事前に書面による明示的な承諾なく、何らかの形式や方法で無断複製することは許可されていません。特に記載がない限り、明示的であるか黙示的であるかを問わず、また禁反言やその他いかなる知的財産権のライセンスを許諾するものではありません。

本書の情報には、以下の目的のために情報を使用せず、第三者にもそれを許可しないと承諾することを条件としてアクセスすることができます。(i) 実装により、いかなる第三者の特許も侵害されないことを確認すること、(ii) Arm のいかなる知的所有権も回避したテクノロジーまたは製品を開発すること、(iii) 既存特許または特許申請に変更を加えるための参考としたり、部分的に書き加えたり、既存特許または特許申請を拡張したりすること、または (iv) 事前に書面による Arm の同意を得ることなく、本書で説明されている Arm テクノロジーの性能または機能をお客様または第三者により作成された何らかの他製品と比較する第三者に対し、公開または開示するためのデータを作成すること。

本書は、「現状」のまま提供されます。Arm は、明示的、黙示的、または制定法上のいずれを問わず、いかなる表明も保証も行いません。これには、本書に関連した商品性、品質基準、非侵害、または特定目的への適合性に関する黙示的保証を含むが、これに限定されません。疑義を避けるため、Arm は第三者の特許、著作権、営業機密、または他の権利の範囲および内容に関して、いかなる表明も行わず、識別や理解のための分析も行いません。

本書には、技術的に不正確な箇所および誤記が含まれる場合があります。

法により禁止されていない限りにおいて、Arm は本書の使用により生じた直接的、間接的、特別、付随的、懲罰的、または結果的損害などを含むすべての損害に対して、たとえそのような損害の可能性が事前に告知されていた場合でも、その原因および責任理論の如何に関わらず一切の責任を負わないものとします。

本書には、商品のみが含まれています。本書の使用、複製、または開示が関連するあらゆる輸出法および輸出規制に完全に準拠し、本書が全体であれ一部であれ、該当する輸出法に違反して直接的または間接的に輸出されることがないことを保証する責任を負うものとします。Arm のお客様に関連して「パートナー」という言葉が使用されている場合でも、他会社と提携関係を設立することや、言及することを意図するものではありません。Arm は、通知することなくいつでも本書を変更することができます。

本契約のいずれかの規定と、Arm と締結された本書の内容を含むクリックスルー契約または署名済みの書面契約の間に矛盾がある場合、クリックスルー契約または署名済みの書面契約を本契約の規定より優先す

るものとします。本書は、便宜上、他言語に翻訳される場合がありますが、本書の英語版と翻訳との間に矛盾がある場合、契約書の英語版に含まれる規定を優先することに同意するものとします。

Arm のロゴおよび ® または ™ のマークが付いた語は、Arm Limited (またはその子会社) の米国およびその他の国における登録商標または商標です。 All rights reserved. 本書に記載されている他の製品名は、各社の所有する商標です。 Arm の商標の使用に関する次のガイドラインに従ってください。 <http://www.arm.com/company/policies/trademarks>

Copyright © , Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

기밀 소유권 고지 사항

이 문서는 기밀 문서이며 귀하가 이를 사용하는 경우 귀하와 Arm 간의 계약 약관 또는 귀하와 Arm이 귀하에게 이 문서를 공개하도록 승인한 당사자 간의 계약 약관이 적용됩니다.

이 문서는 저작권 및 기타 관련 권한에 의해 보호되며 이 문서에 포함된 정보의 실행 및 구현은 하나 이상의 특허 또는 출원 중인 특허 신청에 의해 보호될 수 있습니다. 이 문서의 어떠한 부분도 Arm의 명시적인 사전 서면 승인 없이 어떠한 수단에도 의해서든 어떠한 형태로도 복제할 수 없습니다. 구체적인 진술이 있는 경우가 아닌 한 이 문서로 금반언 또는 다른 방식에 의해 지적 재산권에 대한 명시적 또는 암시적 라이선스가 부여되지 않습니다.

귀하는 이 문서의 정보를 Arm의 사전 서면 동의 없이 (i) 구현 시 타사 특허를 침해하는지 확인할 목적으로, (ii) Arm의 지적 재산을 무효로 하는 기술 또는 제품을 개발할 목적으로 또는 (iii) 기존 특허 또는 특허 신청을 수정하거나 기존 특허 또는 특허 신청의 연속, 부분적인 연속 또는 연장에 대한 참고 자료로 또는 (iv) 이 문서에 설명된 Arm 기술의 성능 또는 기능을 귀하 또는 타사가 제작한 다른 제품과 비교하는 데이터를 생성하여 게시 또는 타사에 공개할 목적으로 사용하거나 다른 사람이 사용하도록 허용하지 않는 것에 동의해야만 정보에 액세스할 수 있습니다.

이 문서는 "있는 그대로" 제공됩니다. Arm은 암시적 상품성 보증, 만족스러운 품질, 문서와 관련해 특정 목적 적합성이 포함되나 그에 국한되지 않는 명시적, 암시적 또는 법적 진술, 보증을 하지 않습니다. 의문을 방지할 수 있도록, Arm은 타사 특허, 저작권, 사업 비밀 또는 기타 권한의 범위 및 내용을 식별 또는 파악하기 위한 분석과 관련하여 어떠한 진술도 하지 않으며 그러한 분석에 착수하지도 않았습니다.

이 문서에는 기술적 부정확성 또는 오타가 포함되어 있을 수 있습니다.

법률에 의해 금지되지 않는 범위 내에서, Arm은 Arm이 해당 손해 발생 가능성에 대해 고지를 받았더라도 원인 및 책임 이론에 관계없이, 이 문서 사용에 의해 발생하는 직접적, 간접적, 특별적, 우발적, 징벌적 또는 결과적 손해가 포함되나 그에 국한되지 않는 손해에 대해 책임을 지지 않습니다.

이 문서는 상업적 항목으로만 구성됩니다. 귀하는 이 문서를 사용, 중복 또는 공개 시 이 문서 또는 문서의 어떤 부분을 관련 수출법을 위반하여 수출되지 않도록 보장하기 위해 관련 수출법 및 규정을 완전하게 준수할 책임이 있습니다. Arm의 고객을 가리키는 "파트너"라는 단어를 사용하더라도 다른 회사와의 파트너 관계를 형성하거나 이를 가리키려는 의도가 아닙니다. Arm은 언제든지 사전 통지 없이 이 문서를 변경할 수 있습니다.

이 약관에 포함된 조항 중 Arm과 이 문서가 포함되는 클릭 또는 서명을 통해 체결된 서면 계약의 조항과 충돌하는 것이 있는 경우 클릭 또는 서명을 통해 체결된 서면 계약이 이 약관의 충돌 조항에 우선하며 이를 대체합니다. 이 문서는 편의를 위해 다른 언어로 번역될 수 있으며 귀하는 이 문서의 영문본과 번역이 충돌하는 경우 영문본 계약의 약관이 우선 적용됨에 동의합니다.

Arm의 기업 로고 및 ® 또는 ™으로 표시된 단어는 미국 및/또는 다른 지역에서 Arm Limited(또는 자회사)의 등록 상표 또는 상표입니다. All rights reserved. 그 외 이 문서에 언급된 브랜드 및 이름은 해당 소유자의 상표일 수 있습니다. <http://www.arm.com/company/policies/trademarks>에 있는 Arm의 상표 사용 지침을 따르십시오.

Copyright © , Arm Limited 또는 그 계열사. All rights reserved.

Arm Limited. 잉글랜드 등록 법인 제02557590호.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

保密所有权

本文档为保密资料，您在使用时须遵守您与 Arm 之间所签协议的各项条款，以及您与获得 Arm 授权向您披露本文档的有关方之间所签协议的各项条款。

本文档受版权和其他相关权利的保护，实践或实施本文档中所含信息可能受到一项或多项专利或待定专利申请的保护。未经 Arm 事先明确书面许可，不得以任何形式通过任何手段复制本文档的任何部分。除非明确说明，否则本文档未以禁止反言或其他方式授予任何知识产权方面的许可，无论是明示还是暗示许可。

您对本文档所含信息的访问取决于您是否接受不出于以下目的使用或允许他人使用此信息的条件：(i) 为了确定实施是否会侵犯任何第三方专利；(ii) 为了开发技术或产品以避免 Arm 的任何知识产权；或 (iii) 作为修改现有专利或专利申请或制造任何延续、部分延续或扩展现有专利或专利申请的参考；或 (iv) 为了生成数据以发布或披露给第三方，在未获得 Arm 事先书面同意的情况下，将本文档中所述的 Arm 技术的性能或功能性与您或第三方创建的任何其他产品进行比较。

本文档“按原样”提供。ARM 就本文档不作任何陈述和保证，无论是明示、暗示或法定保证，包括但不限于对适销性、满意的质量、不侵权或针对特定目的的适用性的暗示保证。为避免疑义，Arm 不就在分析的基础上识别或理解第三方专利、版权、商业机密或其他权利的范围和内容作出任何陈述和承诺。

本文档可能包含技术或排版上的错误。

在不受法律禁止的范围内，ARM 在任何情况下都不会对因使用本文档引起的任何损害承担责任，包括但不限于任何直接、间接、特殊、连带、惩罚性或后果性损害，无论损害如何造成以及何种责任理论，即便 ARM 已被告知有此类损害的可能性。

本文档仅包含商业内容。您应负责确保本文档的任何使用、复制或披露完全符合任何相关的出口法律法规，以确保本文档或其任何部分不会在违反相关出口法律法规的情况下直接或间接出口。在提及 Arm 的客户时使用“合作伙伴”一词并非意欲建立或指代与任何其他公司的任何合作伙伴关系。Arm 可以随时对本文档进行更改，恕不另行通知。

如果这些条款中的任何规定与同 Arm 达成的涵盖本文档的任何点击或签署的书面协议中的任何规定有冲突，则以点击或签署的书面协议为准，并取代这些条款中的冲突规定。本文档可以翻译成其他语言以方便使用，您并且同意，若本文档的英文版与任何翻译版出现任何冲突时，将以协议的英文版条款为准。

Arm 公司的徽标以及带有® 或™ 标记的文字为 Arm Limited (或其子公司) 在美国和/或其他地方的注册商标或商标。保留所有权利。本文档中提及的其他品牌和名称可能是其各自所有者的商标。请遵循 <http://www.arm.com/company/policies/trademarks> 上的 Arm 商标使用指南。

版权 ©，归 ARM Limited 或其关联公司所有。保留所有权利。

Arm Limited。英格兰注册公司，注册号：02557590。

110 Fulbourn Road, Cambridge, England CB1 9NJ。

LES-PRE-20348

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 click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US 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/company/policies/trademarks>.

Copyright © Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

非機密著作権情報

本書は著作権などの権利により保護されており、本書に含まれる手順または実装に関する情報は、1つ以上の特許または申請中の特許により保護されている可能性があります。本書のいかなる部分も、Armから事前に書面による明示的な承諾なく、何らかの形式や方法で無断複製することは許可されていません。特に記載がない限り、明示的であるか黙示的であるかを問わず、また禁反言やその他いかなる知的財産権のライセンスを許諾するものではありません。

本書の情報には、実装により、いかなる第三者の特許も侵害されないことを確認する目的のために情報を使用せず、第三者にもそれを許可しないと承諾することを条件としてアクセスすることができます。

本書は、「現状」のまま提供されます。Armは、明示的、黙示的、または制定法上のいずれを問わず、いかなる表明も保証も行いません。これには、本書に関連した商品性、品質基準、非侵害、または特定目的への適合性に関する黙示的保証を含むが、これに限定されません。疑義を避けるため、Armは第三者の特許、著作権、営業機密、または他の権利の範囲および内容に関して、いかなる表明も行わず、識別や理解のための分析も行いません。

本書には、技術的に不正確な箇所および誤記が含まれる場合があります。

法により禁止されていない限りにおいて、Armは本書の使用により生じた直接的、間接的、特別、付随的、懲罰的、または結果的損害などを含むすべての損害に対して、たとえそのような損害の可能性が事前に告知されていた場合でも、その原因および責任理論の如何に関わらず一切の責任を負わないものとします。

本書には、商品のみが含まれています。本書の使用、複製、または開示が関連するあらゆる輸出法および輸出規制に完全に準拠し、本書が全体であれ一部であれ、該当する輸出法に違反して直接的または間接的に輸出されることがないことを保証する責任を負うものとします。ARMのお客様に関連して「パート

ナー」という言葉が使用されている場合でも、他会社と提携関係を設立することや、言及することを意図するものではありません。Arm は、通知することなくいつでも本書を変更することができます。

本契約のいずれかの規定と、Arm と締結された本書の内容を含むクリックスルー契約または署名済みの書面契約の間に矛盾がある場合、クリックスルー契約または署名済みの書面契約を本契約の規定より優先するものとします。本書は、便宜上、他言語に翻訳される場合がありますが、本書の英語版と翻訳との間に矛盾がある場合、契約書の英語版に含まれる規定を優先することに同意するものとします。

Arm のロゴおよび ® または ™ のマークが付いた語は、Arm Limited (またはその子会社) の米国およびその他の国における登録商標または商標です。All rights reserved. 本書に記載されている他の製品名は、各社の所有する商標です。Arm の商標の使用に関する次のガイドラインに従ってください。<http://www.arm.com/company/policies/trademarks>

Copyright © , Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

공개 소유권 고지 사항

이 문서는 저작권 및 기타 관련 권한에 의해 보호되며 이 문서에 포함된 정보의 실행 및 구현은 하나 이상의 특허 또는 출원 중인 특허 신청에 의해 보호될 수 있습니다. 이 문서의 어떠한 부분도 Arm의 명시적인 사전 서면 승인 없이는 어떠한 수단에 의해서든 어떠한 형태로도 복제할 수 없습니다. 구체적인 진술이 있는 경우가 아닌 한 이 문서로 금반언 또는 다른 방식에 의해 지적 재산권에 대한 명시적 또는 암시적 라이선스가 부여되지 않습니다.

귀하는 이 문서의 정보를 Arm의 사전 서면 동의 없이 구현 시 타사 특허를 침해하는지 확인할 목적으로 사용하거나 다른 사람이 사용하도록 허용하지 않는 것에 동의해야만 정보에 액세스할 수 있습니다.

이 문서는 "있는 그대로" 제공됩니다. Arm은 암시적 상품성 보증, 만족스러운 품질, 문서와 관련해 특정 목적 적합성이 포함되나 그에 국한되지 않는 명시적, 암시적 또는 법적 진술, 보증을 하지 않습니다. 의문을 방지할 수 있도록, Arm은 타사 특허, 저작권, 사업 비밀 또는 기타 권한의 범위 및 내용을 식별 또는 파악하기 위한 분석과 관련하여 어떠한 진술도 하지 않으며 그러한 분석에 착수하지도 않았습니다.

이 문서에는 기술적 부정확성 또는 오타가 포함되어 있을 수 있습니다.

법률에 의해 금지되지 않는 범위 내에서, Arm은 Arm이 해당 손해 발생 가능성에 대해 고지를 받았더라도 원인 및 책임 이론에 관계없이, 이 문서 사용에 의해 발생하는 직접적, 간접적, 특별적, 우발적, 징벌적 또는 결과적 손해가 포함되나 그에 국한되지 않는 손해에 대해 책임을 지지 않습니다.

이 문서는 상업적 항목으로만 구성됩니다. 귀하는 이 문서를 사용, 중복 또는 공개 시 이 문서 또는 문서의 어떤 부분을 관련 수출법을 위반하여 수출되지 않도록 보장하기 위해 관련 수출법 및 규정을 완전하게 준수할 책임이 있습니다. Arm의 고객을 가리키는 "파트너"라는 단어를 사용하더라도 다른 회사와의 파트너 관계를 형성하거나 이를 가리키려는 의도가 아닙니다. Arm은 언제든지 사전 통지 없이 이 문서를 변경할 수 있습니다.

이 약관에 포함된 조항 중 Arm과 이 문서가 포함되는 클릭 또는 서명을 통해 체결된 서면 계약의 조항과 충돌하는 것이 있는 경우 클릭 또는 서명을 통해 체결된 서면 계약이 이 약관의 충돌 조항에 우선하며 이를 대체합니다. 이 문서는 편의를 위해 다른 언어로 번역될 수 있으며 귀하는 이 문서의 영문본과 번역이 충돌하는 경우 영문본 계약의 약관이 우선 적용됨에 동의합니다.

Arm의 기업 로고 및 ® 또는 ™로 표시된 단어는 미국 및/또는 다른 지역에서 Arm Limited(또는 자회사)의 등록 상표 또는 상표입니다. All rights reserved. 그 외 이 문서에 언급된 브랜드 및 이름은 해당 소유자의 상표일 수 있습니다. <http://www.arm.com/company/policies/trademarks>에 있는 Arm의 상표 사용 지침을 따르십시오.

Copyright © , Arm Limited 또는 그 계열사. All rights reserved.

Arm Limited. 잉글랜드 등록 법인 제02557590호.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

非机密专有权声明

本文档受版权和其他相关权利的保护，实践或实施本文档中所含信息可能受到一项或多项专利或待定专利申请的保护。未经 Arm 事先明确书面许可，不得以任何形式通过任何手段复制本文档的任何部分。除非明确说明，否则本文档未以禁止反言或其他方式授予任何知识产权方面的许可，无论是明示还是暗示许可。

您对本文档所含信息的访问取决于您是否接受不会出于确定实施是否侵犯任何第三方专利的目的而使用或允许其他人使用此信息的条件。

本文档“按原样”提供。Arm 就本文档不作任何陈述和保证，无论是明示、暗示或法定保证，包括但不限于对适销性、满意的质量、不侵权或针对特定目的的适用性的暗示保证。为避免疑义，Arm 不就在分析的基础上识别或理解第三方专利、版权、商业机密或其他权利的范围和内容作出任何陈述和承诺。

本文档可能包含技术或排版上的错误。

在不受法律禁止的范围内，Arm 在任何情况下都不会对因使用本文档引起的任何损害承担责任，包括但不限于任何直接、间接、特殊、连带、惩罚性或后果性损害，无论损害如何造成以及何种责任理论，即便 Arm 已被告知有此类损害的可能性。

本文档仅包含商业内容。您应负责确保本文档的任何使用、复制或披露完全符合任何相关的出口法律法规，以确保本文档或其任何部分不会在违反相关出口法律法规的情况下直接或间接出口。在提及 Arm 的客户时使用“合作伙伴”一词并非意欲建立或指代与任何其他公司的任何合作伙伴关系。Arm 可以随时对本文档进行更改，恕不另行通知。

如果这些条款中的任何规定与同 Arm 达成的涵盖本文档的任何点击或签署的书面协议中的任何规定有冲突，则以点击或签署的书面协议为准，并取代这些条款中的冲突规定。本文档可以翻译成其他语言以方便使用，您并且同意，若本文档的英文版与任何翻译版出现任何冲突时，将以协议的英文版条款为准。

Arm 公司的徽标以及带有®或™标记的文字为 Arm Limited（或其子公司）在美国和/或其他地方的注册商标或商标。保留所有权利。本文档中提及的其他品牌和名称可能是其各自所有者的商标。请遵循 <http://www.arm.com/company/policies/trademarks> 上的 Arm 商标使用指南。

版权 ©，归 Arm Limited 或其关联公司所有。保留所有权利。

Arm Limited。英格兰注册公司，注册号：02557590。

110 Fulbourn Road, Cambridge, England CB1 9NJ。

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

This document is Non-Confidential restricted. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Restricted Access is an Arm internal classification.

Product Status

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

The information in this document is for an Alpha product, that is a product under development.

The information in this document is for a Beta product, that is a product under development.

Web Address

<http://www.arm.com>

Contents

Preface: Preface.....	xxi
About this book.....	xxi
Product revision status.....	xxi
Intended audience.....	xxi
Using this book.....	xxi
Additional reading.....	xxiii
Feedback.....	xxiv
Feedback on this product.....	xxiv
Feedback on content.....	xxiv
 Part I: Functional description.....	 25
 Chapter 1: Introduction.....	 27
About the core.....	28
Features.....	28
Implementation options.....	29
Supported standards and specifications.....	30
Test features.....	31
Design tasks.....	31
Product revisions.....	32
 Chapter 2: Technical overview.....	 33
Components.....	34
Instruction fetch.....	36
Instruction decode.....	36
Register rename.....	36
Instruction issue.....	36
Execution pipeline.....	36
L1 data memory system.....	37
L2 memory system.....	37
Interfaces.....	37
About system control.....	37
About the Generic Timer.....	37
 Chapter 3: Clocks, resets, and input synchronization.....	 39
About clocks, resets, and input synchronization.....	40
Asynchronous interface.....	40
 Chapter 4: Power management.....	 41
About power management.....	42
Voltage domains.....	42
Power domains.....	43
Architectural clock gating modes.....	44
Core Wait for Interrupt.....	45

Core Wait for Event.....	45
Power control.....	46
Core power modes.....	46
On.....	46
Off.....	47
Off (emulated).....	47
Core dynamic retention.....	47
Debug recovery mode.....	47
Encoding for power modes.....	48
Power domain states for power modes.....	48
Power up and down sequences.....	49
Debug over powerdown.....	50

Chapter 5: Memory Management Unit.....51

About the MMU.....	52
Main functions.....	52
AArch64 behavior.....	52
TLB organization.....	53
Instruction L1 TLB.....	53
Data L1 TLB.....	53
L2 TLB.....	53
TLB match process.....	54
Translation table walks.....	55
AArch64 behavior.....	55
MMU memory accesses.....	55
Configuring MMU accesses.....	56
Descriptor hardware update.....	56
Specific behaviors on aborts and memory attributes.....	56
External aborts.....	57
Mis-programming contiguous hints.....	57
Memory attributes.....	57
Page Based Hardware Attributes.....	57
Page Based Hardware Attributes.....	58

Chapter 6: Level 1 memory system.....59

About the L1 memory system.....	60
L1 instruction-side memory system.....	60
L1 data-side memory system.....	60
L1 instruction memory system.....	60
L1 data memory system.....	60
Memory system implementation.....	61
Data prefetching.....	62
Direct access to internal memory.....	62
Encoding for L1 instruction cache tag, L1 instruction cache data, L1 BTB, L1 GHCB, L1 TLB instruction, and BPIQ.....	63
Encoding for L1 data cache tag, L1 data cache data, and L1 TLB data.....	67
Encoding for the L2 unified cache.....	72
Encoding for the L2 TLB.....	75

Chapter 7: Level 2 memory system.....79

About the L2 memory system.....	80
About the L2 cache.....	80
Support for memory types.....	80

Chapter 8: Reliability, Availability, and Serviceability (RAS).....	81
Cache ECC and parity.....	82
Uncorrected errors and data poisoning.....	82
RAS error types.....	83
Error Synchronization Barrier.....	83
Error recording.....	84
Error injection.....	84
 Chapter 9: Generic Interrupt Controller CPU interface.....	 87
About the Generic Interrupt Controller CPU interface.....	88
Bypassing the CPU interface.....	88
 Chapter 10: Advanced SIMD and floating-point support.....	 89
About the Advanced SIMD and floating-point support.....	90
Accessing the feature identification registers.....	90
 Part II: Register descriptions.....	 91
 Chapter 11: AArch32 system registers.....	 93
AArch32 architectural system register summary.....	94
 Chapter 12: AArch64 system registers.....	 97
AArch64 registers.....	102
AArch64 architectural system register summary.....	102
AArch64 implementation defined register summary.....	120
AArch64 registers by functional group.....	124
ACTLR_EL1, Auxiliary Control Register, EL1.....	137
ACTLR_EL2, Auxiliary Control Register, EL2.....	137
ACTLR_EL3, Auxiliary Control Register, EL3.....	140
AFSR0_EL1, Auxiliary Fault Status Register 0, EL1.....	143
AFSR0_EL2, Auxiliary Fault Status Register 0, EL2.....	143
AFSR0_EL3, Auxiliary Fault Status Register 0, EL3.....	144
AFSR1_EL1, Auxiliary Fault Status Register 1, EL1.....	144
AFSR1_EL2, Auxiliary Fault Status Register 1, EL2.....	145
AFSR1_EL3, Auxiliary Fault Status Register 1, EL3.....	146
AIDR_EL1, Auxiliary ID Register, EL1.....	146
AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1.....	147
AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2.....	147
AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3.....	148
.....	148
.....	149
.....	149
.....	149
.....	149
CLIDR_EL1, Cache Level ID Register, EL1.....	149
CPACR_EL1, Architectural Feature Access Control Register, EL1.....	151
CPTR_EL2, Architectural Feature Trap Register, EL2.....	151
CPTR_EL3, Architectural Feature Trap Register, EL3.....	152
CPUACTLR_EL1, CPU Auxiliary Control Register, EL1.....	153

CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1.....	154
CPUACTLR3_EL1, CPU Auxiliary Control Register 3, EL1.....	155
CPUCFR_EL1, CPU Configuration Register, EL1.....	156
CPUECTLR_EL1, CPU Extended Control Register, EL1.....	158
CPUPCR_EL3, CPU Private Control Register, EL3.....	168
CPUPMR_EL3, CPU Private Mask Register, EL3.....	170
CPUPOR_EL3, CPU Private Operation Register, EL3.....	171
CPUPSELR_EL3, CPU Private Selection Register, EL3.....	172
CPUPWRCTLR_EL1, Power Control Register, EL1.....	173
CSSELR_EL1, Cache Size Selection Register, EL1.....	176
DISR_EL1, Deferred Interrupt Status Register, EL1.....	177
ERRIDR_EL1, Error ID Register, EL1.....	178
ERRSELR_EL1, Error Record Select Register, EL1.....	179
ERXADDR_EL1, Selected Error Record Address Register, EL1.....	179
ERXCTLR_EL1, Selected Error Record Control Register, EL1.....	180
ERXFR_EL1, Selected Error Record Feature Register, EL1.....	180
ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1.....	180
ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1.....	180
ERXPFPGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1.....	180
ERXPFPGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1.....	182
ERXPFPGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1.....	183
ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1.....	184
ESR_EL1, Exception Syndrome Register, EL1.....	184
ESR_EL2, Exception Syndrome Register, EL2.....	185
ESR_EL3, Exception Syndrome Register, EL3.....	186
HACR_EL2, Hyp Auxiliary Configuration Register, EL2.....	187
HCR_EL2, Hypervisor Configuration Register, EL2.....	187
ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0.....	189
ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1.....	189
ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1.....	189
ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1, EL1.....	190
ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1.....	190
ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1.....	193
ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1.....	194
ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1.....	195
ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1.....	197
ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1.....	198
ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1, EL1.....	200
ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1.....	200
ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1.....	201
ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1.....	202
ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1.....	204
ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1.....	206
ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1.....	208
ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1.....	210
ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1.....	212
ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1.....	215
ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1.....	215
ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1.....	217
ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1.....	218
ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1.....	220
ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1.....	222
ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1.....	223
ID_PFR2_EL1, AArch32 Processor Feature Register 2, EL1.....	224
LORC_EL1, LORegion Control Register, EL1.....	225
LORID_EL1, LORegion ID Register, EL1.....	226

LORN_EL1, LORegion Number Register, EL1.....	227
MDCR_EL3, Monitor Debug Configuration Register, EL3.....	227
MIDR_EL1, Main ID Register, EL1.....	229
MPIDR_EL1, Multiprocessor Affinity Register, EL1.....	230
PAR_EL1, Physical Address Register, EL1.....	232
REVIDR_EL1, Revision ID Register, EL1.....	232
RMR_EL3, Reset Management Register.....	233
RVBAR_EL3, Reset Vector Base Address Register, EL3.....	234
SCTLR_EL1, System Control Register, EL1.....	234
SCTLR_EL3, System Control Register, EL3.....	235
TCR_EL1, Translation Control Register, EL1.....	237
TCR_EL2, Translation Control Register, EL2.....	238
TCR_EL3, Translation Control Register, EL3.....	239
TTBR1_EL1, Translation Table Base Register 1, EL1.....	240
TTBR1_EL2, Translation Table Base Register 1, EL2.....	241
VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2.....	241
VDISR_EL2 at EL1 using AArch64.....	241
VSESR_EL2, Virtual SError Exception Syndrome Register.....	242
VTCTR_EL2, Virtualization Translation Control Register, EL2.....	242
VTTBR_EL2, Virtualization Translation Table Base Register, EL2.....	243

Chapter 13: Error system registers..... 245

Error system register summary.....	246
ERR0ADDR, Error Record Address Register.....	247
ERR0CTLR, Error Record Control Register.....	247
ERR0FR, Error Record Feature Register.....	249
ERR0MISC1, Error Record Miscellaneous Register 1.....	251
ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register.....	251
ERR0PFGCTL, Error Pseudo Fault Generation Control Register.....	252
ERR0PFGF, Error Pseudo Fault Generation Feature Register.....	253
ERR0STATUS, Error Record Primary Status Register.....	255

Chapter 14: GIC registers..... 259

CPU interface registers.....	261
AArch64 physical GIC CPU interface system register summary.....	261
ICC_AP0R0_EL1, Interrupt Controller Active Priorities Group 0 Register 0, EL1.....	263
ICC_AP1R0_EL1, Interrupt Controller Active Priorities Group 1 Register 0, EL1.....	263
ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1.....	264
ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1.....	264
ICC_CTLR_EL1, Interrupt Controller Control Register, EL1.....	265
ICC_CTLR_EL3, Interrupt Controller Control Register, EL3.....	267
ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1.....	269
ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2.....	270
ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3.....	271
AArch64 virtual GIC CPU interface register summary.....	273
ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Register 0, EL1.....	274
ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Register 0, EL1.....	274
ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1.....	274
ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1.....	275
ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1.....	276
AArch64 virtual interface control system register summary.....	278
ICH_AP0R0_EL2, Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2.....	279
ICH_AP1R0_EL2, Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2.....	279
ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2.....	280

ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2.....	283
ICH_VTR_EL2, Interrupt Controller VGIC Type Register, EL2.....	285

Chapter 15: Advanced SIMD and floating-point registers..... 289

AArch64 register summary.....	290
FPCR, Floating-point Control Register.....	290
FPSR, Floating-point Status Register.....	292
MVFR0_EL1, Media and VFP Feature Register 0, EL1.....	294
MVFR1_EL1, Media and VFP Feature Register 1, EL1.....	295
MVFR2_EL1, Media and VFP Feature Register 2, EL1.....	297
AArch32 register summary.....	298
FPSCR, Floating-Point Status and Control Register.....	299

Part III: Debug descriptions..... 303

Chapter 16: Debug.....305

About debug methods.....	306
Debug register interfaces.....	306
Core interfaces.....	307
Breakpoints and watchpoints.....	307
Effects of resets on debug registers.....	307
External access permissions to debug registers.....	308
Debug events.....	308
Watchpoint debug events.....	309
Debug OS Lock.....	309
External debug interface.....	309

Chapter 17: Performance Monitor Unit..... 311

About the PMU.....	312
PMU functional description.....	312
External register access permissions.....	312
PMU events.....	313
PMU interrupts.....	328
Exporting PMU events.....	328

Chapter 18: Activity Monitor Unit.....329

About the AMU.....	330
Accessing the activity monitors.....	330
Access enable bit.....	330
System register access.....	330
External memory-mapped access.....	330
AMU counters.....	330
AMU events.....	331

Chapter 19: Embedded Trace Macrocell..... 333

About the ETM.....	334
ETM trace unit generation options and resources.....	334
ETM trace unit functional description.....	335
Resetting the ETM.....	336
Programming and reading ETM trace unit registers.....	336

ETM trace unit register interfaces.....	337
Part IV: Debug registers.....	339
Chapter 20: AArch32 debug registers.....	341
AArch32 debug register summary.....	342
Chapter 21: AArch64 debug registers.....	343
AArch64 debug register summary.....	344
DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1.....	346
DBGCLAIMSET_EL1, Debug Claim Tag Set Register, EL1.....	349
Chapter 22: Memory-mapped debug registers.....	351
Memory-mapped debug register summary.....	352
EDCIDR0, External Debug Component Identification Register 0.....	357
EDCIDR1, External Debug Component Identification Register 1.....	357
EDCIDR2, External Debug Component Identification Register 2.....	358
EDCIDR3, External Debug Component Identification Register 3.....	358
EDDEVID, External Debug Device ID Register 0.....	359
EDDEVID1, External Debug Device ID Register 1.....	359
EDPIDR0, External Debug Peripheral Identification Register 0.....	360
EDPIDR1, External Debug Peripheral Identification Register 1.....	360
EDPIDR2, External Debug Peripheral Identification Register 2.....	361
EDPIDR3, External Debug Peripheral Identification Register 3.....	361
EDPIDR4, External Debug Peripheral Identification Register 4.....	362
EDPIDRn, External Debug Peripheral Identification Registers 5-7.....	362
EDRCR, External Debug Reserve Control Register.....	362
Chapter 23: AArch32 PMU registers.....	365
AArch32 PMU register summary.....	366
PMCEID0, Performance Monitors Common Event Identification Register 0.....	368
PMCEID1, Performance Monitors Common Event Identification Register 1.....	372
PMCR, Performance Monitors Control Register.....	373
Chapter 24: AArch64 PMU registers.....	377
AArch64 PMU register summary.....	378
PMCEID0_EL0, Performance Monitors Common Event Identification Register 0, EL0.....	380
PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0.....	384
PMCR_EL0, Performance Monitors Control Register, EL0.....	385
Chapter 25: Memory-mapped PMU registers.....	389
Memory-mapped PMU register summary.....	390
PMCFGR, Performance Monitors Configuration Register.....	393
PMCIDR0, Performance Monitors Component Identification Register 0.....	394
PMCIDR1, Performance Monitors Component Identification Register 1.....	394
PMCIDR2, Performance Monitors Component Identification Register 2.....	395
PMCIDR3, Performance Monitors Component Identification Register 3.....	395
PMPIDR0, Performance Monitors Peripheral Identification Register 0.....	396
PMPIDR1, Performance Monitors Peripheral Identification Register 1.....	396

PMPIDR2, Performance Monitors Peripheral Identification Register 2.....	397
PMPIDR3, Performance Monitors Peripheral Identification Register 3.....	398
PMPIDR4, Performance Monitors Peripheral Identification Register 4.....	398
PMPIDRn, Performance Monitors Peripheral Identification Register 5-7.....	399

Chapter 26: PMU snapshot registers..... 401

PMU snapshot register summary.....	402
PMPCSSR, Snapshot Program Counter Sample Register.....	402
PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register.....	403
PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register.....	403
PMSSSR, PMU Snapshot Status Register.....	404
PMOVSSR, PMU Overflow Status Snapshot Register.....	404
PMCCNTSR, PMU Cycle Counter Snapshot Register.....	405
PMEVCNTRn, PMU Cycle Counter Snapshot Registers 0-5.....	405
PMSSCR, PMU Snapshot Capture Register.....	405

Chapter 27: AArch64 AMU registers.....407

AArch64 AMU register summary.....	408
AMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0.....	408
AMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0.....	409
AMCFGR_EL0, Activity Monitors Configuration Register, EL0.....	410
AMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0.....	412
AMEVCNTRn_EL0, Activity Monitor Event Counter Register, EL0.....	413
AMEVTYPERn_EL0, Activity Monitor Event Type Register, EL0.....	414

Chapter 28: ETM registers.....417

TRCACATRn, Address Comparator Access Type Registers 0-7.....	421
TRCACVRn, Address Comparator Value Registers 0-7.....	423
TRCAUTHSTATUS, Authentication Status Register.....	423
TRCAUXCTLR, Auxiliary Control Register.....	424
TRCBBCTLR, Branch Broadcast Control Register.....	426
TRCCCCTLR, Cycle Count Control Register.....	427
TRCCIDCCTLR0, Context ID Comparator Control Register 0.....	428
TRCCIDCVR0, Context ID Comparator Value Register 0.....	429
TRCCIDR0, ETM Component Identification Register 0.....	429
TRCCIDR1, ETM Component Identification Register 1.....	430
TRCCIDR2, ETM Component Identification Register 2.....	430
TRCCIDR3, ETM Component Identification Register 3.....	431
TRCCLAIMCLR, Claim Tag Clear Register.....	431
TRCCLAIMSET, Claim Tag Set Register.....	432
TRCCNTCTLR0, Counter Control Register 0.....	432
TRCCNTCTLR1, Counter Control Register 1.....	434
TRCCNTRL DVRn, Counter Reload Value Registers 0-1.....	435
TRCCNTVRn, Counter Value Registers 0-1.....	436
TRCCONFIGR, Trace Configuration Register.....	437
TRCDEVAFF0, Device Affinity Register 0.....	440
TRCDEVAFF1, Device Affinity Register 1.....	440
TRCDEVARCH, Device Architecture Register.....	440
TRCDEVID, Device ID Register.....	441
TRCDEVTYPE, Device Type Register.....	441
TRCEVENTCTL0R, Event Control 0 Register.....	442
TRCEVENTCTL1R, Event Control 1 Register.....	443
TRCEXTINSELR, External Input Select Register.....	444

TRCIDR0, ID Register 0.....	445
TRCIDR1, ID Register 1.....	447
TRCIDR2, ID Register 2.....	447
TRCIDR3, ID Register 3.....	449
TRCIDR4, ID Register 4.....	450
TRCIDR5, ID Register 5.....	452
TRCIDR8, ID Register 8.....	453
TRCIDR9, ID Register 9.....	453
TRCIDR10, ID Register 10.....	454
TRCIDR11, ID Register 11.....	454
TRCIDR12, ID Register 12.....	455
TRCIDR13, ID Register 13.....	455
TRCIMSPEC0, <i>Implementation Specific</i> Register 0.....	456
TRCITATBIDR, Integration ATB Identification Register.....	456
TRCITCTRL, Integration Mode Control Register.....	457
TRCITIATBINR, Integration Instruction ATB In Register.....	457
TRCITIATBOUTr, Integration Instruction ATB Out Register.....	458
TRCITIDATAR, Integration Instruction ATB Data Register.....	459
TRCLAR, Software Lock Access Register.....	459
TRCLSR, Software Lock Status Register.....	460
TRCCNTVRn, Counter Value Registers 0-1.....	460
TRCOSLAR, OS Lock Access Register.....	461
TRCOSLSR, OS Lock Status Register.....	461
TRCPDCR, Power Down Control Register.....	462
TRCPDSR, Power Down Status Register.....	463
TRCPIDR0, ETM Peripheral Identification Register 0.....	464
TRCPIDR1, ETM Peripheral Identification Register 1.....	465
TRCPIDR2, ETM Peripheral Identification Register 2.....	465
TRCPIDR3, ETM Peripheral Identification Register 3.....	466
TRCPIDR4, ETM Peripheral Identification Register 4.....	466
TRCPIDRn, ETM Peripheral Identification Registers 5-7.....	467
TRCPRGCTLR, Programming Control Register.....	467
TRCRSCTLRn, Resource Selection Control Registers 2-16.....	468
TRCSEQEVRn, Sequencer State Transition Control Registers 0-2.....	469
TRCSEQRSTEVr, Sequencer Reset Control Register.....	470
TRCSEQSTR, Sequencer State Register.....	471
TRCSSCCR0, Single-Shot Comparator Control Register 0.....	471
TRCSSCSR0, Single-Shot Comparator Status Register 0.....	472
TRCSTALLCTLR, Stall Control Register.....	473
TRCSTATR, Status Register.....	474
TRCSYNCPR, Synchronization Period Register.....	475
TRCTRACEIDR, Trace ID Register.....	475
TRCTSCTLR, Global Timestamp Control Register.....	476
TRCVICTLR, ViewInst Main Control Register.....	477
TRCVIIECTLR, ViewInst Include-Exclude Control Register.....	479
TRCVISSCTLR, ViewInst Start-Stop Control Register.....	479
TRCVMIDCVR0, VMID Comparator Value Register 0.....	480
TRCVMIDCCTLR0, Virtual context identifier Comparator Control Register 0.....	481

Appendices..... 483

Appendix A: Cortex[®]-A76 Core AArch32 UNPREDICTABLE behaviors.....	485
Use of R15 by Instruction.....	486

Load/Store accesses crossing page boundaries.....	486
Armv8 Debug UNPREDICTABLE behaviors.....	486
Other UNPREDICTABLE behaviors.....	490
 Appendix B: Revisions.....	 491
Revisions.....	492
Glossary.....	495

Preface

Preface

•
•

About this book

Information about the revision status and the intended audience for the book.

Product revision status

The *rm**pn* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm

Identifies the major revision of the product, for example, r1.

pn

Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

This manual is for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses an Arm core.

Using this book

Information about the chapters in this book.

This book is organized into the following chapters:

This text will be replaced by output generator with the chapter list.

Glossary

The Arm® Glossary is a list of terms used in Arm® documentation, together with definitions for those terms. The Arm® Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the [Arm® Glossary](#) for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names.
Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace italic

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>,
    <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm[®] Glossary*. For example, *IMPLEMENTATION DEFINED*, *IMPLEMENTATION SPECIFIC*, *UNKNOWN*, and *UNPREDICTABLE*.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

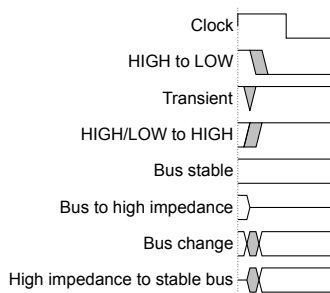


Figure 1: Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

Additional reading

Information published by Arm and by third parties.

This book contains information that is specific to this product. See the following documents for other relevant information.

Arm publications

- *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* (DDI 0487).
- *Arm® Cortex®#A76 Core Cryptographic Extension Technical Reference Manual* (100801).
- *Arm® Cortex®#A76 Core Configuration and Sign-off Guide* (100799).
- *Arm® Cortex®#A76 Core Integration Manual* (100800).
- *Arm® DynamIQ™ Shared Unit Integration Manual* (100455).
- *Arm® DynamIQ™ Shared Unit Technical Reference Manual* (100453).
- *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide* (100454).
- *Arm® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual* (100127).
- *AMBA® AXI and ACE Protocol Specification AXI3, AXI4, AXI5, ACE and ACE5* (IHI 0022).
- *AMBA® APB Protocol Version 2.0 Specification* (IHI 0024).
- *Arm® AMBA® 5 CHI Architecture Specification* (IHI 0050).
- *Arm® CoreSight™ Architecture Specification v3.0* (IHI 0029).
- *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (IHI 0031).
- *AMBA® 4 ATB Protocol Specification* (IHI 0032).
- *Arm® Generic Interrupt Controller Architecture Specification* (IHI 0069).
- *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* (IHI 0064).
- *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces* (IHI 0068).
- *Arm® Reliability, Availability, and Serviceability (RAS) Specification, Armv8, for the Armv8-A architecture profile* (DDI 0587A).

Other publications

- *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic.*



Note: Arm® floating-point terminology is largely based on the earlier ANSI/IEEE Std 754-1985 issue of the standard. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

Information about how to give feedback on the product.

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

Information about how to give feedback on the content.

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title .
- The number .
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.



Note: Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Part

I

Functional description

Topics:

- [Introduction](#)
- [Technical overview](#)
- [Clocks, resets, and input synchronization](#)
- [Power management](#)
- [Memory Management Unit](#)
- [Level 1 memory system](#)
- [Level 2 memory system](#)
- [Reliability, Availability, and Serviceability \(RAS\)](#)
- [Generic Interrupt Controller CPU interface](#)
- [Advanced SIMD and floating-point support](#)

This part describes the main functionality of the Cortex®-A76 core.

Chapter 1

Introduction

Topics:

- [About the core](#)
- [Features](#)
- [Implementation options](#)
- [Supported standards and specifications](#)
- [Test features](#)
- [Design tasks](#)
- [Product revisions](#)

This chapter provides an overview of the Cortex[®]-A76 core and its features.

About the core

The Cortex[®]-A76 core is a high-performance and low-power Arm product that implements the Arm[®]v8-A architecture.

The Cortex[®]-A76 *core* supports:

- The Arm[®]v8.2-A extension.
- The RAS extension.
- The Load acquire (LDAPR) instructions introduced in the Arm[®]v8.3-A extension
- The Dot Product support instructions introduced in the Arm[®]v8.4-A extension.
- The PSTATE *Speculative Store Bypass Safe* (SSBS) bit and the speculation barriers (CSDB, SSBB, PSSBB) instructions introduced in the Arm[®]v8.5-A extension.

The Cortex[®]-A76 has a *Level 1* (L1) memory system and a private, integrated *Level 2* (L2) cache. It also includes a superscalar, variable-length, out-of-order pipeline.

The Cortex[®]-A76 is implemented inside the *DynamiQ™ Shared Unit* (DSU) *cluster*. For more information, see the *Arm[®] DynamiQ™ Shared Unit Technical Reference Manual*.

The following figure shows an example of a configuration with four Cortex[®]-A76 s.

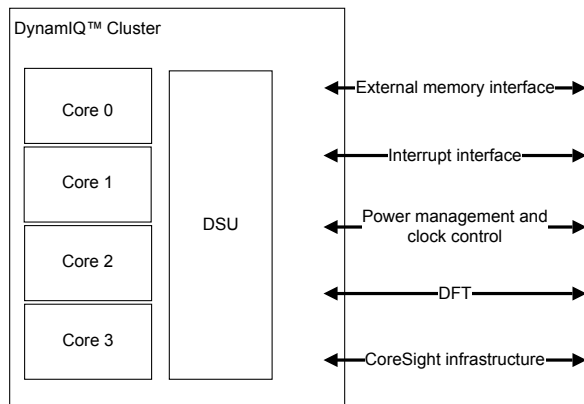


Figure 2: Example Cortex[®]-A76 configuration

Features

The Cortex[®]-A76 core includes the following features:

Core features

- Full implementation of the Armv8.2-A *A64*, *A32*, and *T32* instruction sets.
- Armv8.4 Dot Product instruction support.
- AArch32 execution state at *Exception level* EL0 only. *AArch64* execution state at all s (EL0 to EL3).
- Support for Arm TrustZone[®] technology.
- A *Memory Management Unit* (MMU).
- Superscalar, variable-length, out-of-order pipeline.
- 40-bit *Physical Address* (PA).
- An integrated execution unit that implements the *Advanced SIMD* and *floating-point* architecture support.
- Optional Cryptographic Extension.
- *Generic Interrupt Controller* (GICv4) CPU interface to connect to an external distributor.
- Generic Timers interface supporting 64-bit count input from an external system counter.

- *Reliability, Availability, and Serviceability* (RAS) Extension.

Cache features

- Separate L1 data and instruction caches.
- Private, unified data and instruction L2 cache.
- Optional L1 and L2 memory protection in the form of *Error Correcting Code* (ECC) or parity on all RAM instances.

Debug features

- Armv8.2 debug logic.
- *Performance Monitor Unit* (PMU).
- *Embedded Trace Macrocell* (ETM) that supports instruction trace only.
- *Activity Monitor Unit* (AMU).
- Optional *Coresight Embedded Logic Analyzer* (ELA).

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Implementation options

All Cortex®-A76 cores in the cluster must have the same build-time configuration options, except for the L2 cache size.

The following table lists the implementation options for a [core](#).

Table 1: Core implementation options

Feature	Range of options	Notes
L2 cache size	<ul style="list-style-type: none"> • 128KB • 256KB • 512KB 	-
L2 transaction queue size	<ul style="list-style-type: none"> • 24 entries • 36 entries • 48 entries 	There are two identical L2 banks in the Cortex®-A76 that can be configured with 12, 18, or 24 L2 transaction queue entries per L2 bank.
L1 and L2 ECC or parity protection	Can be included or not included.	L1 and L2 error protection can only be enabled if L3 cache error protection is enabled.
Cryptographic Extension	Can be included or not included.	-

Feature	Range of options	Notes
Core bus width	128-bit, 256-bit	<p>This specifies the bus width between the and the DSU CPU bridge. The legal bus width and master bus width combinations are:</p> <ul style="list-style-type: none"> If the bus width is 128 bits, the master bus interface can be any of the following options. <ul style="list-style-type: none"> Single 128-bit wide ACE interface. Dual 128-bit wide interfaces. Single 128-bit wide CHI interface. Single 256-bit wide CHI interface. If the bus width is 256 bits, the master bus interface is a single 256-bit wide CHI interface.
CoreSight <i>Embedded Logic Analyzer</i> (ELA)	Optional support	Support for integrating CoreSight ELA-500. CoreSight ELA-500 is a separately licensable product.
ELA RAM Address size	See the <i>Arm® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual</i> for the full supported range.	-
<i>Page Based Hardware Attributes</i> (PBHA)	<ul style="list-style-type: none"> Included Not included 	Support for PBHA. For more information, see Page Based Hardware Attributes

Supported standards and specifications

The Cortex®-A76 core implements the Arm®v8-A architecture and some architecture extensions. It also supports interconnect, interrupt, timer, debug, and trace architectures.

Table 2: Compliance with standards and specifications

Architecture specification or standard	Version	Notes
Arm architecture	Arm®v8-A	<ul style="list-style-type: none"> AArch32 execution state at Exception level EL0 only. AArch64 execution state at all s (EL0-EL3). A64, A32, and T32 instruction sets.

Architecture specification or standard	Version	Notes
Arm architecture extensions	<ul style="list-style-type: none"> Arm®v8.1-A extensions Arm®v8.2-A extensions Cryptographic extension RAS extension Arm®v8.3-A extensions Arm®v8.4-A dot product instructions Arm®v8.5-A extensions 	<ul style="list-style-type: none"> The Cortex®-A76 <i>core</i> implements the LDAPR instructions introduced in the Arm®v8.3-A extensions. The Cortex®-A76 implements the SDOT and UDOT instructions introduced in the Arm®v8.4-A extensions. The Cortex®-A76 implements the PSTATE <i>Speculative Store Bypass Safe</i> (SSBS) bit introduced in the Arm®v8.5-A extension.
Generic Interrupt Controller	GICv4	-
Generic Timer	Arm®v8-A	64-bit external system counter with timers within each .
PMU	PMUv3	-
Debug	Arm®v8-A	With support for the debug features added by the Arm®v8.2-A extensions.
CoreSight	CoreSightv3	-
Embedded Trace Macrocell	ETMv4.2	Instruction trace only.

See [\[ARM STANDARD\] Preface topic: Additional reading](#) for a list of architectural references.

Test features

The Cortex®-A76 core provides test signals that enable the use of both *Automatic Test Pattern Generation* (ATPG) and *Memory Built-In Self Test* (MBIST) to test the core logic and memory arrays.

Design tasks

The Cortex®-A76 core is delivered as a synthesizable *Register Transfer Level* (RTL) description in Verilog HDL. Before you can use the Cortex®-A76 core, you must implement it, integrate it, and program it.

A different party can perform each of the following tasks. Each task can include implementation and integration choices that affect the behavior and features of the *core*.

Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell. This task includes integrating RAMs into the design.

Integration

The integrator connects the macrocell into a SoC. This task includes connecting it to a memory system and peripherals.

Programming

In the final task, the system programmer develops the software to configure and initialize the and tests the application software.

The operation of the final *device* depends on the following:

Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and features of the resulting macrocell.

Configuration inputs

The integrator configures some features of the by tying inputs to specific values. These configuration settings affect the start-up behavior before any software configuration is made. They can also limit the options available to the software.

Software configuration

The programmer configures the by programming particular values into registers. The configuration choices affect the behavior of the .

Product revisions

This section indicates the first release and, in subsequent releases, describes the differences in functionality between product revisions.

r0p0

First release.

r1p0

Further development and optimization of the product, including updates to the L2 data RAM control inputs to allow multi-cycle hold timing constraints to ease timing closure.

r2p0

Includes Inter-Exception level isolation of branch predictor structures so that an Exception Level cannot train *branch prediction* for a different Exception Level to reliability hit in these trained prediction entries. Implemented new barrier SSBB.

r3p0

Implemented new barriers PSSBB and CSDB. Support for *Speculative Store Bypass Safe* (SSBS) bit enabling software to indicate whether hardware is permitted to load or store speculatively.

r3p1

No functional changes to *core* for this revision.

r4p0

Addition of PBHA support.

Chapter

2

Technical overview

Topics:

- [Components](#)
- [Interfaces](#)
- [About system control](#)
- [About the Generic Timer](#)

This chapter describes the structure of the Cortex[®]-A76 core.

Components

In a standalone configuration, the core consists of up to four cores and a DSU that connects the cores to an external memory system.

For more information about the DSU, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The main components of the Cortex®-A76 *core* are:

- Instruction fetch.
- Instruction decode.
- Register rename.
- Instruction issue.
- Execution pipelines.
- L1 data memory system.
- L2 memory system.

The following figure is an overview of the Cortex®-A76 .

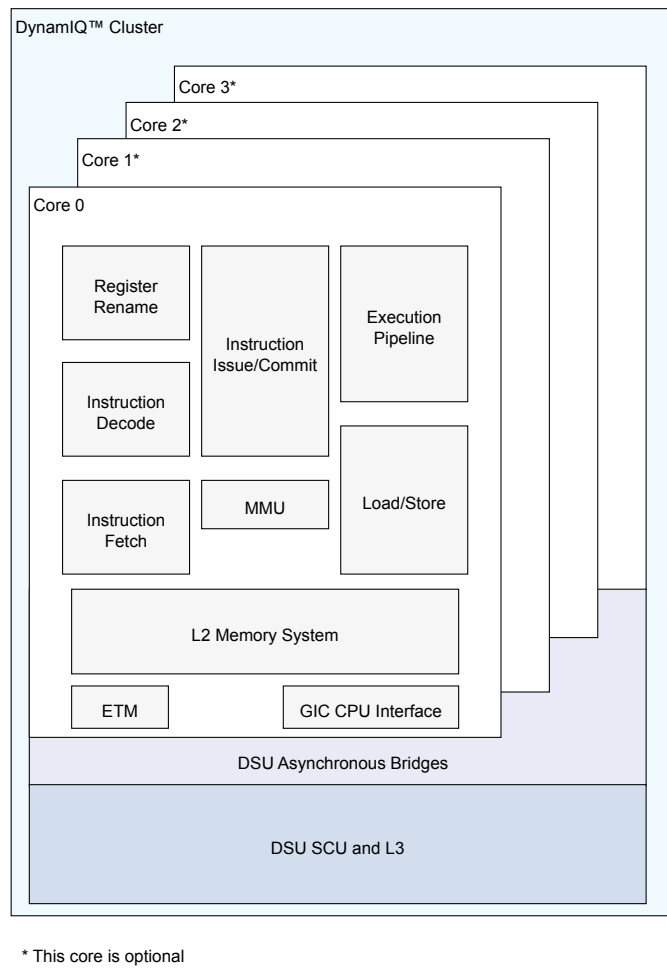


Figure 3: Cortex®-A76 overview



Note: There are multiple asynchronous bridges between the Cortex®-A76 and the DSU. Only the *coherent* interface between the Cortex®-A76 and the DSU can be configured to run synchronously, however it does not affect the other interfaces such as debug, trace, and *GIC* which are always asynchronous. For more information on how to set the interface to run either synchronously or asynchronously, see *Configuration Guidelines* in the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

-
-
-
-
-

-
-

Related reference

[Memory Management Unit](#) on page 51

This chapter describes the *Memory Management Unit* (MMU) of the Cortex®-A76 core.

[Level 1 memory system](#) on page 59

This chapter describes the L1 instruction cache and data cache that make up the L1 memory system.

[Level 2 memory system](#) on page 79

This chapter describes the L2 memory system.

[Generic Interrupt Controller CPU interface](#) on page 87

This chapter describes the Cortex®-A76 core implementation of the Arm *Generic Interrupt Controller* (GIC) CPU interface.

[Debug](#) on page 305

This chapter describes the Cortex®-A76 core debug registers and shows examples of how to use them.

[Performance Monitor Unit](#) on page 311

This chapter describes the *Performance Monitor Unit* (PMU) and the registers that it uses.

[Embedded Trace Macrocell](#) on page 333

This chapter describes the ETM for the Cortex®-A76 core.

Instruction fetch

The instruction fetch unit fetches instructions from the L1 instruction cache and delivers the instruction stream to the instruction decode unit.

The instruction fetch unit includes:

- A 64KB, 4-way, set associative L1 instruction cache with 64-byte cache lines and optional parity protection.
- A fully associative L1 instruction [TLB](#) with native support for 4KB, 16KB, 64KB, 2MB, and 32MB page sizes.
- A dynamic branch predictor.

Instruction decode

The instruction decode unit supports the A32, T32, and A64 instruction sets. It also supports Advanced SIMD and floating-point instructions in each instruction set.

Register rename

The register rename unit performs register renaming to facilitate out-of-order execution and dispatches decoded instructions to various issue queues.

Instruction issue

The instruction issue unit controls when the decoded instructions are dispatched to the execution pipelines. It includes issue queues for storing instruction pending dispatch to execution pipelines.

Execution pipeline

The execution pipeline includes:

- Integer execute unit that performs arithmetic and logical data processing operations.
- Vector execute unit that performs [Advanced SIMD](#) and [floating point](#) operations. Optionally, it can execute the cryptographic instructions.

L1 data memory system

The L1 data memory system executes load and store instructions and encompasses the L1 data side memory system. It also services memory coherency requests.

The load/store unit includes:

- A 64KB, 4-way, set associative L1 data cache with 64-byte cache lines and optional ECC protection per 32 bits.
- A fully associative L1 data [TLB](#) with native support for 4KB, 16KB, 64KB, 2MB, and 512MB page sizes.

L2 memory system

The L2 memory system services L1 instruction and data cache misses from the Cortex®-A76 core.

The L2 memory system includes:

- An 8-way set associative L2 cache with data ECC protection per 64 bits. The L2 cache is configurable with sizes of 128KB, 256KB, or 512KB.
- An interface with the DSU configurable at implementation time for synchronous or asynchronous operation.

Interfaces

The Cortex®-A76 core has several interfaces to connect it to a SoC. The DSU manages all interfaces.

For information on the interfaces, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

About system control

The system registers control and provide status information for the functions that the core implements.

The main functions of the system registers are:

- Overall system control and configuration.
- MMU configuration and management.
- Cache configuration and management.
- System performance monitoring.
- GIC configuration and management.

The system registers are accessible in the [AArch64](#) EL0-EL3 and [AArch32](#) EL0 Execution state. Some of the system registers are accessible through the external [debug interface](#).

About the Generic Timer

The Generic Timer can schedule events and trigger interrupts that are based on an incrementing counter value. It generates timer events as active-LOW interrupt outputs and event streams.

The Cortex®-A76 [core](#) provides a set of timer registers. The timers are:

- An EL1 Non-secure physical timer.
- An EL2 Hypervisor physical timer.
- An EL3 Secure physical timer.
- A virtual timer.
- A Hypervisor virtual timer.

The Cortex®-A76 does not include the system counter. This resides in the SoC. The system counter value is distributed to the with a 64-bit bus.

For more information on the Generic Timer, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* and the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter

3

Clocks, resets, and input synchronization

Topics:

- [About clocks, resets, and input synchronization](#)
- [Asynchronous interface](#)

This chapter describes the clocks, resets, and input synchronization of the Cortex[®]-A76 core.

About clocks, resets, and input synchronization

The Cortex®-A76 core supports hierarchical clock gating.

The Cortex®-A76 *core* contains several interfaces that connect to other components in the system. These interfaces can be in the same clock domain or in other clock domains.

For information about clocks, resets, and input synchronization, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Asynchronous interface

Your implementation can include an optional asynchronous interface between the core and the DSU top level.

See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information.

Chapter

4

Power management

Topics:

- [About power management](#)
- [Voltage domains](#)
- [Power domains](#)
- [Architectural clock gating modes](#)
- [Power control](#)
- [Core power modes](#)
- [Encoding for power modes](#)
- [Power domain states for power modes](#)
- [Power up and down sequences](#)
- [Debug over powerdown](#)

This chapter describes the power domains and the power modes in the Cortex[®]-A76 core.

About power management

The Cortex®-A76 core provides mechanisms to control both dynamic and static power dissipation.

Dynamic power management includes the following features:

- Architectural [clock gating](#).
- Per-core *Dynamic Voltage and Frequency Scaling* (DVFS).

Static power management includes the following features:

- Dynamic retention.
- Powerdown.

Related reference

[Power domain states for power modes](#) on page 48

The power domains can be controlled independently to give different combinations when powered-up and powered-down.

[Power control](#) on page 46

All power mode transitions are performed at the request of the power controller, using a P-Channel interface to communicate with the Cortex®-A76 core.

Voltage domains

The Cortex®-A76 core supports a VCPU voltage domain and a VSYS voltage domain.

The following figure shows the VCPU and VSYS voltage domains in each Cortex®-A76 *core* and in the DSU. The example shows a configuration with four Cortex®-A76 s.

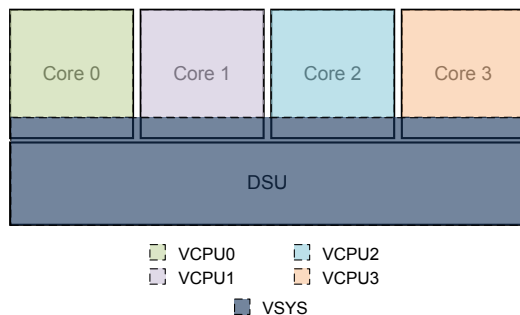


Figure 4: Cortex®-A76 voltage domains

Asynchronous bridge logic exists between the voltage domains. The Cortex®-A76 logic and clock domain of the asynchronous bridge are in the VCPU voltage domain. The DSU clock domain of the asynchronous bridge is in the VSYS voltage domain.



Note:

You can tie VCPU and VSYS to the same supply if one of the following conditions is met:

- The is configured to run synchronously with the DSU sharing the same clock.
- The is not required to support DVFS.

Power domains

The Cortex®-A76 core contains a core power domain (PDCPU), and a core top-level SYS power domain (PDSYS) where all the Cortex®-A76 core I/O signals go through.

The PDCPU power domain contains all `enyo_cpu` logic and part of the *core* asynchronous bridge that belongs to the VCPU domain. The *Advanced SIMD* and *floating-point* unit are included in the PDCPU power domain and is not supported as a separate power domain. The L1 and L2 RAMs are included in the PDCPU power domain and are not part of a separate power domain.

The PDSYS power domain contains the part of the asynchronous bridge that belongs to the DSU power domain.



Note: There are additional system power domains in the DSU. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for information.

The following table shows the power domain that the Cortex®-A76 supports.

Table 3: Power domain description

Power domain	Hierarchy	Description
PDCPU<n>	u_vcpu	<p>The domain includes the and block, the L1 and L2 <i>TLBs</i>, L1 and L2 cache RAMs, and Debug registers that are associated with the Cortex®-A76 .</p> <p><n> is the number of Cortex®-A76 s. The number represents 0, 1, 2, and 3. If a is not present, the corresponding power domain is not present.</p>
PDSYS	Top-level hierarchy and everything outside u_vcpu	<p>The domain is the interface between Cortex®-A76 and the DSU. It contains the <i>cluster</i> clock domain logic of the CPU bridge. The CPU Bridge contains all asynchronous bridges for crossing clock domains, and is split with one half of each bridge in the clock domain and the other half in the relevant domain. All I/O signals go through the CPU bridge and the SYS power domain.</p> <p>The domain is shared between the and hierarchies, and contains:</p> <ul style="list-style-type: none"> Anything outside of the power domain (u_vcpu hierarchy). u_cb_sys.

Clamping cells between power domains are inferred through power intent files rather than instantiated in the RTL.

The following figure shows an example of the organization of the power domains.

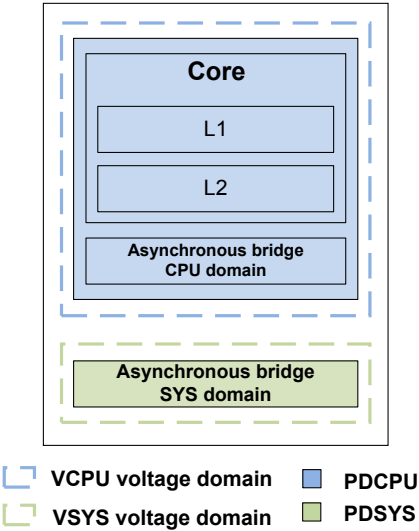


Figure 5: Cortex®-A76 power domain diagram at enyo_ level

The following figure shows the power domains in the DSU, where everything in the same color is part of the same power domain. The example shows four Cortex®-A76 s. The number of Cortex®-A76 s can vary, and the number of domains increases based on the number of Cortex®-A76 s present.

This example only shows the power domains that are associated with the Cortex®-A76 s, other power domains are required for a DSU.

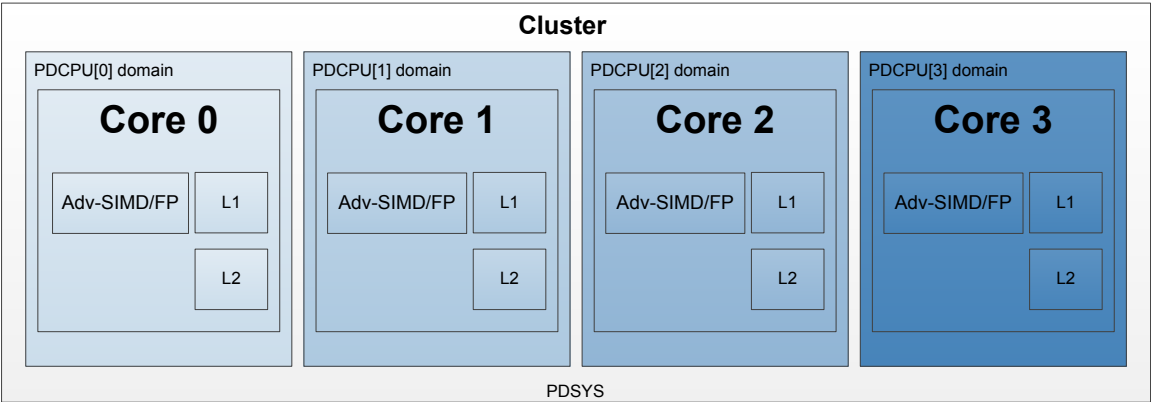


Figure 6: Cortex®-A76 power domains at enyo_ level

Architectural clock gating modes

When the Cortex®-A76 core is in standby mode, it is architecturally clock gated at the top of the clock tree.

Wait for Interrupt (WFI) and *Wait for Event* (WFE) are features of Arm®v8-A architecture that put the *core* in a low-power standby mode by architecturally disabling the clock at the top of the clock tree. The is fully powered and retains all the state in standby mode.

-
-

Core Wait for Interrupt

WFI puts the core in a low-power state by disabling most of the clocks in the core, while keeping the core powered up.

There is a small dynamic power overhead from the logic that is required to wake up the [core](#) from WFI low-power state. Other than this, the power that is drawn is reduced to static leakage current only.

When the executes the WFI instruction, the waits for all instructions in the to retire before it enters low-power state. The WFI instruction ensures that all explicit memory accesses that occurred before the WFI instruction in program order have retired.

In addition, the WFI instruction ensures that store instructions have updated the cache or have been issued to the L3 memory system.

While the is in WFI low-power state, the clocks in the are temporarily enabled without causing the to exit WFI low-power state when any of the following events are detected:

- An L3 snoop request that must be serviced by the data caches.
- A cache or [TLB](#) maintenance operation that must be serviced by the L1 instruction cache, data cache, , or L2 cache.
- An [APB](#) access to the debug or trace registers residing in the power domain.
- A [GIC](#) CPU access through the [AXI4](#) stream channel.

Exit from WFI low-power state occurs when one of the following occurs:

- The detects one of the WFI wake-up events.
- The detects a reset.

For more information, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Core Wait for Event

WFE is a feature of the Arm[®] v8-A architecture. It uses a locking mechanism based on events, to put the core in a low-power state by disabling most of the clocks in the core, while keeping the core powered up.

There is a small dynamic power overhead from the logic that is required to wake up the [core](#) from WFE low-power state. Other than this, the power that is drawn is reduced to static leakage current only.

A enters into WFE low-power state by executing the WFE instruction. When the WFE instruction executes, the waits for all instructions in the to complete before it enters the idle or low-power state.

If the event register is set, execution of WFE does not cause entry into standby state, but clears the event register.

While the is in WFE low-power state, the clocks in the are temporarily enabled without causing the to exit WFE low-power state when any of the following events are detected:

- An external snoop request that must be serviced by the data caches.
- A cache or [TLB](#) maintenance operation that must be serviced by the L1 instruction cache, data cache, , or L2 cache.
- An [APB](#) access to the debug or trace registers residing in the power domain.
- A [GIC](#) CPU access through the [AXI4](#) stream channel.

Exit from WFE low-power state occurs when one of the following occurs:

- The detects one of the WFE wake-up events.
- The EVENTI input signal is asserted.
- The detects a reset.

For more information, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Power control

All power mode transitions are performed at the request of the power controller, using a P-Channel interface to communicate with the Cortex®-A76 core.

There is one P-Channel per *core*, plus one P-Channel for the *cluster*. The Cortex®-A76 provides the current requirements on the PACTIVE signals, so that the power controller can make decisions and request any change with PREQ and PSTATE. The Cortex®-A76 then performs any actions necessary to reach the requested power mode, such as gating clocks, flushing caches, or disabling coherency, before accepting the request.

If the request is not valid, either because of an incorrect transition or because the status has changed so that state is no longer appropriate, then the request is denied. The power mode of each can be independent of other s in the , however the power mode is linked to the mode of the s.

Core power modes

The following figure shows the supported modes for each core domain P-Channel, and the legal transitions between them.

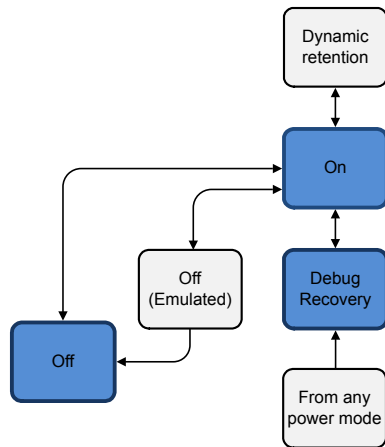


Figure 7: Cortex®-A76 *core* power domain mode transitions

The blue modes indicate the modes the channel can be initialized into.

-
-
-
-
-

On

In this mode, the core is on and fully operational.

The *core* can be initialized into the On mode. If the does not use P-Channel, you can tie the in the On mode by tying PREQ LOW.

When a transition to the On mode completes, all caches are accessible and *coherent*. Other than the normal architectural steps to enable caches, no additional software configuration is required.

When the domain P-Channel is initialized into the On mode, either as a shortcut for entering that mode or as a tie-off for an unused P-Channel, it is an assumed transition from the Off mode. This includes an invalidation of any cache RAM within the domain.

Off

The Cortex®-A76 core supports a full shutdown mode where power can be removed completely and no state is retained.

The shutdown can be for either the whole *cluster* or just for an individual *core*, which allows other s in the to continue operating.

In this mode, all logic and RAMs are off. The domain is inoperable and all state is lost. The L1 and L2 caches are disabled, flushed and the is removed from coherency automatically on transition to Off mode.

A *Cold reset* can reset the in this mode.

The P-Channel can be initialized into this mode.

An attempted debug access when the domain is off returns an error response on the internal *debug interface* indicating the is not available.

Off (emulated)

In this mode, all core domain logic and RAMs are kept on. However, core warm reset can be asserted externally to emulate a power off scenario while keeping core debug state and allowing debug access.

All debug registers must retain their mode and be accessible from the external debug interface. All other functional interfaces behave as if the *core* were Off.

Core dynamic retention

In this mode, all core logic and RAMs are in retention and the core domain is inoperable. The core can be entered into this power mode when it is in WFI or WFE mode.

The *core* dynamic retention can be enabled and disabled separately for WFI and WFE by software running on the . Separate timeout values can be programmed for entry into this mode from WFI and WFE mode:

- Use the CPUPWRCTLR.WFI_RET_CTRL register bits to program timeout values for entry into dynamic retention mode from WFI mode.
- Use the CPUPWRCTLR.WFE_RET_CTRL register bits to program timeout values for entry into dynamic retention mode from WFE mode.

When in dynamic retention and the is synchronous to the *cluster*, the clock to the is automatically gated outside of the domain. However, if the is running asynchronous to the , the system integrator must gate the clock externally during dynamic retention. For more information, see the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

The outputs of the domain must be isolated to prevent buffers without power from propagating *unknown* values to any operational parts of the system.

When the is in dynamic retention there is support for Snoop, *GIC*, and debug access, so the appears as if it were in WFI or WFE mode. When such an incoming access occurs, it stalls and the On PACTIVE bit is set HIGH. The incoming access proceeds when the domain is returned to On using P-Channel.

When the incoming access completes, and if the has not exited WFI or WFE mode, then the On PACTIVE bit is set LOW after the programmed retention timeout. The power controller can then request to reenter the dynamic retention mode.

Debug recovery mode

The debug recovery mode can be used to assist debug of external watchdog-triggered reset events.

It allows contents of the *core* L1 data cache that was present before the reset to be observable after the reset. The contents of the L1 cache are retained and are not altered on the transition back to the On mode.

By default, the *invalidates* its caches when *power-on reset* (nCPUPORESET) is deasserted. If the P-Channel is initialized to the debug recovery mode, and the is cycled through along with the system , then the cache invalidation is disabled. The cache contents are preserved when the is transitioned to the On mode.

Debug recovery mode also supports preserving RAS state, in addition to the cache contents. In this case, a transition to the debug recovery mode is made from any of the current states. Once in debug recovery mode, the is cycled through a warm reset with the system warm reset. The RAS and cache state are preserved when the is transitioned to the On mode.

This mode is strictly for debug purposes. It must not be used for functional purposes, as correct operation of the L1 cache is not guaranteed when entering this mode.



Note: This mode can occur at any time with no guarantee of the state of the . A P-Channel request of this type is accepted immediately, therefore its effects on the , *cluster*, or the wider system are *unpredictable*, and a wider system reset might be required. In particular, if there were outstanding memory system transactions at the time of the reset, then these may complete after the reset when the is not expecting them and cause a system deadlock.

Encoding for power modes

The following table shows the encodings for the supported modes for each core domain P-Channel.

Table 4: Core power modes COREPSTATE encoding

Power mode	Short name	<i>P</i> ACTIVE bit number	PSTATE value ¹	Power mode description
Debug recovery	DEBUG_RECOV	-	0b001010	Logic is off (or in reset), RAM state is retained and not <i>invalidated</i> when transitioning to On mode.
On	ON	8	0b001000	All powerup.
Core dynamic retention	FULL_RET	5	0b000101	Logic and RAM state are inoperable but retained.
Off (emulated)	OFF_EMU	1	0b000001	On with <i>Warm reset</i> asserted, debug state is retained and accessible.
Off	OFF	0 (implicit) ²	0b000000	All powerdown.

Power domain states for power modes

The power domains can be controlled independently to give different combinations when powered-up and powered-down.

However, only some powered-up and powered-down domain combinations are valid and supported. The following information shows the supported power domain states for the Cortex[®]-A76 *core*.


The PDCPU power domain supports the power states described in the following table.

¹ PSTATE[5:4] are don't care.

² It is tied off to 0 and should be inferred when all other CTIVE bits are LOW. For more information, see the *AMBA[®] Low Power Interface Specification Arm[®] Q-Channel and P-Channel Interfaces*.

Table 5: Power state description

Power state	Description
Off	Core off. Power to the block is gated.
Ret	Core retention. Logic and RAM retention power only.
On	Core on. Block is active.

 **CAUTION:** States that are not shown in the following tables are unsupported and must not occur.

The following table describes the power modes, and the corresponding power domain states for individual s. The power mode of each is independent of all other s in the *cluster*.

Table 6: Supported power domain states

Power mode	Power domain state	Description
Debug recovery	On	Core on.
On	On	Core on.
Core dynamic retention	Ret	Core in retention.
Off (emulated)	On	Core on.
Off	Off	Core off.

Deviating from the legal power modes can lead to *unpredictable* results. You must comply with the dynamic power management and powerup and powerdown sequences described in the following sections.

Power up and down sequences


The following approach allows taking the Cortex[®]-A76 cores in the cluster in and out of coherence.

Core powerdown

To take a *core* out of coherence *ready* for powerdown, the following power down steps must be performed:

1. Save all architectural states.
2. Configure the *GIC* distributor to disable or reroute interrupts away from this .
3. Set the CPUPWRCTLR.CORE_PWRDN_EN bit to 1 to indicate to the power controller that a powerdown is requested.
4. Execute an ISB instruction.
5. Execute a WFI instruction.

All L1 and L2 cache disabling, L1 and L2 cache flushing, and communication with the L3 memory system is performed in hardware after the WFI is executed, under the direction of the power controller.

 **Note:** Executing any WFI instruction when the CPUPWRCTLR.CORE_PWRDN_EN bit is set automatically masks out all interrupts and wake-up events in the . If executed when the CPUPWRCTLR.CORE_PWRDN_EN bit is set the WFI never wakes up and the needs to be reset to restart.

For information about *cluster* powerdown, see the *Arm[®] DynamIQ™ Shared Unit Technical Reference Manual*.

Core powerup

To bring a into coherence after reset, no software steps are required.

Debug over powerdown

The Cortex®-A76 core supports debug over powerdown, which allows a debugger to retain its connection with the core even when powered down. This enables debug to continue through powerdown scenarios, rather than having to re-establish a connection each time the core is powered up.

The debug over powerdown logic is part of the DebugBlock, which is external to the [cluster](#), and must remain powered on during the debug over powerdown process.

See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Chapter

5

Memory Management Unit

Topics:

- [About the MMU](#)
- [TLB organization](#)
- [TLB match process](#)
- [Translation table walks](#)
- [MMU memory accesses](#)
- [Specific behaviors on aborts and memory attributes](#)
- [Page Based Hardware Attributes](#)

This chapter describes the *Memory Management Unit* (MMU) of the Cortex[®]-A76 core.

About the MMU

The *Memory Management Unit* (MMU) is responsible for translating addresses of code and data *Virtual Addresses* (VA) to *Physical Addresses* (PAs) in the real system. The MMU also controls memory access permissions, memory ordering, and cache policies for each region of memory.

-
-

Main functions

The three main functions of the MMU are to:

- Control the table walk hardware that accesses *translation tables* in main memory.
- Translate *Virtual Addresses* (VAs) to *Physical Addresses* (PAs).
- Provide fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes that are held in translation tables.

Each stage of address translation uses a set of address translations and associated memory properties that are held in memory mapped tables called translation tables. Translation table entries can be cached into a *Translation Lookaside Buffer* (TLB).

The following table describes the components included in the *MMU*.

Table 7: s and caches in the

Component	Description
Instruction L1	48 entries, fully associative.
Data L1	48 entries, fully associative.
L2 cache	1280 entries, 5-way set associative.
Translation table prefetcher	Detects access to contiguous s and prefetches the next one. This prefetcher can be disabled in the <i>ECTLR</i> register.

The entries contain either one or both of a global indicator and an *Address Space Identifier* (ASID) to permit context switches without requiring the to be *invalidated*.

The entries contain a *Virtual Machine Identifier* (VMID) to permit virtual machine switches by the hypervisor without requiring the to be d.

AArch64 behavior

The Cortex[®]-A76 core is an Armv8 compliant core that supports execution in AArch64 state.

The following table shows the *AArch64* behavior.

Table 8: behavior

Address translation system	The Armv8 address translation system resembles an extension to the Long descriptor format address translation system to support the expanded virtual and <i>physical address</i> space.
Translation granule	4KB, 16KB, or 64KB for Armv8 <i>Virtual Memory System Architecture</i> (VMSA)
ASID size	8 or 16 bits depending on the value of TCR_ELx.AS.

VMID size	8 or 16 bits depending on the value of VTCR_EL2.VS.
PA size	Maximum 40 bits. Any configuration of TCR_ELx.IPS over 40 bits is considered as 40 bits. You can enable or disable each stage of the address translation independently.

The Cortex®-A76 core also supports the *Virtualization Host Extension* (VHE) including ASID space for EL2. When VHE is implemented and enabled, EL2 has the same behavior as EL1.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on concatenated translation tables and for address translation formats.

TLB organization

The TLB is a cache of recently executed page translations within the MMU. The Cortex®-A76 core implements a two-level TLB structure. The TLB stores all page sizes and is responsible for breaking these down in to smaller pages when required for the data or instruction L1 TLB.

-
-
-

Instruction L1 TLB

The instruction L1 TLB is implemented as a 48-entry fully associative structure. This TLB caches entries at the 4KB, 16KB, 64KB, 2MB, and 32MB granularity of VA to PA mapping only.

A hit in the instruction L1 *TLB* provides a single CLK cycle access to the translation, and returns the *PA* to the instruction cache for comparison. It also checks the access permissions to signal an *Instruction Abort*.

Data L1 TLB

The data L1 TLB is a 48-entry fully associative TLB that is used by load and store operations. The cache entries have 4KB, 16KB, 64KB, 2MB, and 512MB granularity of VA to PA mappings only.

A hit in the data L1 *TLB* provides a single CLK cycle access to the translation, and returns the *PA* to the data cache for comparison. It also checks the access permissions to signal a *Data Abort*.

L2 TLB

The L2 TLB structure is shared by instruction and data. It handles misses from the instruction and data L1 TLBs.

The following table describes the characteristic that applies to the L2 *TLB*.

Table 9: Characteristic of the L2

Characteristic	Note
5-way, set associative, 1280-entry cache	<p>Stores:</p> <ul style="list-style-type: none"> • VA to <i>PA</i> mappings for 4KB, 16KB, 64KB, 2MB, 32MB, 512MB, and 1GB block sizes. • <i>Intermediate physical address</i> (I) to mappings for 2MB and 1GB (in a 4KB translation granule), 32MB (in a 16K translation granule), and 512MB (in a 64K granule) block sizes. Only Non-secure EL1 and EL0 stage 2 translations are cached. • Intermediate s obtained during a <i>translation table</i> walk.

Access to the L2 usually takes three cycles. If a different page or block size mapping is used, then access to the L2 can take longer.

The L2 supports four walks in parallel (four misses), and can service two lookups while the walks are in progress. If there are six successive misses, the L2 will stall.



Note: Caches in the *core* are *invalidated* automatically at reset deassertion unless the power mode is initialized to Debug Recovery. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information.

TLB match process

The Armv8-A architecture provides support for multiple maps from the VA space that are translated differently.

TLB entries store the context information required to facilitate a match and avoid the need for a flush on a context or virtual machine switch.

Each entry contains a:

- *VA*.
- *PA*.
- Set of memory properties that include type and access permissions.

Each entry is either associated with a particular ASID or global. In addition, each entry contains a field to store the VMID in the entry applicable to accesses from Non-secure EL0 and EL1 *Exception levels*.

Each entry is associated with a particular translation regime.

- EL3 in Secure state in *AArch64* state only.
- EL2 or EL0 in Non-secure state.
- EL1 or EL0 in Secure state.
- EL1 or EL0 in Non-secure state.

A match entry occurs when the following conditions are met:

- Its , moderated by the page size such as the bits[48:N], where N is log₂ of the block size for that translation that is stored in the entry, matches the requested address.
- Entry translation regime matches the current translation regime.
- The ASID matches the current ASID held in the CONTEXTIDR, TTBR0, or TTBR1 register, or the entry is marked global.
- The VMID matches the current VMID held in the VTTBR_EL2 register.

- The ASID and VMID matches are *ignored* when ASID and VMID are not relevant.

ASID is relevant when the translation regime is:

- EL2 in Non-secure state with HCR_EL2.E2H and HCR_EL2.TGE set to 1.
- EL1 or EL0 in Secure state.
- EL1 or EL0 in Non-secure state.

VMID is relevant for EL1 or EL0 in Non-secure state.

Translation table walks

When the Cortex®-A76 core generates a memory access, the MMU:

1. Performs a lookup for the requested *VA*, current ASID, current VMID, and current translation regime in the relevant instruction or data.
2. If there is a miss in the relevant L1 *TLB*, the *MMU* performs a lookup for the requested , current ASID, current VMID, and translation regime.
3. If there is a miss in the L2 , the performs a hardware *translation table* walk.

In the case of a L2 miss, the hardware does a walk as long as the is enabled, and the translation using the *base register* has not been disabled.

If the walk is disabled for a particular base register, the *core* returns a Translation Fault. If the finds a matching entry, it uses the information in the entry as follows.

The access permission bits determine if the access is permitted. If the matching entry does not pass the permission checks, the signals a Permission *fault*. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for details of Permission s, including:

- A description of the various s.
- The codes.
- Information regarding the registers where the codes are set.
-

AArch64 behavior

When executing in AArch64 state at a particular Exception level, you can configure the hardware translation table walk to use either the 4KB, 16KB, or 64KB translation granule. Program the Translation Granule bit, TG0, in the appropriate translation control register:

- TCR_EL1.
- TCR_EL2.
- TCR_EL3.
- VTCR_EL2.

For TCR_EL1, you can program the Translation Granule bits TG0 and TG1 to configure the translation granule respectively for TTBR0_EL1 and TTBR1_EL1, or TCR_EL2 when VHE is enabled.

MMU memory accesses

During a translation table walk, the MMU generates accesses. This section describes the specific behaviors of the core for MMU memory accesses.

-
-

Configuring MMU accesses

By programming the IRGN and ORGN bits, you can configure the MMU to perform translation table walks in cacheable or non-cacheable regions:

AArch64

Appropriate TCR_ELx register.

If the encoding of both the ORGN and IRGN bits is *Write-Back*, the data cache lookup is performed and data is *read* from the data cache. External memory is accessed, if the ORGN and IRGN bit contain different attributes, or if the encoding of the ORGN and IRGN bits is *-Through* or *Non-cacheable*.

Descriptor hardware update

The core supports hardware update in AArch64 state using hardware management of the access flag and hardware management of dirty state.

These features are enabled in registers TCR_ELx and VTCR_EL2.

Hardware management of the Access flag is enabled by the following configuration fields:

- TCR_ELx.HA for stage 1 translations.
- VTCR_EL2.HA for stage 2 translations.

Hardware management of *dirty* state is enabled by the following configuration fields:

- TCR_ELx.HD for stage 1 translations.
- VTCR_EL2.HD for stage 2 translations.



Note: Hardware management of state can only be enabled if hardware management of the Access flag is enabled.

To support the hardware management of state, the DBM field is added to the *translation table* descriptors as part of Armv8.1 architecture.

The *core* supports hardware update only in outer *Write-Back* and inner *-Back* memory *regions*.

If software requests a hardware update in a memory that is not inner *-Back* or not outer *-Back*, then the returns an *abort* with the following encoding:

- ESR_ELx.DFSC = 110001 for *Data Aborts* in *AArch64*.
- ESR_ELx.IFSC = 110001 for *Instruction Aborts* in .

Specific behaviors on aborts and memory attributes

This section describes specific behaviors caused by aborts and also describes memory attributes.

MMU responses

When one of the following translation is completed, the generates a response to the requester:

- A L1 *TLB* hit.
- A L2 hit.
- A *translation table* walk.

The response from the contains the following information:

- The *PA* corresponding to the translation.
- A set of permissions.
- Secure or Non-secure.
- All the information required to report *aborts*. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more details.
-

-
-
-

External aborts

External aborts are defined as those that occur in the memory system rather than those that the MMU detects. Normally, external memory aborts are rare. External aborts are caused by errors flagged to the external interface.

When an external *abort* to the external interface occurs on an access for a *translation table* walk access, the *MMU* returns a synchronous external . For a load multiple or store multiple operation, the address captured in the *fault* address register is that of the address that generated the synchronous external .

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Mis-programming contiguous hints

In the case of a mis-programming contiguous hint, when there is a descriptor that contains a set CH bit, all contiguous VAs contained in this block should be included in the input VA address space that is defined for stage 1 by TxSZ for TTbX or for stage 2 by {SL0, T0SZ}.

The Cortex®-A76 *core* treats such a block as not causing a translation *fault*.

Memory attributes

The memory region attributes specified in the TLB entry, or in the descriptor in case of translation table walk, determine if the access is:

- Normal Memory or Device type.
- One of the four different *device* memory types that are defined for Armv8:

Device-nGnRnE	Device non-Gathering, non-Reordering, No Early <i>Write</i> Acknowledgment.
Device-nGnRE	Device non-Gathering, non-Reordering, Early Acknowledgment.
Device-nGRE	Device non-Gathering, Reordering, Early Acknowledgment.
Device-GRE	Device Gathering, Reordering, Early Acknowledgment.

In the Cortex®-A76 *core*, a page is *cacheable* only if the inner memory attribute and outer memory attribute are Back. In all other cases, all pages are downgraded to Non- Normal memory.

When the *MMU* is disabled at stage 1 and stage 2, and SCTLR.I is set to 1, instruction prefetches are cached in the instruction cache but not in the unified cache. In all other cases, normal behavior on memory attribute applies.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on *translation table* formats.

Page Based Hardware Attributes

Page Based Hardware Attributes (PBHA) is an optional, implementation defined feature.

It allows software to set up to two bits in the *translation tables*, which are then propagated through the memory system with transactions, and can be used in the system to control system components. The meaning of the bits is specific to the system design.

For information on how to set and enable the PBHA bits in the s, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*. When disabled, the PBHA value that is propagated on the bus is 0.

For memory accesses caused by a walk, the AHTCR, ATTBCR, and AVTCR registers control the PBHA values.

PBHA combination between stage 1 and stage 2 on memory accesses

PBHA should always be considered as an attribute of the *physical address*.

When stage 1 and stage 2 are enabled:

- If both stage 1 PBHA and stage 2 PBHA are enabled, the final PBHA is stage 2 PBHA.
- If stage 1 PBHA is enabled and stage 2 PBHA is disabled, the final PBHA is stage 1 PBHA.
- If stage 1 PBHA is disabled and stage 2 PBHA is enabled, the final PBHA is stage 2 PBHA.
- If both stage 1 PBHA and stage 2 PBHA are disabled, the final PBHA is defined to 0.

Enable of PBHA has granularity of one bit, so this property is applied independently on each PBHA bit.

Mismatched aliases

If the same is accessed through more than one *virtual address* mapping, and the PBHA bits are different in the mappings, then the results are *unpredictable*. The PBHA value sent on the bus could be for either mapping.

Page Based Hardware Attributes

Page Based Hardware Attributes (PBHA) is an optional, implementation defined feature.

It allows software to set up to two bits in the *translation tables*, which are then propagated through the memory system with transactions, and can be used in the system to control system components. The meaning of the bits is specific to the system design.

For information on how to set and enable the PBHA bits in the s, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*. When disabled, the PBHA value that is propagated on the bus is 0.

For memory accesses caused by a walk, the AHTCR, ATTBCR, and AVTCR registers control the PBHA values.

PBHA combination between stage 1 and stage 2 on memory accesses

PBHA should always be considered as an attribute of the *physical address*.

When stage 1 and stage 2 are enabled:

- If both stage 1 PBHA and stage 2 PBHA are enabled, the final PBHA is stage 2 PBHA.
- If stage 1 PBHA is enabled and stage 2 PBHA is disabled, the final PBHA is stage 1 PBHA.
- If stage 1 PBHA is disabled and stage 2 PBHA is enabled, the final PBHA is stage 2 PBHA.
- If both stage 1 PBHA and stage 2 PBHA are disabled, the final PBHA is defined to 0.

Enable of PBHA has granularity of one bit, so this property is applied independently on each PBHA bit.

Mismatched aliases

If the same is accessed through more than one *virtual address* mapping, and the PBHA bits are different in the mappings, then the results are *unpredictable*. The PBHA value sent on the bus could be for either mapping.

Chapter

6

Level 1 memory system

Topics:

- [About the L1 memory system](#)
- [L1 instruction memory system](#)
- [L1 data memory system](#)
- [Data prefetching](#)
- [Direct access to internal memory](#)

This chapter describes the L1 instruction cache and data cache that make up the L1 memory system.

About the L1 memory system

The Cortex[®]-A76 L1 memory system is designed to enhance core performance and save power.

The L1 memory system consists of separate instruction and data caches. Both have a fixed size of 64KB.

-
-

L1 instruction-side memory system

The L1 instruction memory system has the following key features:

- *Virtually Indexed, Physically Tagged* (VIPT), which behaves as a *Physically Indexed, Physically Tagged* (PIPT) 4-way set-associative L1 data cache.
- Fixed cache line length of 64 bytes.
- Pseudo-LRU cache replacement policy.
- 256-bit *read* interface from the L2 memory system.

L1 data-side memory system

The L1 data memory system has the following features:

- *Virtually Indexed, Physically Tagged* (VIPT), which behaves as a *Physically Indexed, Physically Tagged* (PIPT) 4-way set-associative L1 data cache.
- Fixed cache line length of 64 bytes.
- Pseudo-LRU cache replacement policy.
- 256-bit write interface from the L2 memory system.
- 256-bit *read* interface from the L2 memory system.
- Two 128-bit paths from the data L1 memory system to the datapath.
- 256-bit write path from the datapath to the L1 memory system.

L1 instruction memory system

The L1 instruction side memory system provides an instruction stream to the decoder.

To increase overall performance and to reduce power consumption, it uses:

- Dynamic *branch prediction*.
- Instruction caching.
-

L1 data memory system

The L1 data cache is organized as a *Virtually Indexed, Physically Tagged* (VIPT) cache featuring four ways.

Data cache *invalidate* on reset

The Armv8-A architecture does not support an operation to the entire data cache. If software requires this function, it must be constructed by iterating over the cache geometry and executing a series of individual by set/way instructions.

-
-

Memory system implementation

This section describes the implementation of the L1 memory system.

Limited Order Regions

The *core* offers support for four limited ordering *region* descriptors, as introduced by the Armv8.1 Limited Ordering Regions.

Atomic instructions

The Cortex®-A76 supports the atomic instructions added in Armv8.1 architecture.

Atomic instructions to *cacheable* memory can be performed as either near atomics or far atomics, depending on where the cache line containing the data resides.

When an instruction hits in the L1 data cache in a unique state, then it is performed as a near atomic in the L1 memory system. If the atomic operation misses in the L1 cache, or the line is shared with another, then the atomic is sent as a far atomic on the CHI interface.

If the operation misses everywhere within the *cluster*, and the interconnect supports far atomics, then the atomic is passed on to the interconnect to perform the operation.

When the operation hits anywhere inside the, or when an interconnect does not support atomics, the L3 memory system performs the atomic operation. If the line it is not *already* there, it allocates the line into the L3 cache. This depends on whether the DSU is configured with an L3 cache.

Therefore, if software prefers that the atomic is performed as a near atomic, precede the atomic instruction with a PLDW or PRFM PSTL1KEEP instruction.

Alternatively, the CPUECTLR can be programmed such that different types of atomic instructions attempt to execute as a near atomic. One cache fill will be made on an atomic. If the cache line is lost before the atomic operation can be made, it will be sent as a far atomic.

The Cortex®-A76 supports atomics to *device* or non- memory, however this relies on the interconnect also supporting atomics. If such an atomic instruction is executed when the interconnect does not support them, it will result in an *abort*.

For more information on the CPUECTLR register, see [CPUECTLR_EL1, CPU Extended Control Register, EL1](#).

LDAPR instructions

The supports Load acquire instructions adhering to the RCpc consistency semantic introduced in the Armv8.3 extensions for A profile. This is reflected in register ID_AA64ISAR1_EL1 where bits[23:20] are set to 0001 to indicate that the supports LDAPRB, LDAPRH, and LDAPR instructions implemented in *AArch64*.

Transient memory

The has a specific behavior for memory s that are marked as write-back and transient, as defined in the Armv8.0 architecture.

For any load or store that is targeted at a memory that is marked as transient, the following occurs:

- If the memory access misses in the L1 data cache, the returned cache line is allocated in the L1 data cache but is marked as transient.
- When the line is evicted from the L1 data cache, the transient hint is passed to the L2 cache so that the replacement policy will not attempt to retain the line. When the line is subsequently evicted from the L2 cache, it will bypass the next level cache entirely.

Non-temporal loads

Non-temporal loads indicate to the caches that the data is likely to be used for only short periods. For example, when streaming single-use data that is then discarded. In addition to non-temporal loads, there are also prefetch-memory (PRFM) [hint instructions](#) with the STRM qualifier.

Non-temporal loads to memory which are designated as [Write-Back](#) are treated the same as loads to Transient memory.

Data prefetching

This section describes the data prefetching behavior for the Cortex[®]-A76 core.

Preload instructions

The Cortex[®]-A76 [core](#) supports the [AArch64 Prefetch Memory](#) (PRFM) instructions and the [AArch32 Prefetch Data](#) (PLD) and [Preload Data With Intent To Write](#) (PLDW) instructions. These instructions signal to the memory system that memory accesses from a specified address are likely to occur soon. The memory system acts by taking actions that aim to reduce the latency of the memory access when they occur. PRFM instructions perform a lookup in the cache, and if they miss and are to a [cacheable](#) address, a linefill starts. However, the PRFM instruction retires when its linefill is started, rather than waiting for the linefill to complete. This enables other instructions to execute while the linefill continues in the background.

The [Preload Instruction](#) (PLI) memory system hint performs preloading in the L2 cache for accesses if they miss in both the L1 instruction cache and L2 cache. Instruction preloading is performed in the background.

For more information about prefetch memory and preloading caches, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Data [prefetching](#) and monitoring

The load-store unit includes a hardware prefetcher that is responsible for generating prefetches targeting both the L1 and the L2 cache. The load side prefetcher uses the [virtual address](#) to prefetch to both the L1 and L2 Cache. The store side prefetcher uses the [physical address](#), and only prefetches to the L2 Cache.

The CPUECTLR register allows you to have some control over the prefetcher. See [CPUECTLR_EL1, CPU Extended Control Register, EL1](#) for more information on the control of the prefetcher.

Use the prefetch memory system instructions for data where short sequences or irregular pattern fetches are required.

Data cache zero

The Armv8-A architecture introduces a [Data Cache Zero by Virtual Address](#) (DC ZVA) instruction.

In the Cortex[®]-A76, this enables a block of 64 bytes in memory, aligned to 64 bytes in size, to be set to zero.

For more information, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Direct access to internal memory

The Cortex[®]-A76 core provides a mechanism to read the internal memory that is used by the L1 caches, L2 cache, and TLB structures through *implementation defined* system registers. This functionality can be useful when debugging software or hardware issues.

When the [core](#) executes in [AArch64](#) state, there are six [read-only](#) registers that are used to access the contents of the internal memory. The internal memory is selected by programming the implementation-defined RAMINDEX register (using SYS #6, c15, c0, #0 instruction). These operations are available only in EL3. In all other modes, executing these instructions results in an Undefined Instruction [exception](#). The data is from -only registers as shown in the following table.

Table 10: registers used to access internal memory

Register name	Function	Access	Operation	Rd Data
IDATA0_EL3	Instruction Register 0	Read-only	S3_6_c15_c0_0	Data
IDATA1_EL3	Instruction Register 1	Read-only	S3_6_c15_c0_1	Data
IDATA2_EL3	Instruction Register 2	Read-only	S3_6_c15_c0_2	Data
DDATA0_EL3	Data Register 0	Read-only	S3_6_c15_c1_0	Data
DDATA1_EL3	Data Register 1	Read-only	S3_6_c15_c1_1	Data
DDATA2_EL3	Data Register 2	Read-only	S3_6_c15_c1_2	Data

•
•
•
•

Encoding for L1 instruction cache tag, L1 instruction cache data, L1 BTB, L1 GHB, L1 TLB instruction, and BPIQ

The following tables show the encoding required to select a given cache line.

Table 11: L1 instruction cache tag location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 00
[23:20]	Reserved
[19:18]	Way
[17:14]	Reserved
[13:6]	Index [13:6]
[5:0]	Reserved

Table 12: L1 instruction cache data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 01
[23:20]	Reserved
[19:18]	Way
[17:14]	Reserved
[13:3]	Index [13:3]
[2:0]	Reserved

Table 13: L1 BTB data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 02

Bit fields of Rd	Description
[23:20]	Reserved
[19:18]	Way
[17:15]	Reserved
[14:5]	Index [14:5]
[4:0]	Reserved

Table 14: L1 GHB data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 03
[23:14]	Reserved
[13:4]	Index [13:4]
[3:0]	Reserved

Table 15: L1 instruction *TLB* data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 04
[23:8]	Reserved
[7:0]	Entry (<=47)

Table 16: BPIQ data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 05
[23:10]	Reserved
[9:4]	Index [5:0]
[3:0]	Reserved

The following table shows the data that is returned from accessing the L1 instruction tag RAM.

Table 17: L1 instruction cache tag format

Register	Bit field	Description
Instruction Register 0	[31]	Non-secure identifier for the physical address
	[30:3]	Physical address [39:12]

Register	Bit field	Description
	[2:1]	Instruction state [1:0]
		00 Invalid
		01 T32
		10 A32
		11 A64
	[0]	Parity
Instruction Register 1	[63:0]	0
Instruction Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 instruction data RAM.

Table 18: L1 instruction cache data format

Register	Bit field	Description
Instruction Register 0	[63:0]	Data [63:0]
Instruction Register 1	[63:9]	0
	[8]	Parity
	[7:0]	Data [71:64]
Instruction Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 BTB RAM.

Table 19: L1 BTB cache format

Register	Bit field	Description
Instruction Register 0	[63:0]	Data [63:0]
Instruction Register 1	[63:18]	0
	[17:0]	Data [81:64]
Instruction Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 GHB RAM.

Table 20: L1 GHB cache format

Register	Bit field	Description
Instruction Register 0	[63:0]	Data [63:0]
Instruction Register 1	[63:32]	0
	[31:0]	Data [95:64]
Instruction Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 instruction RAM.

Table 21: L1 instruction cache format

Register	Bit field		Description
	CORE_PBHA=FALSE	CORE_PBHA=TRUE	
Instruction Register 0	[63:59]	[63:59]	Virtual address [16:12]
	[58:56]	-	attribute
	-	[58:57]	PBHA[1:0]
	-	[56]	attribute
	[55:53]	[55:53]	Memory attributes:
			000 Device nGnRnE
			001 Device nGnRE
			010 Device nGRE
			011 Device GRE
			100 Non-cacheable
			101 <i>Write-Back</i> No-Allocate
			110 -Back Transient
			111 -Back <i>Read-Allocate</i> and - Allocate
	[52:50]	[52:50]	Page size:
			000 4KB
			001 16KB
			010 64KB
			011 256KB
			100 2MB
			101 32MB
			11x Reserved
	[49:46]	[49:46]	attribute
	[45]	[45]	Outer-shared
	[44]	[44]	Inner-shared
	[43:39]	[43:39]	attribute

Register	Bit field		Description
	CORE_PBHA=FALSE	CORE_PBHA=TRUE	
Instruction Register 1	[38:23]	[38:23]	ASID
	[22:7]	[22:7]	VMID
	[6:5]	[6:5]	Translation regime:
			00 Secure EL1/ EL0
			01 Secure EL3
			10 Non-secure EL1/EL0
			11 Non-secure EL2
	[4:1]	[4:1]	attribute
	[0]	[0]	Valid
	[60]	[60]	Non-secure
	[59:32]	[59:32]	Physical address [39:12]
	[31:0]	[31:0]	Virtual address[48:17]

The following table shows the data that is returned from accessing the BPIQ RAM.

Table 22: BPIQ cache format

Register	Bit field	Description
Instruction Register 0	[63:0]	Data [63:0]
Instruction Register 1	[63:32]	0
	[31:0]	Data [95:64]
Instruction Register 2	[63:0]	0

Encoding for L1 data cache tag, L1 data cache data, and L1 TLB data

The core data cache consists of a 4-way set-associative structure.

The encoding, which is set in *Rd* in the appropriate MCR instruction, used to locate the required cache data entry for tag, data, and *TLB* memory is shown in the following tables. It is similar for both the tag RAM, data RAM, and access. Data RAM access includes an additional field to locate the appropriate *doubleword* in the cache line.

Tag RAM encoding includes an additional field to select which one of the two cache channels must be used to perform any access.

Table 23: L1 data cache tag location encoding

Bit fields of <i>Rd</i>	Description
[31:24]	RAMID = 08
[23:20]	Reserved
[19:18]	Way

Bit fields of Rd	Description
[17]	Copy
	<div>0</div> <div>Tag RAM associated with Pipe 0</div>
	<div>1</div> <div>Tag RAM associated with Pipe 1</div>
[16:14]	Reserved
[13:6]	Index [13:6]
[5:0]	Reserved

Table 24: L1 data cache data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 09
[23:20]	Reserved
[19:18]	Way
[17:16]	BankSel
[15:14]	Unused
[13:6]	Index [13:6]
[5:0]	Reserved

Table 25: L1 data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 0A
[23:6]	Reserved
[5:0]	Entry (0->47)

Data cache *reads* return 64 bits of data in Data Register 0, Data Register 1, and Data Register 2. If cache protection is supported, Data Register 2 is used to report ECC information using the format shown in the following tables.

The following table shows the data that is returned from accessing the L1 data cache tag RAM with ECC.

Table 26: L1 data cache tag format with ECC

Register	Bit field	Description
Data Register 0	[63:41]	0
	[40:34]	ECC
	[33]	Non-secure identifier for the <i>physical address</i>

Register	Bit field	Description
	[32:5]	Physical address [39:12]
	[4:3]	Reserved
	[2]	Transient/WBNA
	[1:0]	MESI
		00 Invalid
		01 Shared
		10 Exclusive
		11 Modified with respect to the L2 cache
Data Register 1	[63:0]	0
Data Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 data cache tag RAM without ECC.

Table 27: L1 data cache tag format without ECC

Register	Bit field	Description
Data Register 0	[63:34]	0
	[33]	Non-secure identifier for the
	[32:5]	Physical address [39:12]
	[4:3]	Reserved
	[2]	Transient/WBNA
	[1:0]	MESI
		00 Invalid
		01 Shared
		10 Exclusive
		11 Modified
Data Register 1	[63:0]	0
Data Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 data cache data RAM with ECC.

Table 28: L1 data cache data format with ECC

Register	Bit field	Description
Data Register 0	[63:0]	Word1_data [31:0], Word0_data [31:0]

Register	Bit field	Description
Data Register 1	[63:0]	Word3_data [31:0], Word2_data [31:0]
Data Register 2	[63:32]	0
	[31:0]	Word3_poison, Word3_ecc [6:0], Word2_poison, Word2_ecc [6:0], Word1_poison, Word1_ecc [6:0], Word0_poison, Word0_ecc [6:0]

The following table shows the data that is returned from accessing the L1 data cache data RAM without ECC.

Table 29: L1 data cache data format without ECC

Register	Bit field	Description
Data Register 0	[63:0]	Word1_data [31:0], Word0_data [31:0]
Data Register 1	[63:0]	Word3_data [31:0], Word2_data [31:0]
Data Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L1 data RAM.

Table 30: L1 data cache format

Register	Bit field		Description
	CORE_PBHA=FALSE	CORE_PBHA=TRUE	
Data Register 0	[63:62]	[63:62]	Virtual address [13:12]
	[58]	[58]	Outer-shared
	[57]	[57]	Inner-shared

Register	Bit field		Description
	CORE_PBHA=FALSE	CORE_PBHA=TRUE	
	[52:50]	[52:50]	Memory attributes:
			000 Device nGnRnE
			001 Device nGnRE
			010 Device nGRE
			011 Device GRE
			100 Non-cacheable
			101 <i>Write-Back</i> No-Allocate
			110 -Back Transient
			111 -Back <i>Read-Allocate</i> and - Allocate
	[38:36]	[38:36]	Page size:
			000 4KB
			001 16KB
			010 64KB
			011 256KB
			100 Reserved
			101 2MB
			110 512MB
			111 Reserved
	[35]	[35]	Non-secure
	[34:33]	[34:33]	Translation regime:
			00 Secure EL1/EL0
			01 Secure EL3
			10 Non-secure EL1/EL0
			11 Non-secure EL2

Register	Bit field		Description
	CORE_PBHA=FALSE	CORE_PBHA=TRUE	
Data Register 1	[32:17]	[32:17]	ASID
	[16:1]	[16:1]	VMID
	[0]	[0]	Valid
	-	[63]	PBHA[0]
	[62:35]	[62:35]	Physical address [39:12]
Data Register 2	[34:0]	Virtual address	Virtual address[48:14]
	-	[0]	PBHA[1]

Encoding for the L2 unified cache

The following tables show the encoding required to select a given cache line.

Table 31: L2 tag location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 10
[23:21]	Reserved
[20:18]	Way (0->7)
[17:16]	Reserved
[15:6]	Index[15:6]
[5:0]	Reserved

Table 32: L2 data location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 11
[23:21]	Reserved
[20:18]	Way (0->7)
[17:16]	Reserved
[15:4]	Index[15:4]
[3:0]	Reserved

Table 33: L2 *victim* location encoding

Bit fields of Rd	Description
[31:24]	RAMID = 12
[23:16]	Reserved
[15:6]	Index[15:6]
[5:0]	Reserved

The following table shows the data that is returned from accessing the L2 tag RAM when L2 is configured with a 128KB cache size.

Table 34: L2 tag format with a 128KB L2 cache size

Register	Bit field	Description
Data Register 0	[63:45]	0
	[44:38]	ECC [6:0] if configured with ECC for a 128KB L2 cache size, otherwise 0
	[37:12]	Physical address [39:14]
	[11]	Non-secure identifier for the <i>physical address</i>
	[10:9]	Virtual index [13:12]
	[8:6]	Reserved
	[5]	Shareable
	[4]	Outer allocation hint
	[3]	L1 data cache valid
	[2:0]	L2 State
		101 Modified 001 Exclusive x11 Shared xx0 Invalid
Data Register 1	[63:0]	0
Data Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L2 tag RAM when L2 is configured with a 256KB cache size.

Table 35: L2 tag format with a 256KB L2 cache size

Register	Bit field	
	CORE_PBHA=FALSE	CORE_PBHA=TRUE
Data Register 0	[63:46]	[63:46]
	[45:44]	-
	[43:37]	[45:39]
	-	[38:37]
	[36:12]	[36:12]
	[11]	[11]
	[10:9]	[10:9]
	[8:6]	[8:6]

Register	Bit field	
	CORE_PBHA=FALSE	CORE_PBHA=TRUE
	[5]	[5]
	[4]	[4]
	[3]	[3]
	[2:0]	[2:0]
Data Register 1	[63:0]	[63:0]
Data Register 2	[63:0]	[63:0]

The following table shows the data that is returned from accessing the L2 tag RAM when L2 is configured with a 512KB cache size.

Table 36: L2 tag format with a 512KB L2 cache size

Register	Bit field	Description
Data Register 0	[63:43]	0
	[42:36]	ECC [6:0] if configured with ECC for a 512KB L2 cache size, otherwise 0
	[35:12]	Physical address [39:16]
	[11]	Non-secure identifier for the
	[10:9]	Virtual index [13:12]
	[8:6]	Reserved
	[5]	Shareable
	[4]	Outer allocation hint
	[3]	L1 data cache valid
	[2:0]	L2 State
		101 Modified
		001 Exclusive
		x11 Shared
		xx0 Invalid
Data Register 1	[63:0]	0
Data Register 2	[63:0]	0

The following table shows the data that is returned from accessing the L2 data RAM.

Table 37: L2 data format

Register	Bit field	Description
Data Register 0	[63:0]	Data [63:0]
Data Register 1	[63:0]	Data [127:64]

Register	Bit field	Description
Data Register 2	[63:16]	0
	[15:8]	ECC for Data [127:64] if configured with ECC
	[7:0]	ECC for Data [63:0] if configured with ECC

The following table shows the data that is returned from accessing the L2 RAM.

Table 38: L2 format

Register	Bit field	Description
Data Register 0	[63:7]	0
	[6:0]	PLRU [6:0]
Data Register 1	[63:0]	0
Data Register 2	[63:0]	0

Encoding for the L2 TLB

The following section describes the encoding for L2 TLB direct accesses.

The following table shows the encoding that is required to select a given *TLB* entry.

Table 39: L2 encoding

Bit fields of Rd	Description
[31:24]	RAMID = 18
[23:21]	Reserved
[20:18]	Way
	000 way0
	001 way1
	010 way2
	011 way3
	100 way4
[17:8]	Reserved
[7:0]	Index

The following table shows the data that is returned from accessing the L2 .

Table 40: L2 format

Register	Bit field	Description
Instruction Register 0	[63:59]	Reserved
	[58]	Non-global
	[57]	Outer-shared

Register	Bit field	Description
Instruction Register 1	[56]	Inner-shared
	[55]	Reserved
	[54:52]	Memory attributes:
	000	Device nGnRnE
	001	Device nGnRE
	010	Device nGRE
	011	Device GRE
	100	Non-cacheable
	101	<i>Write</i> -Back No-Allocate
	110	-Back Transient
	111	-Back <i>Read-Allocate</i> and -Allocate
	[51:48]	Reserved
	[47:20]	Physical address [39:12]
	[19:17]	Page size:
	000	4KB
	001	16KB
	010	64KB
	011	256KB
	100	2MB
	101	32MB
	110	512MB
	111	1GB
	[16:7]	Reserved
	[6]	Indicates that the entry is coalesced and holds translations for four contiguous pages
	[5:2]	This bit field contains the valid bits for four contiguous pages. If the entry is non-coalesced, then 0001 indicates a valid entry.
	[1:0]	Reserved
	[63:54]	VMID [9:0]
	[53:38]	ASID [15:0]
	[37]	Walk cache entry

Register	Bit field	Description
Instruction Register 2	[36]	Prefetched translation
	[35:7]	Virtual address [48:20]
	[6]	Non-secure
	[5:0]	Reserved
	[63:8]	Reserved
	[7:6]	Translation regime: 00 Secure EL1 01 EL3 10 Non-secure EL1 11 EL2
	[5:0]	VMID [15:10]

Chapter

7

Level 2 memory system

Topics:

- [About the L2 memory system](#)
- [About the L2 cache](#)
- [Support for memory types](#)

This chapter describes the L2 memory system.

About the L2 memory system

The L2 memory subsystem consists of:

- An 8-way set associative L2 cache with a configurable size of 128KB, 256KB or 512KB. Cache lines have a fixed length of 64 bytes.
- Optional ECC protection for all RAM structures except *victim* array.
- Strictly inclusive with L1 data cache. Weakly inclusive with L1 instruction cache.
- Configurable CHI interface to the DSU or CHI compliant system with support for 128-bit and 256-bit data widths.
- Dynamic biased replacement policy.
- Modified Exclusive Shared Invalid (MESI) coherency.

About the L2 cache

The integrated L2 cache is the Point of Unification for the Cortex[®]-A76 core. It handles both instruction and data requests from the instruction side and data side of each core respectively.

When fetched from the system, instructions are allocated to the L2 cache and can be *invalidated* during maintenance operations.



Note: Caches in the *core* are d automatically at reset deassertion unless the power mode is initialized to Debug Recovery. See the *Arm[®] DynamIQ[™] Shared Unit Technical Reference Manual* for more information.

Support for memory types

The Cortex[®]-A76 core simplifies the coherency logic by downgrading some memory types.

- Memory that is marked as both Inner *Write*-Back Cacheable and Outer -Back Cacheable is cached in the L1 data cache and the L2 cache.
- Memory that is marked Inner -Through is downgraded to Non-cacheable.
- Memory that is marked Outer -Through or Outer Non-cacheable is downgraded to Non-cacheable, even if the inner attributes are -Back cacheable.

The following table shows the transaction capabilities of the Cortex[®]-A76 *core*. It lists the maximum possible values for *read*, write, DVM issuing, and snoop capabilities of the private L2 cache.

Table 41: Cortex[®]-A76 Transaction Capabilities

Attribute	Value	Description
issuing capability	22/34/46	Maximum number of outstanding write transactions. Dependent on the configured TQ size. (24/36/48)
Read issuing capability	22/34/46	Maximum number of outstanding transactions. Dependent on the configured TQ size. (24/36/48)
Snoop acceptance capability	17/23/29	Maximum number of outstanding snoops and stashes accepted. Dependent on the TQ size. (24/36/48)
DVM issuing capability	22/34/46	Maximum number of outstanding DVMOp transactions. Dependent on the configured TQ size. (24/36/48)

Chapter

8

Reliability, Availability, and Serviceability (RAS)

Topics:

- [Cache ECC and parity](#)
- [Uncorrected errors and data poisoning](#)
- [RAS error types](#)
- [Error Synchronization Barrier](#)
- [Error recording](#)
- [Error injection](#)

This chapter describes the RAS features implemented in the Cortex[®]-A76 core.

Cache ECC and parity

The Cortex®-A76 core implements the RAS extension to the Arm® v8-A architecture which provides mechanisms for standardized reporting of the errors generated by cache protection mechanisms.

When configured with *core* cache protection, the Cortex®-A76 can detect and correct a 1-bit error in any RAM and detect 2-bit errors in some RAMs.



Note: For information about SCU-L3 cache protection, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The RAS extension improves the system by reducing unplanned outages:

- Transient errors can be detected and corrected before they cause application or system failure.
- Failing components can be identified and replaced.
- Failure can be predicted ahead of time to allow replacement during planned maintenance.

Errors that are present but not detected are known as latent or undetected errors. A transaction carrying a latent error is corrupted. In a system with no error detection, all errors are latent errors and are silently propagated by components until either:

- They are masked and do not affect the outcome of the system. These are benign or false errors.
- They affect the service interface of the system and cause failure. These are silent data corruptions.

The severity of a failure can range from minor to catastrophic. In many systems, data or service loss is regarded as more of a minor failure than data corruption, as long as backup data is available.

The RAS extension focuses on errors that are produced from hardware *faults*, which fall into two main categories:

- Transient s.
- Persistent s.

The RAS extension describes data corruption s, which mostly occur in memories and on data links. RAS concepts can also be used for the management of other types of physical s found in systems, such as lock-step errors, thermal trip, and mechanical failure. The RAS extension provides a common programmers model and mechanisms for handling and error recovery.

Uncorrected errors and data poisoning

When an error is detected, the correction mechanism is triggered. However, if the error is a 2-bit error in a RAM protected by ECC, then the error is not correctable.

The behavior on an uncorrected error depends on the type of RAM.

Uncorrected error detected in a data RAM

When an uncorrected error is detected in a data RAM, the chunk of data with the error is marked as poisoned. This poison information is then transferred with the data and stored in the cache if the data is allocated into another cache. The poisoned information is stored per 64 bits of data, except in the L1 data cache where it is stored per 32 bits of data.

Uncorrected error detected in a tag RAM

When an uncorrected error is detected in a tag RAM, either the address or coherency state of the line is not known, and the corresponding data cannot be poisoned. In this case, the line is *invalidated* and an error recovery interrupt is generated to notify software that data has potentially been lost.

RAS error types

This section describes the RAS error types that are introduced by the RAS extension and supported in the Cortex®-A76 core.

When a component accesses memory, an error might be detected in that memory and then be corrected, deferred, or detected but silently propagated. The following table lists the types of RAS errors that are supported in the Cortex®-A76 *core*.

Table 42: RAS error types supported in the Cortex®-A76

RAS error type	Definition
Corrected	A <i>Corrected Error</i> (CE) is reported for a single-bit ECC error on any protected RAM.
Deferred	A <i>Deferred Error</i> (DE) is reported for a double-bit ECC error that affects the data RAM on either the L1 data cache or the L2 cache.
Uncorrected	An <i>Uncorrected Error</i> (UE) is reported for a double-bit ECC error that affects the tag RAM of either the L1 data cache or the L2 cache. An Uncorrected Error is also reported for external <i>aborts</i> received in response to a store, data cache maintenance, instruction cache maintenance, <i>TLBI</i> maintenance, or cache copyback of <i>dirty</i> data.

Error Synchronization Barrier

The *Error Synchronization Barrier* (ESB) instruction synchronizes unrecoverable system errors.

In the Cortex®-A76 *core*, the ESB instruction allows efficient isolation of errors:

- The ESB instruction does not wait for completion of accesses that cannot generate an asynchronous external *abort*. For example, if all external s are handled synchronously or it is known that no such accesses are outstanding.
- The ESB instruction does not order accesses and does not guarantee a pipeline flush.

All system errors must be synchronized by an ESB instruction, which guarantees the following:

- All system errors that are generated before the ESB instruction have pended a *System Error Interrupts* (SEI) *exception*.
- If a physical SEI is pended by or was pending before the ESB instruction executes, then:
 - It is taken before completion of the ESB instruction, if the physical SEI is unmasked at the current *Exception level*.
 - The pending SEI is cleared, the SEI status is recorded in DISR_EL1, and DISR_EL1.A is set to 1 if the physical SEI is masked at the current . It indicates that the SEI was generated before the ESB instruction by instructions that occur in program order.
- If a virtual SEI is pended by or was pending before the ESB instruction executes, then:
 - It is taken before completion of the ESB instruction, if the virtual SEI is unmasked.
 - The pending virtual SEI is cleared and the SEI status is recorded in VDISR_EL2 using the information provided by software in VESR_EL2, if the virtual SEI is masked.

After the ESB instruction, one of the following scenarios occurs:

- SEIs pended by errors are taken and their status is recorded in ESR_ELn.
- SEIs pended by errors are deferred and their status is recorded in DISR_EL1 or VDISR_EL2.

This includes unrecoverable SEIs that are generated by instructions, translation table walks, and instruction fetches on the same .



Note:

DISR_EL1 can only be accessed at EL1 and above. If EL2 is implemented and HCR_EL2.AMO is set to 1, then *reads* and writes of DISR_EL1 at Non-secure EL1 access VDISR_EL2.

See the following registers:

- [DISR_EL1, Deferred Interrupt Status Register, EL1.](#)
- [HCR_EL2, Hypervisor Configuration Register, EL2.](#)
- [VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2.](#)

Error recording

The component that detects an error is called a node. The Cortex®-A76 core is a node that interacts with the DynamIQ™ Shared Unit node. There is one record per node for the errors detected.

For more information on error recording generated by cache protection, see the *Arm® Reliability, Availability, and Serviceability (RAS) Specification, Armv8, for the Armv8-A architecture profile*. The following points apply specifically to the Cortex®-A76 *core*:

- Error recording is only available when the cache protection is implemented.
- In the Cortex®-A76 , any error that is detected is reported and recorded in the error record registers:
 - [ERRSELR_EL1, Error Record Select Register, EL1](#)
 - [ERXADDR_EL1, Selected Error Record Address Register, EL1](#)
 - [ERXCTLR_EL1, Selected Error Record Control Register, EL1](#)
 - [ERXFR_EL1, Selected Error Record Feature Register, EL1](#)
 - [ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1](#)
 - [ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1](#)
 - [ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1](#)
 - [ERXPFGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1](#)
 - [ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1](#)
 - [ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1](#)
- There are two error records provided, which can be selected with the ERRSELR_EL1 register:
 - Record 0 is private to the , and is updated on any error in the RAMs including L1 caches, *TLB*, and L2 cache.
 - Record 1 records any error in the L3 and snoop filter RAMs and is shared between all s in the *cluster*.
- The *fault* handling interrupt is generated on the nFAULTIRQ[0] pin for L3 and snoop filter errors, or on the nFAULTIRQ[n+1] pin for *n* L1 and L2 errors.

Error injection

To support testing of error handling software, the Cortex®-A76 core can inject errors in the error detection logic.

The following table describes all the possible types of error that the *core* can encounter and therefore inject.

Table 43: Errors injected in the Cortex®-A76


Error type	Description
Corrected errors	A CE is generated for a single-bit ECC error on L1 data caches and L2 caches, both on data and tag RAMs.
Deferred errors	A DE is generated for a double-bit ECC error on L1 data caches and L2 caches, but only on data RAM.

Error type	Description
Uncorrected errors	A UE is generated for a double-bit ECC error on L1 data caches and L2 caches, but only on tag RAM.

The following table describes the registers that handle error injection in the Cortex®-A76 .

Table 44: Error injection registers

Register name	Description
ERR0PFGF_EL1	The ERR Pseudo Fault Generation Feature register defines which errors can be injected.
ERR0PFGCTL_EL1	The ERR Pseudo Fault Generation Control register controls the errors that are injected.
ERR0PFGCDN_EL1	The ERR Pseudo Fault Generation Count Down register controls the <i>fault</i> injection timing.

 **Note:** This mechanism simulates the corruption of any RAM but the data is not actually corrupted.

See also:

- [ERR0PFGF, Error Pseudo Fault Generation Feature Register.](#)
- [ERR0PFGCTL, Error Pseudo Fault Generation Control Register.](#)
- [ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register.](#)

Chapter

9

Generic Interrupt Controller CPU interface

Topics:

- [About the Generic Interrupt Controller CPU interface](#)
- [Bypassing the CPU interface](#)

This chapter describes the Cortex[®]-A76 core implementation of the Arm *Generic Interrupt Controller* (GIC) CPU interface.

About the Generic Interrupt Controller CPU interface

The Cortex®-A76 core implements the GIC CPU interface as described in the Arm® *Generic Interrupt Controller Architecture Specification*.

This interfaces with an external *GICv3* or *v4* distributor component within the *cluster* system and is a resource for supporting and managing interrupts. The CPU interface *hosts* registers to mask, identify and control states of interrupts forwarded to that *core*. Each in the system has a CPU interface component and connects to a common external distributor component.



Note: This chapter describes only features that are specific to the Cortex®-A76 implementation. Additional information specific to the can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The v4 architecture supports:

- Two security states.
- Interrupt virtualization.
- *Software-generated Interrupts* (SGIs).
- Message Based Interrupts.
- System register access for the CPU interface.
- Interrupt masking and prioritization.
- Cluster environments, including systems that contain more than eight s.
- Wake-up events in power management environments.

The includes interrupt grouping functionality that supports:

- Configuring each interrupt to belong to an interrupt group.
- Signaling Group 1 interrupts to the target using either the *IRQ* or the *FIQ exception* request. Group 1 interrupts can be Secure or Non-secure.
- Signaling Group 0 interrupts to the target using the request only.
- A unified scheme for handling the priority of Group 0 and Group 1 interrupts.

This chapter describes only features that are specific to the Cortex®-A76 implementation.

Related reference

[GIC registers](#) on page 259

This chapter describes the GIC registers.

Bypassing the CPU interface

The GIC CPU Interface is always implemented within the Cortex®-A76 core.

However, you can disable it if you assert the *GICCDISABLE* signal HIGH at reset. If you disable the CPU interface, the input pins nVIRQ and nVFIQ can be driven by an external in the SoC. system register access generates *UNDEFINED* instruction *exceptions* when the CDISABLE signal is HIGH.

If the is enabled, the input pins nVIRQ and nVFIQ must be tied off to HIGH. This is because the internal CPU interface generates the virtual interrupt signals to the *cores*. The nIRQ and nFIQ signals are controlled by software, therefore there is no requirement to tie them HIGH.

Chapter 10

Advanced SIMD and floating-point support

Topics:

- [About the Advanced SIMD and floating-point support](#)
- [Accessing the feature identification registers](#)

This chapter describes the Advanced SIMD and floating-point features and registers in the Cortex[®]-A76 core. The unit in charge of handling the Advanced SIMD and floating-point features is also referred to as data engine in this manual.

About the Advanced SIMD and floating-point support

The Cortex®-A76 core supports the Advanced SIMD and scalar floating-point instructions in the A64 instruction set and the Advanced SIMD and floating-point instructions in the A32 and T32 instruction sets.

The Cortex®-A76 *floating-point* implementation:

- Does not generate *exceptions*.
- Implements all scalar operations in hardware with support for all combinations of:
 - Rounding modes.
 - Flush-to-zero.
 - Default *Not a Number* (NaN) modes.

The Arm®v8-A architecture does not define a separate version number for its *Advanced SIMD* and support in the *AArch64* execution state because the instructions are always implicitly present.

Accessing the feature identification registers

Software can identify the Advanced SIMD and floating-point features using the feature identification registers in the AArch64 Execution state only.

The Cortex®-A76 *core* only supports *AArch32* in EL0, therefore none of the feature identification registers are accessible in the Execution state.

You can access the feature identification registers in the *AArch64* Execution state using the MRS instruction, for example:

```
MRS <Xt>, ID_AA64PFR0_EL1 ; Read ID_AA64PFR0_EL1 into Xt
MRS <Xt>, MVFR0_EL1       ; Read MVFR0_EL1 into Xt
MRS <Xt>, MVFR1_EL1       ; Read MVFR1_EL1 into Xt
MRS <Xt>, MVFR2_EL1       ; Read MVFR2_EL1 into Xt
```

Table 45: *Advanced SIMD* and scalar *floating-point* feature identification registers

Register name	Description
ID_AA64PFR0_EL1	See ID_AA64PFR1_EL1, Processor Feature Register 1, EL1.
MVFR0_EL1	See MVFR0_EL1, Media and VFP Feature Register 0, EL1.
MVFR1_EL1	See MVFR1_EL1, Media and Feature Register 1, EL1.
MVFR2_EL1	See MVFR2_EL1, Media and Feature Register 2, EL1.

Part II

Register descriptions

Topics:

- [AArch32 system registers](#)
- [AArch64 system registers](#)
- [Error system registers](#)
- [GIC registers](#)
- [Advanced SIMD and floating-point registers](#)

This part describes the non-debug registers of the Cortex[®]-A76 core.

Chapter 11

AArch32 system registers

Topics:

- [AArch32 architectural system register summary](#)

This chapter describes the system registers in the AArch32 state.

AArch32 architectural system register summary

This chapter identifies the AArch32 architectural system registers implemented in the Cortex®-A76 core.

The following table identifies the architecturally defined registers that are implemented in the Cortex®-A76 *core*. For a description of these registers see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

For the registers listed in the following table, coproc==1111.

Table 46: Architecturally defined registers

Name	CRn	Opc1	CRm	Opc2	Width	description
CNTFRQ	c14	0	c0	0	32	Timer Clock Ticks per Second
CNTP_CTL	c14	0	c2	1	32	Counter- timer Physical Timer Control register
CNTP_CVAL	-	2	c14	-	64	Counter-timer Physical Timer CompareValue register
CNTP_TVAL	c14	0	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTPCT	-	0	c14	-	64	Counter-timer Physical Count register
CNTV_CTL	c14	0	c3	1	32	Counter- timer Virtual Timer Control register
CNTV_CVAL	-	3	c14	-	64	Counter-timer Virtual Timer CompareValue register
CNTV_TVAL	c14	0	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTVCT	-	1	c14	-	64	Counter-timer Virtual Count register
CP15ISB	c7	0	c5	4	32	Instruction Synchronization Barrier System instruction

Name	CRn	Opc1	CRm	Opc2	Width	description
CP15DSB	c7	0	c10	4	32	Data Synchronization Barrier System instruction
CP15DMB	c7	0	c10	5	32	Data Memory Barrier System instruction
DLR	c4	3	c5	1	32	Debug Link Register
DSPSR	c4	3	c5	0	32	Debug Saved <i>Program Status Register</i>
TPIDRURO	c13	0	c0	3	32	User Read Only Thread ID Register
TPIDRURW	c13	0	c0	2	32	User Read/Write Thread ID Register

Chapter

12

AArch64 system registers

Topics:

- [AArch64 registers](#)
- [AArch64 architectural system register summary](#)
- [AArch64 implementation defined register summary](#)
- [AArch64 registers by functional group](#)
- [ACTLR_EL1, Auxiliary Control Register, EL1](#)
- [ACTLR_EL2, Auxiliary Control Register, EL2](#)
- [ACTLR_EL3, Auxiliary Control Register, EL3](#)
- [AFSR0_EL1, Auxiliary Fault Status Register 0, EL1](#)
- [AFSR0_EL2, Auxiliary Fault Status Register 0, EL2](#)
- [AFSR0_EL3, Auxiliary Fault Status Register 0, EL3](#)
- [AFSR1_EL1, Auxiliary Fault Status Register 1, EL1](#)
- [AFSR1_EL2, Auxiliary Fault Status Register 1, EL2](#)
- [AFSR1_EL3, Auxiliary Fault Status Register 1, EL3](#)
- [AIDR_EL1, Auxiliary ID Register, EL1](#)
- [AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1](#)
- [AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2](#)
- [AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3](#)
-
-

This chapter describes the system registers in the AArch64 state.

-
-
-
- CLIDR_EL1, Cache Level ID Register, EL1
- CPACR_EL1, Architectural Feature Access Control Register, EL1
- CPTR_EL2, Architectural Feature Trap Register, EL2
- CPTR_EL3, Architectural Feature Trap Register, EL3
- CPUACTLR_EL1, CPU Auxiliary Control Register, EL1
- CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1
- CPUACTLR3_EL1, CPU Auxiliary Control Register 3, EL1
- CPUCFR_EL1, CPU Configuration Register, EL1
- CPUECTLR_EL1, CPU Extended Control Register, EL1
- CPUPCR_EL3, CPU Private Control Register, EL3
- CPUPMR_EL3, CPU Private Mask Register, EL3
- CPUPOR_EL3, CPU Private Operation Register, EL3
- CPUPSELR_EL3, CPU Private Selection Register, EL3
- CPUPWRCTLR_EL1, Power Control Register, EL1
- CSSELR_EL1, Cache Size Selection Register, EL1
- DISR_EL1, Deferred Interrupt Status Register, EL1
- ERRIDR_EL1, Error ID Register, EL1
- ERRSELR_EL1, Error Record Select Register, EL1
- ERXADDR_EL1, Selected Error Record Address Register, EL1
- ERXCTLR_EL1, Selected Error Record Control Register, EL1
- ERXFR_EL1, Selected Error Record Feature Register, EL1
- ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1

- ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1
- ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1
- ERXPFGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1
- ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1
- ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1
- ESR_EL1, Exception Syndrome Register, EL1
- ESR_EL2, Exception Syndrome Register, EL2
- ESR_EL3, Exception Syndrome Register, EL3
- HACR_EL2, Hyp Auxiliary Configuration Register, EL2
- HCR_EL2, Hypervisor Configuration Register, EL2
- ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0
- ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1
- ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1
- ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1, EL1
- ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1
- ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1
- ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1
- ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1
- ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1
- ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1

- ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1, EL1
- ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1
- ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1
- ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1
- ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1
- ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1
- ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1
- ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1
- ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1
- ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1
- ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1
- ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1
- ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1
- ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1
- ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1
- ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1
- ID_PFR2_EL1, AArch32 Processor Feature Register 2, EL1

- LORC_EL1, LORegion Control Register, EL1
- LORID_EL1, LORegion ID Register, EL1
- LORN_EL1, LORegion Number Register, EL1
- MDCR_EL3, Monitor Debug Configuration Register, EL3
- MIDR_EL1, Main ID Register, EL1
- MPIDR_EL1, Multiprocessor Affinity Register, EL1
- PAR_EL1, Physical Address Register, EL1
- REVIDR_EL1, Revision ID Register, EL1
- RMR_EL3, Reset Management Register
- RVBAR_EL3, Reset Vector Base Address Register, EL3
- SCTLR_EL1, System Control Register, EL1
- SCTLR_EL3, System Control Register, EL3
- TCR_EL1, Translation Control Register, EL1
- TCR_EL2, Translation Control Register, EL2
- TCR_EL3, Translation Control Register, EL3
- TTBR1_EL1, Translation Table Base Register 1, EL1
- TTBR1_EL2, Translation Table Base Register 1, EL2
- VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2
- VESR_EL2, Virtual SErrors Exception Syndrome Register
- VTCR_EL2, Virtualization Translation Control Register, EL2
- VTTBR_EL2, Virtualization Translation Table Base Register, EL2

AArch64 registers

This chapter provides information about the AArch64 system registers with *implementation defined* bit fields and *implementation defined* registers associated with the core.

The chapter provides *implementation specific* information, for a complete description of the registers, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The chapter is presented as follows:

AArch64 architectural system register summary

This section identifies the architectural system registers implemented in the Cortex®-A76 *core* that have *implementation defined* bit fields. The register descriptions for these registers only contain information about the *implementation defined* bits.

***implementation defined* register summary**

This section identifies the architectural registers implemented in the Cortex®-A76 that are *implementation defined*.

registers by functional group

This section groups the *implementation defined* registers and architectural system registers with *implementation defined* bit fields, as identified previously, by function. It also provides reset details for key register types.

Register descriptions

The remainder of the chapter provides register descriptions of the *implementation defined* registers and architectural system registers with *implementation defined* bit fields, as identified previously. These are listed in alphabetic order.

AArch64 architectural system register summary

This section describes the AArch64 architectural system registers implemented in the Cortex®-A76 core.

The section contains two tables:

Registers with implementation defined bit fields

This table identifies the architecturally defined registers in Cortex®-A76 that have implementation defined bit fields. The register descriptions for these registers only contain information about the implementation defined bits.

See [Table 1](#).

Other architecturally defined registers

This table identifies the other architecturally defined registers that are implemented in the Cortex®-A76 *core*. These registers are described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

See [Table 48: Other architecturally defined registers](#) on page 112.

Table 47: Registers with implementation defined bit fields

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ACTLR_EL1	3	c1	0	c0	1	64	ACTLR_EL1, Auxiliary Control Register, EL1
ACTLR_EL2	3	c1	4	c0	1	64	ACTLR_EL2, Auxiliary Control Register, EL2
ACTLR_EL3	3	c1	6	c0	1	64	ACTLR_EL3, Auxiliary Control Register, EL3
AIDR_EL1	3	c0	1	c0	7	32	AIDR_EL1, Auxiliary ID Register, EL1
AFSR0_EL1	3	c5	0	c1	0	32	AFSR0_EL1, Auxiliary Fault Status Register 0, EL1
AFSR0_EL2	3	c5	4	c1	0	32	AFSR0_EL2, Auxiliary Fault Status Register 0, EL2
AFSR0_EL3	3	c5	6	c1	0	32	AFSR0_EL3, Auxiliary Fault Status Register 0, EL3
AFSR1_EL1	3	c5	0	c1	1	32	AFSR1_EL1, Auxiliary Fault Status Register 1, EL1
AFSR1_EL2	3	c5	4	c1	1	32	AFSR1_EL2, Auxiliary Fault Status Register 1, EL2

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
AFSR1_EL3	3	c5	6	c1	1	32	AFSR1_EL3, Auxiliary Fault Status Register 1, EL3
AMAIR_EL1	3	c10	0	c3	0	64	AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1
AMAIR_EL2	3	c10	4	c3	0	64	AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2
AMAIR_EL3	3	c10	6	c3	0	64	AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3
CCSIDR_EL1	3	c0	1	c0	0	32	CCSIDR_EL1, Cache Size ID Register, EL1
CLIDR_EL1	3	c0	1	c0	1	64	CLIDR_EL1, Cache Level ID Register, EL1
CPACR_EL1	3	c1	0	c0	2	32	CPACR_EL1, Architectural Feature Access Control Register, EL1
CPTR_EL2	3	c1	4	c1	2	32	CPTR_EL2, Architectural Feature Trap Register, EL2

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CPTR_EL3	3	c1	6	c1	2	32	CPTR_EL3, Architectural Feature Trap Register, EL3
CSSELR_EL13		c0	2	c0	0	32	CSSELR_EL1, Cache Size Selection Register, EL1
CTR_EL0	3	c0	3	c0	1	32	CTR_EL0, Cache Type Register, EL0
DISR_EL1	3	c12	0	c1	1	64	DISR_EL1, Deferred Interrupt Status Register, EL1
ERRIDR_EL13		c5	0	c3	0	32	ERRIDR_EL1, Error ID Register, EL1
ERRSELR_EL13		c5	0	c3	1	32	ERRSELR_EL1, Error Record Select Register, EL1
ERXADDR_EL1		c5	0	c4	3	64	ERXADDR_EL1, Selected Error Record Address Register, EL1
ERXCTLR_EL1		c5	0	c4	1	64	ERXCTLR_EL1, Selected Error Record Control Register, EL1
ERXFR_EL1	3	c5	0	c4	0	64	ERXFR_EL1, Selected Error Record Feature Register, EL1

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ERXMISC0_EL1		c5	0	c5	0	64	ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1
ERXMISC1_EL1		c5	0	c5	1	64	ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1
ERXSTATUS_EL1		c5	0	c4	2	32	ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1
ESR_EL1	3	c5	0	c2	0	32	ESR_EL1, Exception Syndrome Register, EL1
ESR_EL2	3	c5	4	c2	0	32	ESR_EL2, Exception Syndrome Register, EL2
ESR_EL3	3	c5	6	c2	0	32	ESR_EL3, Exception Syndrome Register, EL3
HACR_EL2	3	c1	4	c1	7	32	HACR_EL2, Hyp Auxiliary Configuration Register, EL2
HCR_EL2	3	c1	4	c1	0	64	HCR_EL2, Hypervisor Configuration Register, EL2

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ID_AFR0_EL1	B	c0	0	c1	3	32	ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1
ID_DFR0_EL1	B	c0	0	c1	2	32	ID_DFR0_EL1, Debug Feature Register 0, EL1
ID_ISAR0_EL1	B	c0	0	c2	0	32	ID_ISAR0_EL1, Instruction Set Attribute Register 0, EL1
ID_ISAR1_EL1	B	c0	0	c2	1	32	ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1
ID_ISAR2_EL1	B	c0	0	c2	2	32	ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1
ID_ISAR3_EL1	B	c0	0	c2	3	32	ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1
ID_ISAR4_EL1	B	c0	0	c2	4	32	ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1
ID_ISAR5_EL1	B	c0	0	c2	5	32	ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1
ID_ISAR6_EL1	B	c0	0	c2	7	32	ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ID_MMFR0_EL1		c0	0	c1	4	32	ID_MMFR0_EL1, Memory Model Feature Register 0, EL1
ID_MMFR1_EL1		c0	0	c1	5	32	ID_MMFR1_EL1, Memory Model Feature Register 1, EL1
ID_MMFR2_EL1		c0	0	c1	6	32	ID_MMFR2_EL1, Memory Model Feature Register 2, EL1
ID_MMFR3_EL1		c0	0	c1	7	32	ID_MMFR3_EL1, Memory Model Feature Register 3, EL1
ID_MMFR4_EL1		c0	0	c2	6	32	ID_MMFR4_EL1, Memory Model Feature Register 4, EL1
ID_PFR0_EL13		c0	0	c1	0	32	ID_PFR0_EL1, Processor Feature Register 0, EL1
ID_PFR1_EL13		c0	0	c1	1	32	ID_PFR1_EL1, Processor Feature Register 1, EL1
ID_PFR2_EL13		c0	0	c3	4	32	ID_PFR2_EL1, Processor Feature Register 2, EL1

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ID_AA64DFR0_EL1	0	c0	0	c5	0	64	ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1
ID_AA64ISAR0_EL1	0	c0	0	c6	0	64	ID_AA64ISAR0_EL1, Instruction Set Attribute Register 0, EL1
ID_AA64ISAR1_EL1	0	c0	0	c6	1	64	ID_AA64ISAR1_EL1, Instruction Set Attribute Register 1, EL1
ID_AA64MMBR0_EL1	0	c0	0	c7	0	64	ID_AA64MMFR0_EL1, Memory Model Feature Register 0, EL1
ID_AA64MMBR1_EL1	0	c0	0	c7	1	64	ID_AA64MMFR1_EL1, Memory Model Feature Register 1, EL1
ID_AA64MMBR2_EL1	0	c0	0	c7	2	64	ID_AA64MMFR2_EL1, Memory Model Feature Register 2, EL1
ID_AA64PFR0_EL1	0	c0	0	c4	0	64	ID_AA64PFR0_EL1, Processor Feature Register 0, EL1
IFSR32_EL2 3		c5	4	c0	1	32	
LORC_EL1 3		c10	0	c4	3	64	LORC_EL1, LORegion Control Register, EL1
LORID_EL1 3		c10	0	c4	7	64	LORID_EL1, LORegion ID Register, EL1

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
LORN_EL1	3	c10	0	c4	2	64	LORN_EL1, LORegion Number Register, EL1
MDCR_EL3	3	c1	6	c3	1	32	MDCR_EL3, Monitor Debug Configuration Register, EL3
MIDR_EL1	3	c0	0	c0	0	32	MIDR_EL1, Main ID Register, EL1
MPIDR_EL1	3	c0	0	c0	5	64	MPIDR_EL1, Multiprocessor Affinity Register, EL1
R_EL1	3	c7	0	c4	0	64	R_EL1, Physical Address Register, EL1
RVBAR_EL3	3	c12	6	c0	1	64	RVBAR_EL3, Reset Vector Base Address Register, EL3
REVIDR_EL1	3	c0	0	c0	6	32	REVIDR_EL1, Revision ID Register, EL1
SCTLR_EL1	3	c1	0	c0	0	32	SCTLR_EL1, System Control Register, EL1
SCTLR_EL2	3	c1	4	c0	0	32	SCTLR_EL2, System Control Register, EL2

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
SCTLR_EL1	3	c1	5	c0	0	32	SCTLR_EL1, System Control Register, EL1
SCTLR_EL3	3	c1	6	c0	0	32	SCTLR_EL3, System Control Register, EL3
TCR_EL1	3	c2	0	c0	2	64	TCR_EL1, Translation Control Register, EL1
TCR_EL2	3	c2	4	c0	2	64	TCR_EL2, Translation Control Register, EL2
TCR_EL3	3	c2	6	c0	2	64	TCR_EL3, Translation Control Register, EL3
TTBR0_EL1	3	c2	0	c0	0	64	TTBR0_EL1, Translation Table Base Register 0, EL1
TTBR0_EL2	3	c2	4	c0	0	64	TTBR0_EL2, Translation Table Base Register 0, EL2
TTBR0_EL3	3	c2	6	c0	0	64	TTBR0_EL3, Translation Table Base Register 0, EL3
TTBR1_EL1	3	c2	0	c0	1	64	TTBR1_EL1, Translation Table Base Register 1, EL1

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
TTBR1_EL2	3	c2	4	c0	1	64	TTBR1_EL2, Translation Table Base Register 1, EL2
VDISR_EL2	3	c12	4	c1	1	64	VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2
VSESR_EL2	3	c5	4	c2	3	64	VSESR_EL2, Virtual SError Exception Syndrome Register (Ares Specific)
VTCR_EL2	3	c2	4	c1	2	32	VTCR_EL2, Virtualization Translation Control Register, EL2
VTTBR_EL2	3	c2	4	c1	0	64	VTTBR_EL2, Virtualization Translation Table Base Register, EL2

Table 48: Other architecturally defined registers

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
AFSR0_EL123		c5	5	1	0	32	Auxiliary Fault Status Register 0
AFSR1_EL123		c5	5	1	1	32	Auxiliary Fault Status Register 1
AMAIR_EL123		c10	5	c3	0	64	Auxiliary Memory Attribute Indirection Register

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CNTFRQ_EL0		c14	3	0	0	32	Counter-timer Frequency register
CNTHCTL_EB2		c14	4	c1	0	32	Counter-timer Hypervisor Control register
CNTHP_CTL3EL2		c14	4	c2	1	32	Counter-timer Hypervisor Physical Timer Control register
CNTHP_CVAB_EL2		c14	4	c2	2	64	Counter-timer Hyp Physical CompareValue register
CNTHP_TVAB_EL2		c14	4	c2	0	32	Counter-timer Hyp Physical Timer TimerValue register
CNTHV_CTL3EL2		c14	4	c3	1	32	Counter-timer Virtual Timer Control register
CNTHV_CVAB_EL2		c14	4	c3	2	64	Counter-timer Virtual Timer CompareValue register
CNTHV_TVAB_EL2		c14	4	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTKCTL_EB1		c14	0	c1	0	32	Counter-timer Kernel Control register

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CNTKCTL_EB12		c14	5	c1	0	32	Counter-timer Kernel Control register
CNTP_CTL_EL0		c14	3	c2	1	32	Counter-timer Physical Timer Control register
CNTP_CTL_EL02		c14	5	c2	1	32	Counter-timer Physical Timer Control register
CNTP_CVAL_EL0		c14	3	c2	2	64	Counter-timer Physical Timer CompareValue register
CNTP_CVAL_EL02		c14	5	c2	2	64	Counter-timer Physical Timer CompareValue register
CNTP_TVAL_EL0		c14	3	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTP_TVAL_EL02		c14	5	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTPCT_EL0B		c14	3	c0	1	64	Counter-timer Physical Count register

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CNTPS_CTL_EL1		c14	7	c2	1	32	Counter-timer Physical Secure Timer Control register
CNTPS_CVAL3_EL1		c14	7	c2	2	64	Counter-timer Physical Secure Timer CompareValue register
CNTPS_TVAL3_EL1		c14	7	c2	0	32	Counter-timer Physical Secure Timer TimerValue register
CNTV_CTL_EL0		c14	3	c3	1	32	Counter-timer Virtual Timer Control register
CNTV_CTL_EL02		c14	5	c3	1	32	Counter-timer Virtual Timer Control register
CNTV_CVAL3_EL0		c14	3	c3	2	64	Counter-timer Virtual Timer CompareValue register
CNTV_CVAL3_EL02		c14	5	c3	2	64	Counter-timer Virtual Timer CompareValue register
CNTV_TVAL3_EL0		c14	3	c3	0	32	Counter-timer Virtual Timer TimerValue register

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CNTV_TVAL_3EL02		c14	5	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTVCT_EL0		c14	3	c0	2	64	Counter-timer Virtual Count register
CNTVOFF_EB2		c14	4	c0	3	64	Counter-timer Virtual Offset register
CONTEXTIDR_EL1		c13	0	c0	1	32	Context ID Register (EL1)
CONTEXTIDR_EL12		c13	5	c0	1	32	Context ID Register (EL12)
CONTEXTIDR_EL2		c13	4	c0	1	32	Context ID Register (EL2)
CCR_EL12 3		c1	5	c0	2	32	Architectural Feature Access Control Register
CPTR_EL3 3		c1	6	c1	2	32	Architectural Feature Trap Register (EL3)
DACR32_EL23		c3	4	c0	0	32	Domain Access Control Register
ESR_EL12 3		c5	5	c2	0	32	Exception Syndrome Register (EL12)
FAR_EL1 3		c6	0	c0	0	64	Fault Address Register (EL1)
FAR_EL12 3		c6	5	c0	0	64	Fault Address Register (EL12)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
FAR_EL2	3	c6	4	c0	0	64	Fault Address Register (EL2)
FAR_EL3	3	c6	6	c0	0	64	Fault Address Register (EL3)
FXC32_EL2	3	c5	4	c3	0	32	Floating-point Exception Control register
HPFAR_EL2	3	c6	4	c0	4	64	Hypervisor Fault Address Register
HSTR_EL2	3	c1	4	c1	3	32	Hypervisor System Trap Register
ID_AA64AFR0_EL1		c0	0	c5	4	64	Auxiliary Feature Register 0
ID_AA64AFR3_EL1		c0	0	c5	5	64	Auxiliary Feature Register 1
ID_AA64DFR3_EL1		c0	0	c5	1	64	Debug Feature Register 1
ID_AA64PFR3_EL1		c0	0	c4	1	64	Core Feature Register 1
ISR_EL1	3	c12	0	c1	0	32	Interrupt Status Register
LOREA_EL1	3	c10	0	c4	1	64	LORegion End Address Register
LORSA_EL1	3	c10	0	c4	0	64	LORegion Start Address Register
MAIR_EL1	3	c10	0	c2	0	64	Memory Attribute Indirection Register (EL1)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
MAIR_EL12	3	c10	5	c2	0	64	Memory Attribute Indirection Register (EL12)
MAIR_EL2	3	c10	4	c2	0	64	Memory Attribute Indirection Register (EL2)
MAIR_EL3	3	c10	6	c2	0	64	Memory Attribute Indirection Register (EL3)
MDCR_EL2	3	c1	4	c1	1	32	Monitor Debug Configuration Register
MVFR0_EL1	3	c0	0	c3	0	32	Media and <i>VFP</i> Feature Register 0
MVFR1_EL1	3	c0	0	c3	1	32	Media and Feature Register 1
MVFR2_EL1	3	c0	0	c3	2	32	Media and Feature Register 2
RMR_EL3	3	c12	6	c0	2	32	Reset Management Register
SCR_EL3	3	c1	6	c1	0	32	Secure Configuration Register
SDER32_EL3	3	c1	6	c1	1	32	Secure Debug Enable Register
TCR_EL12	3	c2	5	c0	2	64	Translation Control Register (EL12)
TPIDR_EL0	3	c13	3	c0	2	64	EL0 Read/Write Software Thread ID Register

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
TPIDR_EL1	3	c13	0	c0	4	64	EL1 Software Thread ID Register
TPIDR_EL2	3	c13	4	c0	2	64	EL2 Software Thread ID Register
TPIDR_EL3	3	c13	6	c0	2	64	EL3 Software Thread ID Register
TPIDRRO_EL0	3	c13	3	c0	3	64	EL0 Read-Only Software Thread ID Register
TTBR0_EL123	3	c2	5	c0	0	64	Translation Table Base Register 0 (EL12)
TTBR1_EL123	3	c2	5	c0	1	64	Translation Table Base Register 1 (EL12)
VBAR_EL1	3	c12	0	c0	0	64	Vector Base Address Register (EL1)
VBAR_EL12	3	c12	5	c0	0	64	Vector Base Address Register (EL12)
VBAR_EL2	3	c12	4	c0	0	64	Vector Base Address Register (EL2)
VBAR_EL3	3	c12	6	c0	0	64	Vector Base Address Register (EL3)
VMPIDR_EL2	3	c0	4	c0	5	64	Virtualization Multiprocessor ID Register
VPIDR_EL2	3	c0	4	c0	0	32	Virtualization Core ID Register

AArch64 implementation defined register summary

This section describes the AArch64 registers in the Cortex[®]-A76 core that are implementation defined.

The following tables lists the AArch 64 implementation defined registers, sorted by opcode.

Table 49: AArch64 implementation defined registers

Name	Copro	CRn	Op1	CRm	Op2	Width	Description
ATCR_EL1	3	c15	0	c7	0	32	on page 148
ATCR_EL2	3	c15	4	c7	0	32	on page 149
ATCR_EL12	3	c15	5	c7	0	32	on page 149
ATCR_EL3	3	c15	6	c7	0	32	on page 149
AVTCR_EL2	3	c15	4	c7	1	32	on page 149
CPUACTLR_BL1		c15	0	c1	0	64	CPUACTLR_EL1, CPU Auxiliary Control Register, EL1
CPUACTLR2_EL1		c15	0	c1	1	64	CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1
CPUACTLR3_EL1		c15	0	c1	2	64	CPUACTLR3_EL1, CPU Auxiliary Control Register 3, EL1
CPUCFR_EL3		c15	0	c0	0	32	CPUCFR_EL1, CPU Configuration Register, EL1
CPUECTLR_BL1		c15	0	c1	4	64	CPUECTLR_EL1, CPU Extended Control Register, EL1

Name	Copro	CRn	Op1	CRm	Op2	Width	Description
CPUPCR_EL3		c15	6	c8	1	64	CPUPCR_EL3, CPU Private Control Register, EL3
CPUPMR_EL3		c15	6	c8	3	64	CPUPMR_EL3, CPU Private Mask Register, EL3
CPUPOR_EL3		c15	6	c8	2	64	CPUPOR_EL3, CPU Private Operation Register, EL3
CPUPSELR_EL3		c15	6	c8	0	32	CPUPSELR_EL3, CPU Private Selection Register, EL3
CPUPWRCTLR_EL1		c15	0	c2	7	32	CPUPWRCTLR_EL1, Power Control Register, EL1
ERXPFPGCDNR_EL1		c15	0	c2	2	32	ERXPFPGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1
ERXPFPGCTLR_EL1		c15	0	c2	1	32	ERXPFPGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1
ERXPFGFR_EL1		c15	0	c2	0	32	ERXPFGFR_EL1, Selected Error Pseudo Fault Generation Feature Register, EL1

The following table shows the 32-bit wide implementation defined Cluster registers. Details of these registers can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table 50: Cluster registers

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERCFR_EL1		c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR_EL1		c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR_EL1		c15	0	c3	2	32-bit	Cluster ECO ID.
CLUSTERACTLR_EL1		c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR_EL1		c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCtrlr_EL1		c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWrdn_EL1		c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWStat_EL1		c15	0	c3	7	32-bit	Cluster power status register.
CLUSTERTHREADSID_EL1		c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID_EL1		c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID_EL1		c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERPARTCR_EL1		c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS_EL1		c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT_EL1		c15	0	c4	5	32-bit	Cluster L3 hit counter register.

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERL3MISS_EL1		c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERTHREADSIDOVR_EL1		c15	0	c4	7	32-bit	Cluster thread scheme ID override register
CLUSTERPMCR_EL1		c15	0	c5	0	32-bit	Cluster Performance Monitors Control Register
CLUSTERPMCNENSET_EL1		c15	0	c5	1	32-bit	Cluster Count Enable Set Register
CLUSTERPMCNENCLR_EL1		c15	0	c5	2	32-bit	Cluster Count Enable Clear Register
CLUSTERPMOVSET_EL1		c15	0	c5	3	32-bit	Cluster Overflow Flag Status Set
CLUSTERPMOVCLR_EL1		c15	0	c5	4	32-bit	Cluster Overflow Flag Status Clear
CLUSTERPMSELR_EL1		c15	0	c5	5	32-bit	Cluster <i>Event</i> Counter Selection Register
CLUSTERPMINENSET_EL1		c15	0	c5	6	32-bit	Cluster Interrupt Enable Set Register
CLUSTERPMINENCLR_EL1		c15	0	c5	7	32-bit	Cluster Interrupt Enable Clear Register
CLUSTERPMXEVTYPER_EL1		c15	0	c6	1	32-bit	Cluster Selected Type and Filter Register

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERPMXEVCNTR_EL1	3	c15	0	c6	2	32-bit	Cluster Selected Counter Register
Reserved/ RAZ	3	c15	0	c6	3	32-bit	Cluster Monitor Debug Configuration Register
CLUSTERPMCEID0_EL1	3	c15	0	c6	4	32-bit	Cluster Common Identification ID0 Register
CLUSTERPMCEID1_EL1	3	c15	0	c6	5	32-bit	Cluster Common Identification ID1 Register
CLUSTERPMCLAIMSET_EL1	3	c15	0	c6	6	32-bit	Cluster Performance Monitor Claim Tag Set Register
CLUSTERPMCLAIMCLR_EL1	3	c15	0	c6	7	32-bit	Cluster Performance Monitor Claim Tag Clear Register

AArch64 registers by functional group

This section identifies the AArch64 registers by their functional groups and applies to the registers in the core that are implementation defined or have micro-architectural bit fields. Reset values are provided for these registers.

Identification registers

Name	Type	Reset	Description
AIDR_EL1	RO	00000000	AIDR_EL1 , Auxiliary ID Register, EL1
CCSIDR_EL1	RO	-	CCSIDR_EL1 , Cache Size ID Register, EL1
CLIDR_EL1	RO	<ul style="list-style-type: none"> C3000123 if L3 cache present. 82000023 if no L3 cache. 	CLIDR_EL1 , Cache Level ID Register, EL1

Name	Type	Reset	Description
CSSELR_EL1	RW	UNK	CSSELR_EL1, Cache Size Selection Register, EL1
CTR_EL0	RO	8444C004	CTR_EL0, Cache Type Register, EL0
DCZID_EL0	RO	00000004	DCZID_EL0, Data Cache Zero ID Register, EL0
ERRIDR_EL1	RO	00000002	ERRIDR_EL1, Error ID Register, EL1
ID_AA64AFR0_EL1	RO	00000000	ID_AA64AFR0_EL1, <i>AArch64</i> Auxiliary Feature Register 0
ID_AA64AFR1_EL1	RO	00000000	ID_AA64AFR1_EL1, Auxiliary Feature Register 1
ID_AA64DFR0_EL1	RO	0000000010305408	ID_AA64DFR0_EL1, Debug Feature Register 0, EL1
ID_AA64DFR1_EL1	RO	00000000	ID_AA64DFR1_EL1, Debug Feature Register 1, EL1
ID_AA64ISAR0_EL1	RO	<ul style="list-style-type: none"> 0000100010211120 if the Cryptographic Extension is implemented. 0000100010210000 if the Cryptographic Extension is not implemented. 	ID_AA64ISAR0_EL1, Instruction Set Attribute Register 0, EL1
ID_AA64ISAR1_EL1	RO	0000000000100001	ID_AA64ISAR1_EL1, Instruction Set Attribute Register 1, EL1
ID_AA64MMFR0_EL1	RO	0000000000101122	ID_AA64MMFR0_EL1, Memory Model Feature Register 0, EL1
ID_AA64MMFR1_EL1	RO	0000000010212122	ID_AA64MMFR1_EL1, Memory Model Feature Register 1, EL1
ID_AA64MMFR2_EL1	RO	0000000000001011	ID_AA64MMFR2_EL1, Memory Model Feature Register 2, EL1

Name	Type	Reset	Description
ID_AA64PFR0_EL1	RO	<ul style="list-style-type: none"> 1100000010111112 if the <i>GIC</i>v4 interface is disabled. 1100000011111112 if the v4 interface is enabled. 	ID_AA64PFR0_EL1, Processor Feature Register 0, EL1
ID_AA64PFR1_EL1	RO	0000000000000010	ID_AA64PFR1_EL1, Processor Feature Register 1, EL1
ID_AFR0_EL1	RO	00000000	ID_AFR0_EL1, <i>AArch32</i> Auxiliary Feature Register 0, EL1
ID_DFR0_EL1	RO	04010088	ID_DFR0_EL1, Debug Feature Register 0, EL1
ID_ISAR0_EL1	RO	02101110	ID_ISAR0_EL1, Instruction Set Attribute Register 0, EL1
ID_ISAR1_EL1	RO	13112111	ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1
ID_ISAR2_EL1	RO	21232042	ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1
ID_ISAR3_EL1	RO	01112131	ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1
ID_ISAR4_EL1	RO	00010142	ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1
ID_ISAR5_EL1	RO	01011121 ID_ISAR5 has the value 01010001 if the Cryptographic Extension is not implemented and enabled.	ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1
ID_ISAR6_EL1	RO	00000010	ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1
ID_MMFR0_EL1	RO	10201105	ID_MMFR0_EL1, Memory Model Feature Register 0, EL1

Name	Type	Reset	Description
ID_MMFR1_EL1	RO	40000000	ID_MMFR1_EL1, Memory Model Feature Register 1, EL1
ID_MMFR2_EL1	RO	01260000	ID_MMFR2_EL1, Memory Model Feature Register 2, EL1
ID_MMFR3_EL1	RO	02122211	ID_MMFR3_EL1, Memory Model Feature Register 3, EL1
ID_MMFR4_EL1	RO	00021110	ID_MMFR4_EL1, Memory Model Feature Register 4, EL1
ID_PFR0_EL1	RO	10010131	ID_PFR0_EL1, Processor Feature Register 0, EL1
ID_PFR1_EL1	RO	10010000 Bits [31:28] are 1 if the CPU interface is implemented and enabled, and 0 otherwise.	ID_PFR1_EL1, Processor Feature Register 1, EL1
ID_PFR2_EL1	RO	00000011	ID_PFR2_EL1, Processor Feature Register 2, EL1
LORID_EL1	RO	0000000000040004	LORID_EL1, LORegion ID Register, EL1
MIDR_EL1	RO	413FD0B1	MIDR_EL1, Main ID Register, EL1
MPIDR_EL1	RO	The reset value depends on CLUSTERIDAFF2[7:0] and CLUSTERIDAFF3[7:0]. See register description for details.	MPIDR_EL1, Multiprocessor Affinity Register, EL1
REVIDR_EL1	RO	00000000	REVIDR_EL1, Revision ID Register, EL1
VMPIDR_EL2	RW	The reset value is the value of MPIDR_EL1.	Virtualization Multiprocessor ID Register EL2
VPIDR_EL2	RW	The reset value is the value of MIDR_EL1.	Virtualization Core ID Register EL2

Other system control registers

Name	Type	Description
ACTLR_EL1	RW	ACTLR_EL1, Auxiliary Control Register, EL1
ACTLR_EL2	RW	ACTLR_EL2, Auxiliary Control Register, EL2
ACTLR_EL3	RW	ACTLR_EL3, Auxiliary Control Register, EL3
CPACR_EL1	RW	CPACR_EL1, Architectural Feature Access Control Register, EL1
SCTLR_EL1	RW	SCTLR_EL1, System Control Register, EL1
SCTLR_EL2	RW	SCTLR_EL2, System Control Register, EL2
SCTLR_EL3	RW	SCTLR_EL3, System Control Register, EL3
SCTLR_EL12	RW	SCTLR_EL1, System Control Register, EL1

Reliability, Availability, Serviceability (RAS) registers

Name	Type	Description
DISR_EL1	RW	DISR_EL1, Deferred Interrupt Status Register, EL1
ERRIDR_EL1	RW	ERRIDR_EL1, Error ID Register, EL1
ERRSELR_EL1	RW	ERRSELR_EL1, Error Record Select Register, EL1
ERXADDR_EL1	RW	ERXADDR_EL1, Selected Error Record Address Register, EL1
ERXCTLR_EL1	RW	ERXCTLR_EL1, Selected Error Record Control Register, EL1
ERXFR_EL1	RO	ERXFR_EL1, Selected Error Record Feature Register, EL1
ERXMISC0_EL1	RW	ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1

Name	Type	Description
ERXMISC1_EL1	RW	ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1
ERXSTATUS_EL1	RW	ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1
ERXPFGCDNR_EL1	RW	ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1
ERXPFCTLR_EL1	RW	ERXPFGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1
ERXPFGFR_EL1	RO	ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1
HCR_EL2	RW	HCR_EL2, Hypervisor Configuration Register, EL2
VDISR_EL2	RW	VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2
VSESR_EL2	RW	VSESR_EL2, Virtual SError Exception Syndrome Register (Ares Specific)

Virtual Memory control registers

Name	Type	Description
AMAIR_EL1	RW	AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1
AMAIR_EL2	RW	AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2
AMAIR_EL3	RW	AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3
ATCR_EL1	RW	on page 148
ATCR_EL2	RW	on page 149
ATCR_EL12	RW	on page 149
ATCR_EL3	RW	on page 149
AVTCR_EL2	RW	on page 149

Name	Type	Description
LORC_EL1	RW	LORC_EL1, LORegion Control Register, EL1
LOREA_EL1	RW	LORegion End Address Register EL1
LORID_EL1	RO	LORID_EL1, LORegion ID Register, EL1
LORN_EL1	RW	LORN_EL1, LORegion Number Register, EL1
LORSA_EL1	RW	LORegion Start Address Register EL1
TCR_EL1	RW	TCR_EL1, Translation Control Register, EL1
TCR_EL2	RW	TCR_EL2, Translation Control Register, EL2
TCR_EL3	RW	TCR_EL3, Translation Control Register, EL3
TTBR0_EL1	RW	TTBR0_EL1, Translation Table Base Register 0, EL1
TTBR0_EL2	RW	TTBR0_EL2, Translation Table Base Register 0, EL2
TTBR0_EL3	RW	TTBR0_EL3, Translation Table Base Register 0, EL3
TTBR1_EL1	RW	TTBR1_EL1, Translation Table Base Register 1, EL1
TTBR1_EL2	RW	TTBR1_EL2, Translation Table Base Register 1, EL2
VTTBR_EL2	RW	VTTBR_EL2, Virtualization Translation Table Base Register, EL2

Virtualization registers

Name	Type	Description
ACTLR_EL2	RW	ACTLR_EL2, Auxiliary Control Register, EL2
AFSR0_EL2	RW	AFSR0_EL2, Auxiliary Fault Status Register 0, EL2

Name	Type	Description
AFSR1_EL2	RW	AFSR1_EL2, Auxiliary Fault Status Register 1, EL2
AMAIR_EL2	RW	AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2
CPTR_EL2	RW	CPTR_EL2, Architectural Feature Trap Register, EL2
ESR_EL2	RW	ESR_EL2, Exception Syndrome Register, EL2
HACR_EL2	RW	HACR_EL2, Hyp Auxiliary Configuration Register, EL2
HCR_EL2	RW	HCR_EL2, Hypervisor Configuration Register, EL2
HPFAR_EL2	RW	Hypervisor / Fault Address Register EL2
TCR_EL2	RW	TCR_EL2, Translation Control Register, EL2
VMPIDR_EL2	RW	Virtualization Multiprocessor ID Register EL2
VPIDR_EL2	RW	Virtualization Core ID Register EL2
VSESR_EL2	RW	VSESR_EL2, Virtual SError Exception Syndrome Register (Ares Specific)
VTCCR_EL2	RW	VTCCR_EL2, Virtualization Translation Control Register, EL2
VTTBR_EL2	RW	VTTBR_EL2, Virtualization Translation Table Base Register, EL2

Exception and handling registers

Name	Type	Description
AFSR0_EL1	RW	AFSR0_EL1, Auxiliary Fault Status Register 0, EL1
AFSR0_EL2	RW	AFSR0_EL2, Auxiliary Fault Status Register 0, EL2
AFSR0_EL3	RW	AFSR0_EL3, Auxiliary Fault Status Register 0, EL3
AFSR1_EL1	RW	AFSR1_EL1, Auxiliary Fault Status Register 1, EL1

Name	Type	Description
AFSR1_EL2	RW	AFSR1_EL2 , Auxiliary Fault Status Register 1, EL2
AFSR1_EL3	RW	AFSR1_EL3 , Auxiliary Fault Status Register 1, EL3
DISR_EL1	RW	DISR_EL1 , Deferred Interrupt Status Register, EL1
ESR_EL1	RW	ESR_EL1 , Exception Syndrome Register, EL1
ESR_EL2	RW	ESR_EL2 , Exception Syndrome Register, EL2
ESR_EL3	RW	ESR_EL3 , Exception Syndrome Register, EL3
HPFAR_EL2	RW	Hypervisor Fault Address Register EL2
VDISR_EL2	RW	VDISR_EL2 , Virtual Deferred Interrupt Status Register, EL2
VSESR_EL2	RW	VSESR_EL2 , Virtual SError Exception Syndrome Register (Ares Specific)

Implementation defined registers

Name	Type	Description
ATCR_EL1	RW	on page 148
ATCR_EL2	RW	on page 149
ATCR_EL12	RW	on page 149
ATCR_EL3	RW	on page 149
AVTCR_EL2	RW	on page 149
CPUACTLR_EL1	RW	CPUACTLR_EL1 , CPU Auxiliary Control Register, EL1
CPUACTLR2_EL1	RW	CPUACTLR2_EL1 , CPU Auxiliary Control Register 2, EL1
CPUCFR_EL1	RO	CPUCFR_EL1 , CPU Configuration Register, EL1
CPUECTLR_EL1	RW	CPUECTLR_EL1 , CPU Extended Control Register, EL1

Name	Type	Description
CPUPWRCTLR_EL1	RW	CPUPWRCTLR_EL1, Power Control Register, EL1
ERXPFPGCDNR_EL1	RW	ERXPFPGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1
ERXPFPGCTLR_EL1	RW	ERXPFPGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1
ERXPFGFR_EL1	RW	ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1

The following table shows the 32-bit wide implementation defined Cluster registers. Details of these registers can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table 51: Cluster registers

Name	Copro	CRn	Opcl	CRm	Opc2	Width	Description
CLUSTERCFR_EL1		c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR_EL1		c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR_EL1		c15	0	c3	2	32-bit	Cluster ECO ID.
CLUSTERACTLR_EL1		c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR_EL1		c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCTLR_EL1		c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWRDN_EL1		c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWRSTAT_EL1		c15	0	c3	7	32-bit	Cluster power status register.

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERTHREADSID_EL1	c15	c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID_EL1	c15	c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID_EL1	c15	c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERRTGR_EL1	c15	c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS_EL1	c15	c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT_EL1	c15	c15	0	c4	5	32-bit	Cluster L3 hit counter register.
CLUSTERL3MISS_EL1	c15	c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERTHREADSIDOVR_EL1	c15	c15	0	c4	7	32-bit	Cluster thread scheme ID override register.
CLUSTERPMCR_EL1	c15	c15	0	c5	0	32-bit	Cluster Performance Monitors Control Register.
CLUSTERPMCNENSET_EL1	c15	c15	0	c5	1	32-bit	Cluster Count Enable Set Register.
CLUSTERPMCNENCLR_EL1	c15	c15	0	c5	2	32-bit	Cluster Count Enable Clear Register.
CLUSTERPMOVSET_EL1	c15	c15	0	c5	3	32-bit	Cluster Overflow Flag Status Set.

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERPM0VSCLR_EL1	0	c15	0	c5	4	32-bit	Cluster Overflow Flag Status Clear
CLUSTERPM0SELR_EL1	0	c15	0	c5	5	32-bit	Cluster <i>Event</i> Counter Selection Register
CLUSTERPM0INTENSET_EL1	0	c15	0	c5	6	32-bit	Cluster Interrupt Enable Set Register
CLUSTERPM0INTENCLR_EL1	0	c15	0	c5	7	32-bit	Cluster Interrupt Enable Clear Register
CLUSTERPM0XEVTYR_EL1	0	c15	0	c6	1	32-bit	Cluster Selected Type and Filter Register
CLUSTERPM0XEVCNTR_EL1	0	c15	0	c6	2	32-bit	Cluster Selected Counter Register
Reserved/ RAZ	3	c15	0	c6	3	32-bit	Cluster Monitor Debug Configuration Register
CLUSTERPM0CEID0_EL1	0	c15	0	c6	4	32-bit	Cluster Common Identification ID0 Register
CLUSTERPM0CEID1_EL1	0	c15	0	c6	5	32-bit	Cluster Common Identification ID1 Register
CLUSTERPM0CLAIMSET_EL1	0	c15	0	c6	6	32-bit	Cluster Performance Monitor Claim Tag Set Register

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERPMCCCLAIMCLR_EL3		c6	0	c6	7	32-bit	Cluster Performance Monitor Claim Tag Clear Register

Security

Name	Type	Description
ACTLR_EL3	RW	ACTLR_EL3, Auxiliary Control Register, EL3
AFSR0_EL3	RW	AFSR0_EL3, Auxiliary Fault Status Register 0, EL3
AFSR1_EL3	RW	AFSR1_EL3, Auxiliary Fault Status Register 1, EL3
AMAIR_EL3	RW	AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3
CPTR_EL3	RW	CPTR_EL3, Architectural Feature Trap Register, EL3
MDCR_EL3	RW	MDCR_EL3, Monitor Debug Configuration Register, EL3

Reset management registers

Name	Type	Description
RMWR_EL3	RW	R_EL3, Reset Management Register
RVBAR_EL3	RW	RVBAR_EL3, Reset Vector Base Address Register, EL3

Address registers

Name	Type	Description
R_EL1	RW	R_EL1, Physical Address Register, EL1

ACTLR_EL1, Auxiliary Control Register, EL1

ACTLR_EL1 provides *IMPLEMENTATION DEFINED* configuration and control options for execution at EL1 and EL0.

Bit field descriptions

ACTLR_EL1 is a 64-bit register, and is part of:

- The Other system control registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

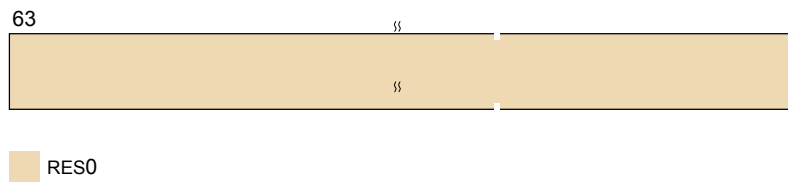


Figure 8: ACTLR_EL1 bit assignments

RES0, [63:0]	RES0	Reserved.
Configurations	There are no configuration notes. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

ACTLR_EL2, Auxiliary Control Register, EL2

The ACTLR_EL2 provides *IMPLEMENTATION DEFINED* configuration and control options for EL2.

Bit field descriptions

ACTLR_EL2 is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

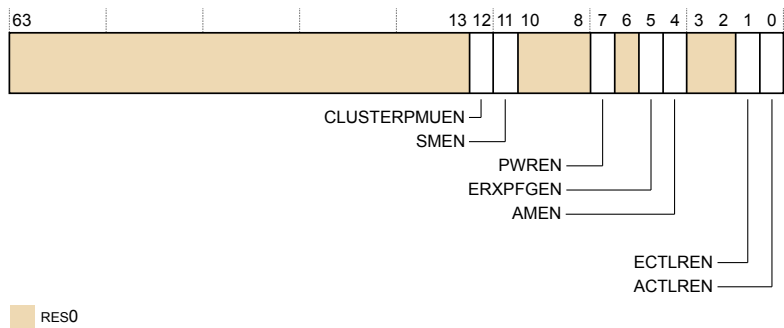


Figure 9: ACTLR_EL2 bit assignments

RES0, [63:13]	RES0	Reserved.
CLUSTERPMUEN, [12]		
	0	CLUSTERPM* registers are not write-accessible from a lower <i>Exception level</i> . This is the reset value.
	1	CLUSTERPM* registers are write-accessible from EL1 Non-secure if they are write-accessible from EL2.
SMEN, [11]		
	0	Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are not write-accessible from EL1 Non-secure. This is the reset value.
	1	Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are write-accessible from EL1 Non-secure if they are write-accessible from EL2.
RES0, [9:8]	RES0	Reserved.

PWREN, [7]

Power Control Registers enable. The possible values are:

- | | |
|----------|--|
| 0 | Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write-accessible from EL1 Non-secure. This is the reset value. |
| 1 | Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are write-accessible from EL1 Non-secure if they are write-accessible from EL2. |

RES0, [6]

RES0 Reserved.

ERXPFGEN, [5]

Error Record Registers enable. The possible values are:

- | | |
|----------|---|
| 0 | ERXPFG* are not write-accessible from EL1 Non-secure. This is the reset value. |
| 1 | ERXPFG* are write-accessible from EL1 Non-secure if they are write-accessible from EL2. |

AMEN, [4]

Activity Monitor enable. The possible values are:

- | | |
|----------|--|
| 0 | Non-secure accesses from EL1 and EL0 to activity monitor registers are trapped to EL2. |
| 1 | Non-secure accesses from EL1 and EL0 to activity monitor registers are not trapped to EL2. |

RES0, [3:2]

RES0 Reserved.

ECTLREN, [1]

Extended Control Registers enable. The possible values are:

- | | |
|----------|---|
| 0 | CPUECTLR and CLUSTERECTLR are not write-accessible from EL1 |
|----------|---|

		Non-secure. This is the reset value.
1		CPUECTLR and CLUSTERECTLR are write-accessible from EL1 Non-secure if they are write-accessible from EL2.
Configurations		
	Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

ACTLR_EL3, Auxiliary Control Register, EL3

The ACTLR_EL3 provides *IMPLEMENTATION DEFINED* configuration and control options for EL3.

Bit field descriptions

ACTLR_EL3 is a 64-bit register, and is part of:

- The Other system control registers functional group.
- The Security registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

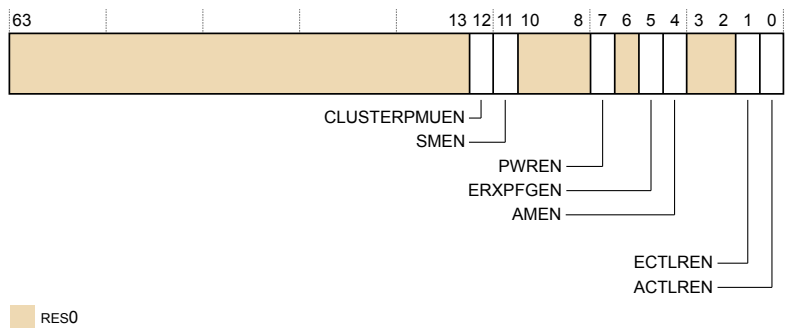


Figure 10: ACTLR_EL3 bit assignments

RES0, [63:13]	<i>RES0</i>	Reserved.
CLUSTERPMUEN, [12]	0	CLUSTERPM* registers are not write-accessible from a lower <i>Exception level</i> . This is the reset value.
	1	CLUSTERPM* registers are write-accessible from EL2 and EL1 Secure.

SMEN, [11]

Scheme Management Registers enable. The possible values are:

0	Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are not write-accessible from EL2 and EL1 Secure. This is the reset value.
1	Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are write-accessible from EL2 and EL1 Secure.

TSIDEN, [10]

Thread Scheme ID Register enable. The possible values are:

0	Register CLUSTERTHREADSID is not write-accessible from EL2 and EL1 Secure. This is the reset value.
1	Register CLUSTERTHREADSID is write-accessible from EL2 and EL1 Secure.

RES0, [9:8]

RES0 Reserved.

PWREN, [7]

Power Control Registers enable. The possible values are:

0	Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write-accessible from EL2 and EL1 Secure. This is the reset value.
1	Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are

		write-accessible from EL2 and EL1 Secure.
RES0, [6]	RES0	Reserved.
ERXPFGEN, [5]	Error Record Registers enable. The possible values are:	
	0	ERXPFG* are not write-accessible from EL2 and EL1 Secure. This is the reset value.
	1	ERXPFG* are write-accessible from EL2 and EL1 Secure.
AMEN, [4]	Activity Monitor enable. The possible values are:	
	0	Accesses from EL2, EL1 and EL0 to activity monitor registers are trapped to EL3.
	1	Accesses from EL2, EL1 and EL0 to activity monitor registers are not trapped to EL3.
RES0, [3:2]	RES0	Reserved.
ECTLREN, [1]	Extended Control Registers enable. The possible values are:	
	0	CPUECTLR and CLUSTERECTLR are not write-accessible from EL2 and EL1 Secure. This is the reset value.
	1	CPUECTLR and CLUSTERECTLR are write-accessible from EL2 and EL1 Secure.

Configurations

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AFSR0_EL1, Auxiliary Fault Status Register 0, EL1

AFSR0_EL1 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL1. In the Cortex[®]-A76 core, no additional information is provided for these exceptions. Therefore this register is not used.

Bit field descriptions

AFSR0_EL1 is a 32-bit register, and is part of:

- The Exception and *fault* handling registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

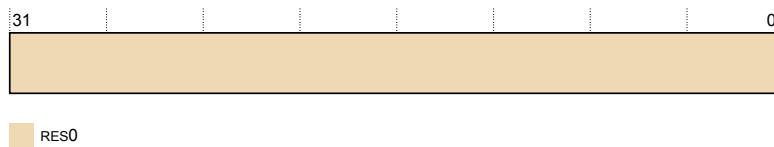


Figure 11: AFSR0_EL1 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AFSR0_EL2, Auxiliary Fault Status Register 0, EL2

AFSR0_EL2 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL2.

Bit field descriptions

AFSR0_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and *fault* handling registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

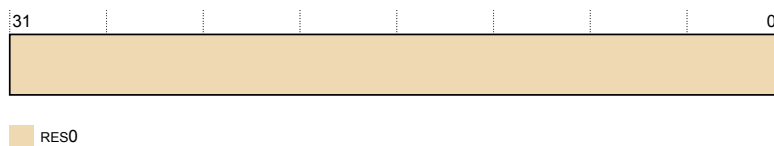


Figure 12: AFSR0_EL2 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AFSR0_EL3, Auxiliary Fault Status Register 0, EL3

AFSR0_EL3 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL3. In the Cortex®-A76 core, no additional information is provided for these exceptions. Therefore this register is not used.

Bit field descriptions

AFSR0_EL3 is a 32-bit register, and is part of:

- The Exception and *fault* handling registers functional group.
- The Security registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

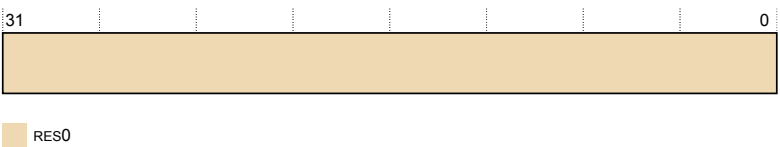


Figure 13: AFSR0_EL3 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AFSR1_EL1, Auxiliary Fault Status Register 1, EL1

AFSR1_EL1 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL1. This register is not used in Cortex®-A76.

Bit field descriptions

AFSR1_EL1 is a 32-bit register, and is part of:

- The Exception and *fault* handling registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

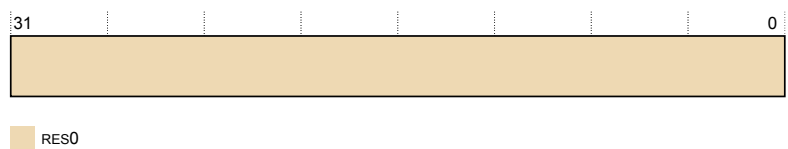


Figure 14: AFSR1_EL1 bit assignments

RES0, [31:0]	Reserved, <i>RES0</i> .
Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>

AFSR1_EL2, Auxiliary Fault Status Register 1, EL2

AFSR1_EL2 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL2. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AFSR1_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and *fault* handling registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

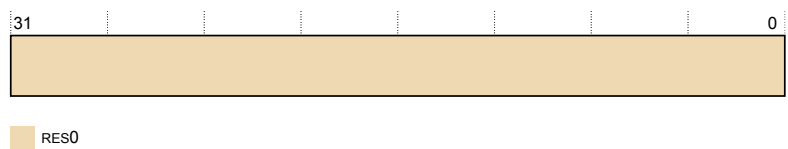


Figure 15: AFSR1_EL2 bit assignments

RES0, [31:0]	Reserved, <i>RES0</i> .
Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>

AFSR1_EL3, Auxiliary Fault Status Register 1, EL3

AFSR1_EL3 provides additional *IMPLEMENTATION DEFINED* fault status information for exceptions that are taken to EL3. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AFSR1_EL3 is a 32-bit register, and is part of:

- The Exception and *fault* handling registers functional group.
- The Security registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

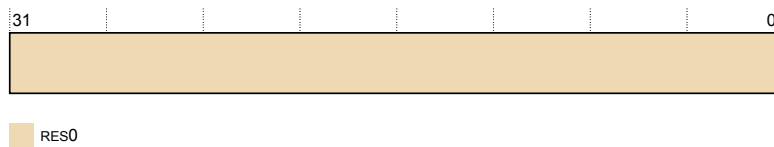


Figure 16: AFSR1_EL3 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AIDR_EL1, Auxiliary ID Register, EL1

AIDR_EL1 provides *IMPLEMENTATION DEFINED* identification information. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AIDR_EL1 is a 32-bit register, and is part of:

- The Identification registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

This register is Read Only.

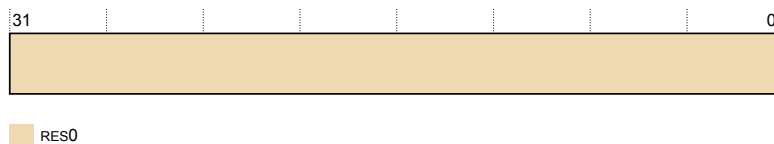


Figure 17: AIDR_EL1 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1

AMAIR_EL1 provides *IMPLEMENTATION DEFINED* memory attributes for the memory regions specified by MAIR_EL1. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AMAIR_EL1 is a 64-bit register, and is part of:

- The Virtual memory control registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

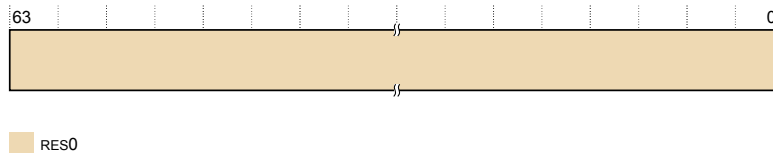


Figure 18: AMAIR_EL1 bit assignments

RES0, [63:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2

AMAIR_EL2 provides *IMPLEMENTATION DEFINED* memory attributes for the memory regions specified by MAIR_EL2. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AMAIR_EL2 is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

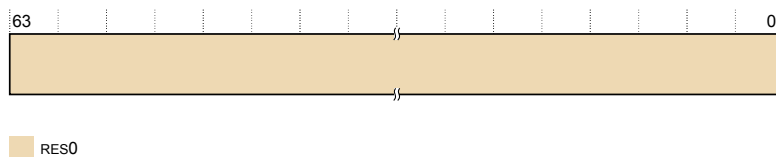


Figure 19: AMAIR_EL1 bit assignments

RES0, [63:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3

AMAIR_EL3 provides *IMPLEMENTATION DEFINED* memory attributes for the memory regions specified by MAIR_EL3. This register is not used in the Cortex®-A76 core.

Bit field descriptions

AMAIR_EL3 is a 64-bit register, and is part of:

- The Virtual memory control registers functional group.
- The Security registers functional group.
- The *IMPLEMENTATION DEFINED* functional group.

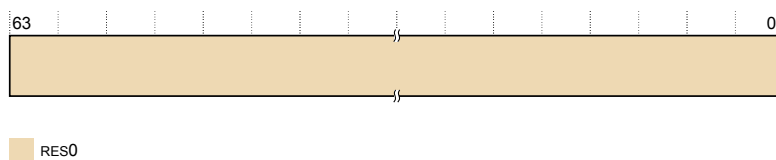


Figure 20: AMAIR_EL3 bit assignments

RES0, [63:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CLIDR_EL1, Cache Level ID Register, EL1

The CLIDR_EL1 identifies the type of cache, or caches, implemented at each level, up to a maximum of seven levels.

Bit field descriptions

It also identifies the *Level of Coherency* (LoC) and *Level of Unification* (LoU) for the cache hierarchy.

CLIDR_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

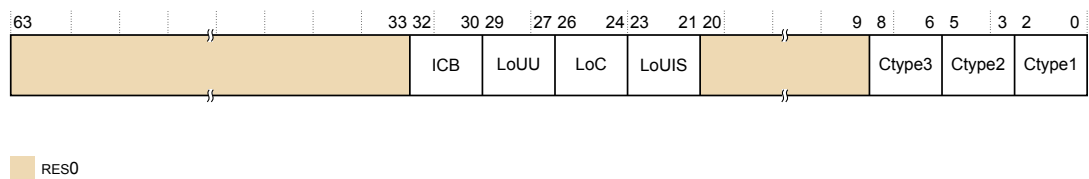


Figure 21: CLIDR_EL1 bit assignments

RES0, [63:33]	RES0	Reserved.
ICB, [32:30]		Inner cache boundary. This field indicates the boundary between the inner and the outer domain:
	010	L2 cache is the highest inner level.
	011	L3 cache is the highest inner level.
LoUU, [29:27]		Indicates the Level of Unification Uniprocessor for the cache hierarchy:
	000	No levels of cache need to <i>cleaned</i> or <i>invalidated</i> when ing or invalidating to the Point of Unification.

	This is the value if no caches are configured.				
LoC, [26:24]	Indicates the Level of Coherency for the cache hierarchy: <table> <tr> <td>010</td><td>L3 cache is not implemented.</td></tr> <tr> <td>011</td><td>L3 cache is implemented.</td></tr> </table>	010	L3 cache is not implemented.	011	L3 cache is implemented.
010	L3 cache is not implemented.				
011	L3 cache is implemented.				
LoUIS, [23:21]	Indicates the <i>Level of Unification Inner Shareable</i> (LoUIS) for the cache hierarchy. <table> <tr> <td>000</td><td>No cache level needs ing to Point of Unification.</td></tr> </table>	000	No cache level needs ing to Point of Unification.		
000	No cache level needs ing to Point of Unification.				
RES0, [20:9]	No cache at levels L7 down to L4. <table> <tr> <td>RES0</td><td>Reserved.</td></tr> </table>	RES0	Reserved.		
RES0	Reserved.				
Ctype3, [8:6]	Indicates the type of cache if the <i>core</i> implements L3 cache. If present, unified instruction and data caches at Level 3: <table> <tr> <td>100</td><td>Both per- L2 and <i>cluster</i> L3 caches are present.</td></tr> <tr> <td>000</td><td>All other options.</td></tr> </table> If Ctype2 has a value of 000, then the value of Ctype3 must be <i>ignored</i> .	100	Both per- L2 and <i>cluster</i> L3 caches are present.	000	All other options.
100	Both per- L2 and <i>cluster</i> L3 caches are present.				
000	All other options.				
Ctype2, [5:3]	Indicates the type of unified instruction and data caches at Level 2: <table> <tr> <td>100</td><td>Either per- L2 or L2 cache is present.</td></tr> <tr> <td>000</td><td>All other options.</td></tr> </table>	100	Either per- L2 or L2 cache is present.	000	All other options.
100	Either per- L2 or L2 cache is present.				
000	All other options.				
Ctype1, [2:0]	Indicates the type of cache implemented at L1: <table> <tr> <td>011</td><td>Separate instruction and data caches at L1.</td></tr> </table>	011	Separate instruction and data caches at L1.		
011	Separate instruction and data caches at L1.				
Configurations	There are no configuration notes. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .				

CPACR_EL1, Architectural Feature Access Control Register, EL1

The CPACR_EL1 controls access to trace functionality and access to registers associated with Advanced SIMD and floating-point execution.

Bit field descriptions

CPACR_EL1 is a 32-bit register, and is part of the Other system control registers functional group.

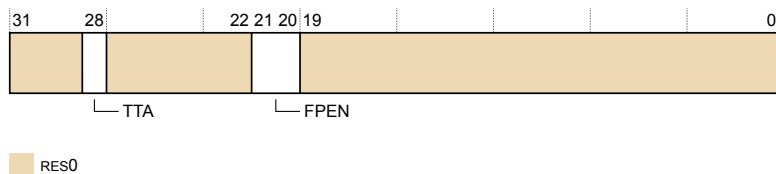


Figure 22: CPACR_EL1 bit assignments

RES0, [31:29]

RES0

Reserved.

TTA, [28]

Traps EL0 and EL1 System register accesses to all implemented trace registers to EL1, from both Execution states. This bit is *RES0*. The *core* does not provide System Register access to *ETM* control.

Configurations

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CPTR_EL2, Architectural Feature Trap Register, EL2

The CPTR_EL2 controls trapping to EL2 for accesses to CPACR, trace functionality and registers associated with Advanced SIMD and floating-point execution. It also controls EL2 access to this functionality.

Bit field descriptions

CPTR_EL2 is a 32-bit register, and is part of the Virtualization registers functional group.

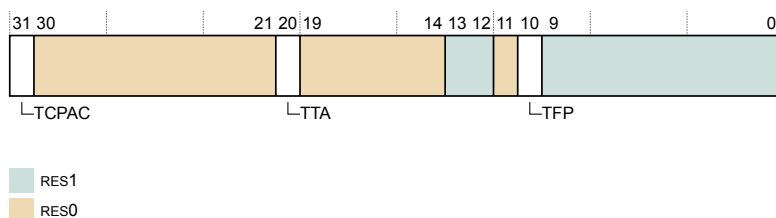


Figure 23: CPTR_EL2 bit assignments

TTA, [20]

Trap Trace Access.

This bit is not implemented. *RES0*.

Configurations	<p>RW fields in this register reset to <i>unknown</i> values.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>
----------------	---

CPTR_EL3, Architectural Feature Trap Register, EL3

The CPTR_EL3 controls trapping to EL3 of access to CPACR_EL1, CPTR_EL2, trace functionality and registers associated with Advanced SIMD and floating-point execution.

Bit field descriptions

It also controls EL3 access to trace functionality and registers associated with *Advanced SIMD* and *floating-point* execution.

CPTR_EL3 is a 32-bit register, and is part of the Security registers functional group.

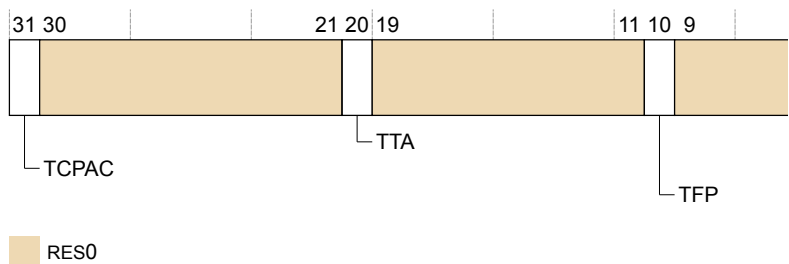


Figure 24: CPTR_EL3 bit assignments

TTA, [20]	Trap Trace Access. Not implemented. <i>RES0</i> .
TFP, [10]	Traps all accesses to SVE, and functionality to EL3. This applies to all <i>Exception levels</i> , both Security states, and both Execution states. The possible values are: 0 Does not cause any instruction to be trapped. This is the reset value. 1 Any attempt at any to execute an instruction that uses the registers that are associated with SVE, and is trapped to EL3, subject to the <i>exception</i> prioritization rules.

Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>
----------------	--

CPUACTLR_EL1, CPU Auxiliary Control Register, EL1

The CPUACTLR_EL1 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUACTLR_EL1 is a 64-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

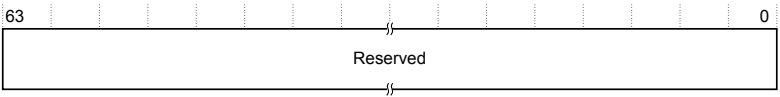


Figure 25: CPUACTLR_EL1 bit assignments

Reserved, [63:0]

Reserved for Arm® internal use.

Configurations

CPUACTLR_EL1 is common to the Secure and Non-secure states.

Usage constraints

Accessing the CPUACTLR_EL1

The CPU Auxiliary Control Register can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the *MMU* is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be *read* with the MRS instruction using the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C1_0	11	000	1111	0001	000

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_0	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_0	x	0	1	-	RW	RW	RW
S3_0_C15_C1_0	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The [core](#) cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated [exceptions](#), see *Synchronous prioritization in the Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1

The CPUACTLR2_EL1 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUACTLR2_EL1 is a 64-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.



Figure 26: CPUACTLR2_EL1 bit assignments

Reserved, [63:0]

Reserved for Arm® internal use.

Configurations

CPUACTLR2_EL1 is common to the Secure and Non-secure states.

Usage constraints

Accessing the CPUACTLR2_EL1

The CPUACTLR2_EL1 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the [MMU](#) is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be [read](#) using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	Op0	Op1	CRn	CRm	Op2
S3_0_C15_C1_1	11	000	1111	0001	001

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_1	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_1	x	0	1	-	RW	RW	RW
S3_0_C15_C1_1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The [PE](#) cannot be executing at this [Exception level](#), so this access is not possible.

Traps and enables

For a description of the prioritization of any generated [exceptions](#), see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to [AArch64](#) state, and see *Synchronous prioritization* for s taken to state.

Write access to this register from EL1 or EL2 depends on the value of bit[0] of ACTLR_EL2 and ACTLR_EL3.

CPUACTLR3_EL1, CPU Auxiliary Control Register 3, EL1

The CPUACTLR3_EL1 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUACTLR3_EL1 is a 64-bit register, and is part of the *implementation defined* registers functional group.



Figure 27: CPUACTLR3_EL1 bit assignments

Reserved, [63:0]

Reserved for Arm® internal use.

Configurations

CPUACTLR3_EL1 is common to the Secure and Non-secure states.

Usage constraints

Accessing the CPUACTLR3_EL1

The CPUACTLR3_EL1 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the [MMU](#) is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be [read](#) using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	Op0	Op1	CRn	CRm	Op2
S3_0_C15_C1_2	11	000	1111	0001	010

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_2	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_2	x	0	1	-	RW	RW	RW
S3_0_C15_C1_2	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The [PE](#) cannot be executing at this [Exception level](#), so this access is not possible.

Traps and enables

For a description of the prioritization of any generated [exceptions](#), see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to [AArch64](#) state, and see *Synchronous prioritization* for s taken to state.

Write access to this register from EL1 or EL2 depends on the value of bit[0] of ACTLR_EL2 and ACTLR_EL3.

CPUCFR_EL1, CPU Configuration Register, EL1

The CPUCFR_EL1 provides configuration information for the core.

Bit field descriptions

CPUCFR_EL1 is a 32-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

This register is Read Only.



RES0

Figure 28: CPUCFR_EL1 bit assignments

RES0, [31:2]	Reserved, <i>RES0</i> .
ECC, [1:0]	Indicates whether ECC is present or not. The possible values are: <div> <div>00</div> <div>ECC is not present.</div> <div>01</div> <div>ECC is present.</div> </div>
Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>

Usage constraints

Accessing the CPUCFR_EL1	<p>This register can be <i>read</i> with the MRS instruction using the following syntax:</p> <pre>MRS <Xt>, <systemreg></pre> <p>To access the CPUCFR_EL1:</p> <pre>MRS <Xt>, CPUCFR_EL1 ; Read CPUCFR_EL1 into Xt</pre>
---------------------------------	--

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C0_0	11	000	1111	0000	000

Accessibility	This register is accessible in software as follows:
----------------------	---

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C0_0	x	x	0	-	RO	n/a	RO
S3_0_C15_C0_0	x	0	1	-	RO	RO	RO
S3_0_C15_C0_0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The *PE* cannot be executing at this *Exception level*, so this access is not possible.

CPUECTLR_EL1, CPU Extended Control Register, EL1

The CPUECTLR_EL1 provides additional *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUECTLR_EL1 is a 64-bit register, and is part of the 64-bit registers functional group.

This register resets to value 00000000961563000.

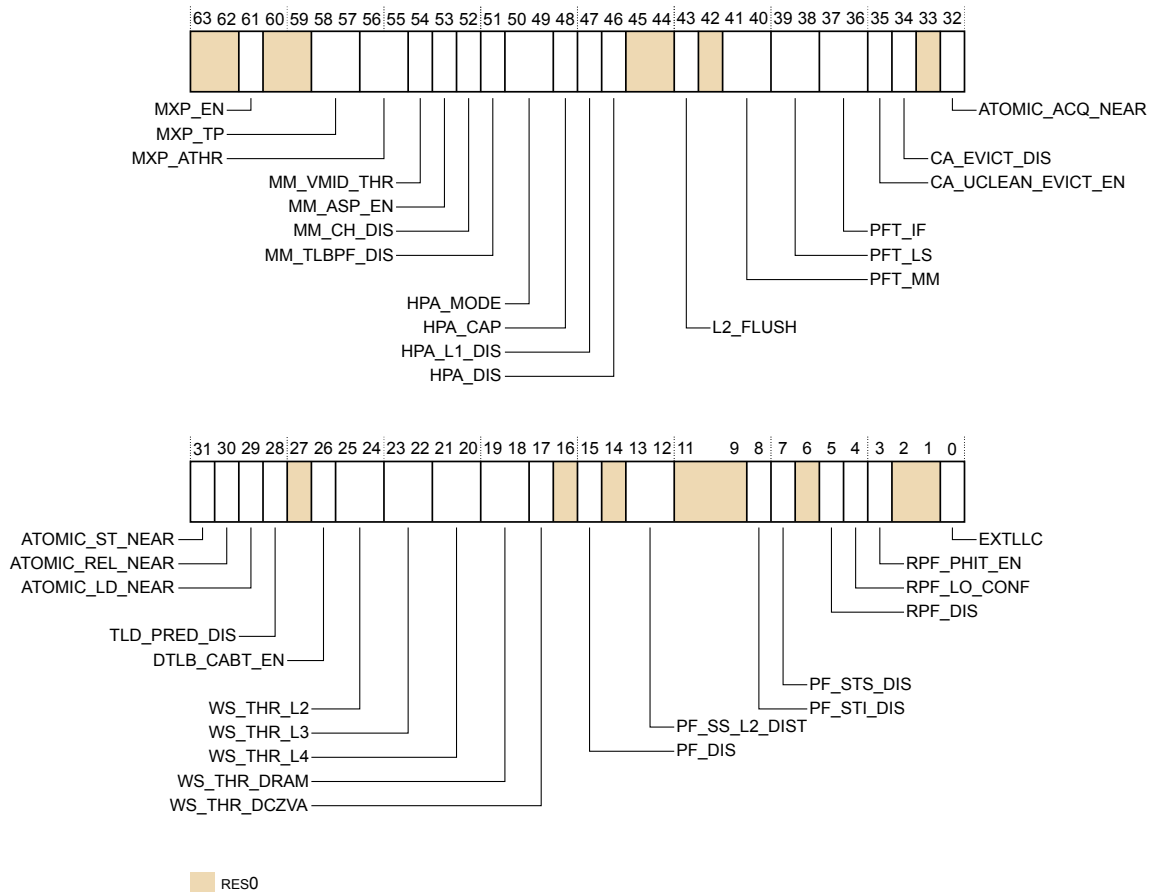


Figure 29: CPUECTLR_EL1 bit assignments

RES0, [63:62]

res0

Reserved.

MXP_EN, [61]

Max-power throttle enable. The possible values are:

0

Disables max-power throttling mechanism. This is the reset value.

1

Enables max-power throttling mechanism.



Note: Both the MXP_EN bit and the MPMEN

input pin at the DSU *cluster* level must be asserted to enable the max-power throttling mechanism.

RES0, [60:59]

res0

Reserved.

MXP_TP, [58:57]

Percentage of throttling in the Load-Store and Vector Execute units during the period when throttling has been triggered and is active. The possible values are:

00	Throttle by 60%. This is the reset value.
01	Throttle by 50%.
10	Throttle by 40%.
11	Throttle by 30%.

MXP_ATHR, [56:55]

Peak activity threshold at which max-power throttling is triggered. The possible values are:

00	Max-power throttling triggered at 70% of peak activity. This is the reset value.
01	Max-power throttling triggered at 60% of peak activity.
10	Max-power throttling triggered at 50% of peak activity.
11	Max-power throttling triggered at 40% of peak activity.

MM_VMID_THR, [54]

VMID filter threshold. The possible values are:

0	Flush VMID filter after 16 unique VMID allocations to the <i>MMU</i> Translation Cache. This is the reset value.
1	Flush VMID filter after 32 unique VMID allocations to the Translation Cache.

MM_ASP_EN, [53]

Disables allocation of splintered pages in L2 *TLB*. The possible values are:

	0	Enables allocation of splintered pages in the L2 . This is the reset value.
	1	Disables allocation of splintered pages in the L2 .
MM_CH_DIS, [52]	Disables use of contiguous hint. The possible values are:	
	0	Enables use of contiguous hint. This is the reset value.
	1	Disables use of contiguous hint.
MM_PF_DIS, [51]	Disables L2 prefetcher. The possible values are:	
	0	Enables L2 prefetcher. This is the reset value.
	1	Disables L2 prefetcher.
HPA_MODE, [50:49]	<i>Hardware Page Aggregation</i> (HPA) mode. The possible values are:	
	00	Moderately conservative hardware page aggregation. This is the reset value.
	01	Aggressive hardware page aggregation.
	10	Moderately aggressive hardware page aggregation.
	11	Conservative hardware page aggregation.
HPA_CAP, [48]	Limited or full hardware page aggregation selection . The possible values are:	
	0	Limited hardware page aggregation. This is the reset value.
	1	Full hardware page aggregation.
HPA_L1_DIS, [47]	Disables HPA in L1 s (but continues to use HPA in L2). The possible values are:	
	0	Enables hardware page aggregation in L1 s. This is the reset value.
	1	Disables hardware page aggregation in L1 s.

HPA_DIS, [46]

Disables hardware page aggregation. The possible values are:

- | | |
|----------|---|
| 0 | Enables hardware page aggregation. This is the reset value. |
| 1 | Disables hardware page aggregation. |

RES0, [45:44]

res0 Reserved.

L2_FLUSH, [43]

Allocation behavior of copybacks caused by L2 cache hardware flush and DC C1SW instructions targeting the L2 cache. If it is known that data is likely to be used soon by another *core*, setting this bit can improve system performance. The possible values are:

- | | |
|----------|---|
| 0 | L2 cache flushes and <i>invalidates</i> by set/way do not allocate in the L3 cache. Cache lines in the UniqueDirty state cause <i>WriteBack</i> transactions with the allocation hint cleared, while cache lines in UniqueClean or SharedClean states cause address-only Evict transactions. This is the reset value. |
| 1 | L2 cache flushes by set/way allocate in the L3 cache. Cache lines in the UniqueDirty or UniqueClean state cause BackFull or EvictFull transactions, respectively, both with the allocation hint set. Cache lines in the SharedClean state cause address-only Evict transactions. |

RES0, [42]

res0 Reserved.

PFT_MM, [41:40]

DRAM prefetch using PrefetchTgt transactions for table walk requests. The possible values are:

- | | |
|-----------|--|
| 00 | Disable prefetchtgt generation for requests from the Memory Management unit (). This is the reset value. |
|-----------|--|

	01	Conservatively generate prefetchtgt for <i>cacheable</i> requests from the , always generate for non-.
	10	Aggressively generate prefetchtgt for requests from the , always generate for non-.
	11	Always generate prefetchtgt for requests from the , always generate for non-.
PFT_LS, [39:38]	DRAM prefetch using PrefetchTgt transactions for load and store requests. The possible values are:	
	00	Disable prefetchtgt generation for requests from the Load-Store unit (LS). This is the reset value.
	01	Conservatively generate prefetchtgt for requests from the LS, always generate for non-.
	10	Aggressively generate prefetchtgt for requests from the LS, always generate for non-.
	11	Always generate prefetchtgt for requests from the LS, always generate for non-.
PFT_IF, [37:36]	DRAM prefetch using PrefetchTgt transactions for instruction fetch requests. The possible values are:	
	00	Disable prefetchtgt generation for requests from the Instruction Fetch unit (IF). This is the reset value.
	01	Conservatively generate prefetchtgt for requests from the IF, always generate for non-.
	10	Aggressively generate prefetchtgt for requests from the IF, always generate for non-.
	11	Always generate prefetchtgt for requests

	from the IF, always generate for non-.				
CA_UCLEAN_EVICT_EN, [35]	<p>Enables sending Evict transactions on the CPU CHI interface for UniqueClean evictions. Evict transactions update downstream caches. Enable Evict transactions only if there is an additional level of cache below the CPU's Level 2 cache. The possible values are:</p> <table> <tr> <td>0</td><td>Disables sending data with UniqueClean evictions.</td></tr> <tr> <td>1</td><td>Enables sending data with UniqueClean evictions. This is the reset value.</td></tr> </table>	0	Disables sending data with UniqueClean evictions.	1	Enables sending data with UniqueClean evictions. This is the reset value.
0	Disables sending data with UniqueClean evictions.				
1	Enables sending data with UniqueClean evictions. This is the reset value.				
CA_EVICT_DIS, [34]	<p>Disables sending of Evict transactions on the CPU CHI interface for <i>clean</i> cache lines that are evicted from the . Evict transactions are required only if the system contains a snoop filter that requires notification when the evicts the cache line. The possible values are:</p> <table> <tr> <td>0</td><td>Enables sending Evict transactions. This is the reset value.</td></tr> <tr> <td>1</td><td>Disables sending Evict transactions.</td></tr> </table>	0	Enables sending Evict transactions. This is the reset value.	1	Disables sending Evict transactions.
0	Enables sending Evict transactions. This is the reset value.				
1	Disables sending Evict transactions.				
RES0, [33]	<table> <tr> <td><i>res0</i></td><td>Reserved.</td></tr> </table>	<i>res0</i>	Reserved.		
<i>res0</i>	Reserved.				
ATOMIC_ACQ_NEAR, [32]	<p>An atomic instruction to WB memory with acquire semantics that does not hit in the cache in Exclusive state, may make up to one fill request. The possible values are:</p> <table> <tr> <td>0</td><td>Acquire-atomic is near if cache line is already Exclusive, otherwise make far atomic request.</td></tr> <tr> <td>1</td><td>Acquire-atomic will make up to 1 fill request to perform near. This is the reset value.</td></tr> </table>	0	Acquire-atomic is near if cache line is already Exclusive, otherwise make far atomic request.	1	Acquire-atomic will make up to 1 fill request to perform near. This is the reset value.
0	Acquire-atomic is near if cache line is already Exclusive, otherwise make far atomic request.				
1	Acquire-atomic will make up to 1 fill request to perform near. This is the reset value.				
ATOMIC_ST_NEAR, [31]	<p>A store atomic instruction to WB memory that does not hit in the cache in Exclusive state, may make up to one fill request. The possible values are:</p> <table> <tr> <td>0</td><td>Store-atomic is near if cache line is already Exclusive, otherwise make far atomic request. This is the reset value.</td></tr> </table>	0	Store-atomic is near if cache line is already Exclusive, otherwise make far atomic request. This is the reset value.		
0	Store-atomic is near if cache line is already Exclusive, otherwise make far atomic request. This is the reset value.				

	1	Store-atomic will make up to 1 fill request to perform near.
ATOMIC_REL_NEAR, [30]	An atomic instruction to WB memory with release semantics that does not hit in the cache in Exclusive state, may make up to one fill request. The possible values are:	
	0	Release-atomic is near if cache line is aly Exclusive, otherwise make far atomic request.
	1	Release-atomic will make up to 1 fill request to perform near. This is the reset value.
ATOMIC_LD_NEAR, [29]	A load atomic (including SWP and CAS) instruction to WB memory that does not hit in the cache in Exclusive state, may make up to one fill request. The possible values are:	
	0	Load-atomic is near if cache line is aly Exclusive, otherwise make far atomic request.
	1	Load-atomic will make up to 1 fill request to perform near. This is the reset value.
TLD_PRED_DIS, [28]	Disables Transient Load Prediction. The possible values are:	
	0	Enables transient load prediction. This is the reset value.
	1	Disables transient load prediction.
RES0, [27]	<i>res0</i>	Reserved.
D_CABT_EN , [26]	Enables Conflict <i>Data Abort</i> Exception. The possible values are:	
	0	Disables conflict data <i>abort exception</i> . This is the reset value.
	1	Enables conflict data .
WS_THR_L2, [25:24]	Threshold for direct stream to L2 cache on store. The possible values are:	

	00	256B.
	01	4KB. This is the reset value.
	10	8KB.
	11	Disables direct stream to L2 cache on store.
WS_THR_L3, [23:22]	Threshold for direct stream to L3 cache on store. The possible values are:	
	00	768B.
	01	16KB. This is the reset value.
	10	32KB.
	11	Disables direct stream to L3 cache on store.
WS_THR_L4, [21:20]	Threshold for direct stream to L4 cache on store. The possible values are:	
	00	16KB.
	01	64KB. This is the reset value.
	10	128KB.
	11	Disables direct stream to L4 cache on store.
WS_THR_DRAM, [19:18]	Threshold for direct stream to DRAM on store. The possible values are:	
	00	64KB.
	01	1MB, for memory designated as outer-allocate. This is the reset value.
	10	1MB, allocating irrespective of outer-allocation designation.
	11	Disables direct stream to DRAM on store.
WS_THR_DCZVA, [17]	Have DCZVA use a lower WS_THR_L2 configuration. The possible values are:	
	0	DCZVA behaves like normal store wrt WS_THR_L2.
	1	DCZVA will use one lower stream threshold from

		WS_THR_L2. This is the reset value.
RES0, [16]	<i>res0</i>	Reserved.
PF_DIS, [15]		Disables data-side hardware <i>prefetching</i> . The possible values are:
	0	Enables hardware . This is the reset value.
	1	Disables hardware .
RES0, [14]	<i>res0</i>	Reserved.
PF_SS_L2_DIST, [13:12]		Single cache line stride L2 distance. The possible values are:
	00	22
	01	28
	10	34
	11	40. This is the reset value.
RES0, [11:10]	<i>res0</i>	Reserved.
RES0, [9]	<i>res0</i>	Reserved.
PF_STI_DIS, [8]		Disables store prefetches at issue (not overridden by CPUECTLR_EL1[15]). The possible values are:
	0	Enables store . This is the reset value.
	1	Disables store .
PF_STS_DIS, [7]		Disables store-stride prefetches. The possible values are:
	0	Enables store . This is the reset value.
	1	Disables store .
RES0, [6]	<i>res0</i>	Reserved.
RPF_DIS, [5]		Disables <i>region</i> prefetcher. The possible values are:
	0	Enables . This is the reset value.
	1	Disables .

RPF_LO_CONF, [4]

Region prefetcher training behavior. The possible values are:

- | | |
|----------|---|
| 0 | Limited training for prefetcher on single accesses. This is the reset value. |
| 1 | Always train the prefetcher on single accesses, which results in fewer prefetch requests. |

RPF_PHIT_EN, [3]

Enable prefetcher propagation on hit. The possible values are:

- | | |
|----------|--|
| 0 | Disables prefetcher propagation on hit. This is the reset value. |
| 1 | Enables prefetcher propagation on hit. |

RES0, [2:1]

res0 Reserved.

EXTLLC, [0]

Internal or external Last-level cache (LLC) in the system. The possible values are:

- | | |
|----------|--|
| 0 | Indicates that an internal Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count. This is the reset value. |
| 1 | Indicates that an external Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count. |

Configurations

This register has no configuration options.

Usage constraints

Accessing the CPUECTLR_EL1

The CPU Extended Control Register can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the is enabled.

This register can be using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
CPUECTLR_EL1	11	000	1111	0001	100

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CPUECTLR_EL1	x	x	0	-	RW	n/a	RW
CPUECTLR_EL1	x	0	1	-	RW	RW	RW
CPUECTLR_EL1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The [PE](#) cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated s, see *Synchronous prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to [AArch64](#) state.

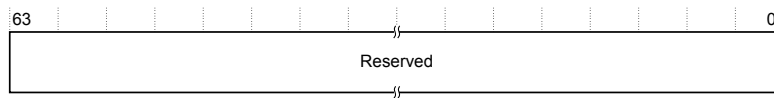
Access to this register depends on bit[1] of ACTLR_EL2 and ACTLR_EL3.

CPUPCR_EL3, CPU Private Control Register, EL3

The CPUPCR_EL3 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUPCR_EL3 is a 64-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

**Figure 30: CPUPCR_EL3 bit assignments****Reserved, [63:0]**

Reserved for Arm® internal use.

Configurations

CPUPCR_EL3 is only accessible in Secure state.

Usage constraints**Accessing the CPUPCR_EL3**

The CPUPCR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the *MMU* is enabled.

Writing to this register might cause *UNPREDICTABLE* behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be *read* with the MRS instruction using the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_1	11	110	1111	1000	001

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_1	x	x	0	-	-	n/a	RW
S3_6_C15_8_1	x	0	1	-	-	-	RW
S3_6_C15_8_1	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The *core* cannot be executing at this *Exception level*, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Synchronous prioritization* in the *Arm®*

CPUPMR_EL3, CPU Private Mask Register, EL3

The CPUPMR_EL3 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUPMR_EL3 is a 64-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

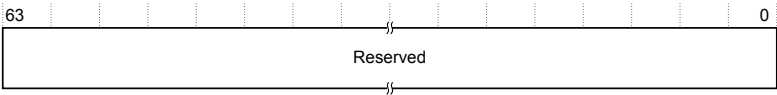


Figure 31: CPUPMR_EL3 bit assignments

Reserved, [63:0]

Reserved for Arm[®] internal use.

Configurations

CPUPMR_EL3 is only accessible in Secure state.

Usage constraints

Accessing the CPUPMR_EL3

The CPUPMR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the *MMU* is enabled.

Writing to this register might cause *UNPREDICTABLE* behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be *read* with the MRS instruction using the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_3	11	110	1111	1000	011

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_3	x	x	0	-	-	n/a	RW
S3_6_C15_8_3	x	0	1	-	-	-	RW
S3_6_C15_8_3	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The *core* cannot be executing at this *Exception level*, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Synchronous prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CPUPOR_EL3, CPU Private Operation Register, EL3

The CPUPOR_EL3 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUPOR_EL3 is a 64-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

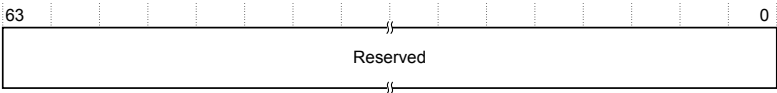


Figure 32: CPUPOR_EL3 bit assignments

Reserved, [63:0]

Reserved for Arm® internal use.

Configurations

CPUPOR_EL3 is only accessible in Secure state.

Usage constraints

Accessing the CPUPOR_EL3

The CPUPOR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the *MMU* is enabled.

Writing to this register might cause *UNPREDICTABLE* behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be *read* with the MRS instruction using the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_2	11	110	1111	1000	010

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_2	x	x	0	-	-	n/a	RW
S3_6_C15_8_2	x	0	1	-	-	-	RW
S3_6_C15_8_2	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The *core* cannot be executing at this *Exception level*, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Synchronous prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CPUPSELR_EL3, CPU Private Selection Register, EL3

The CPUPSELR_EL3 provides *IMPLEMENTATION DEFINED* configuration and control options for the core.

Bit field descriptions

CPUPSELR_EL3 is a 32-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

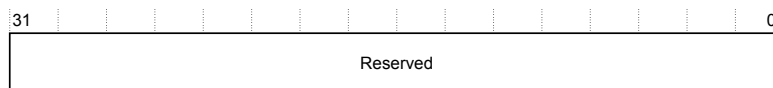


Figure 33: CPUPSELR_EL3 bit assignments

Reserved, [31:0]

Reserved for Arm® internal use.

Configurations

CPUPSELR_EL3 is only accessible in Secure state.

Usage constraints

Accessing the CPUPSELR_EL3

The CPUPSELR_EL3 can be written only when the system is idle. Arm recommends that you write to

this register after a powerup reset, before the [MMU](#) is enabled.

Writing to this register might cause *UNPREDICTABLE* behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be [read](#) with the MRS instruction using the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_0	11	110	1111	1000	000

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_0	x	x	0	-	-	n/a	RW
S3_6_C15_8_0	x	0	1	-	-	-	RW
S3_6_C15_8_0	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The [core](#) cannot be executing at this [Exception level](#), so this access is not possible.

Traps and enables

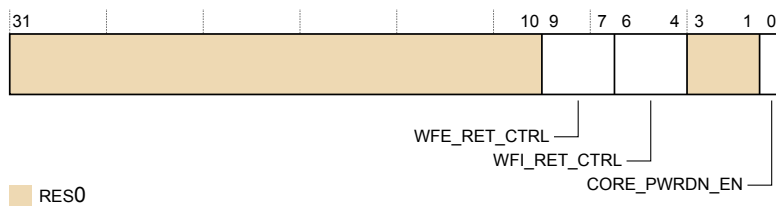
For a description of the prioritization of any generated [exceptions](#), see *Synchronous prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CPUPWRCTLR_EL1, Power Control Register, EL1

The CPUPWRCTLR_EL1 provides information about power control support for the core.

Bit field descriptions

CPUPWRCTLR_EL1 is a 32-bit register, and is part of the *IMPLEMENTATION DEFINED* registers functional group.

**Figure 34: CPUPWRCTLR_EL1 bit assignments**

RES0, [31:10]	<i>res0</i>	Reserved.
WFE_RET_CTRL, [9:7]	CPU WFE retention control:	
	000	Disable the retention circuit. This is the default value, see Table 1 for more retention control options.
WFI_RET_CTRL, [6:4]	CPU WFI retention control:	
	000	Disable the retention circuit. This is the default value, see Table 1 for more retention control options.
RES0, [3:1]	<i>res0</i>	Reserved.
CORE_PWRDN_EN, [0]	Indicates to the power controller using <i>PACTIVE</i> if the <i>core</i> wants to power down when it enters WFI state.	
	0	No power down requested. This is the reset value.
	1	A power down is requested.

Table 52: CPUPWRCTLR Retention Control Field

Encoding	Number of counter ticks ³	Minimum retention entry delay (System counter at 50MHz-10MHz)
000	Disable the retention circuit	Default Condition.
001	2	40ns-200ns
010	8	160ns-800ns
011	32	640ns – 3,200ns
100	64	1,280ns-6,400ns

³ The number of system counter ticks required before the signals retention *readiness* on CTIVE to the power controller. The does not accept a retention entry request until this time.

Encoding	Number of counter ticks ³	Minimum retention entry delay (System counter at 50MHz-10MHz)
101	128	2,560ns-12,800ns
110	256	5,120ns-25,600ns
111	512	10,240ns-51,200ns

Configurations

There are no configuration notes.

Usage constraints

Accessing the CPUPWRCTLR_EL1

This register can be using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_7	11	000	1111	0010	111

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_7	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_7	x	0	1	-	RW	RW	RW
S3_0_C15_C2_7	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The *PE* cannot be executing at this *Exception level*, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Synchronous prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to *AArch64* state.

Write access to this register from EL1 or EL2 depends on the value of bit[7] of ACTLR_EL2 and ACTLR_EL3.

³ The number of system counter ticks required before the signals retention *readiness* on CTIVE to the power controller. The does not accept a retention entry request until this time.

CSSELR_EL1, Cache Size Selection Register, EL1

CSSELR_EL1 selects the current Cache Size ID Register (CCSIDR_EL1), by specifying:

Bit field descriptions

- The required cache level.
- The cache type, either instruction or data cache.

For details of the CCSIDR_EL1, see [CCSIDR_EL1, Cache Size ID Register, EL1](#).

CSSELR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

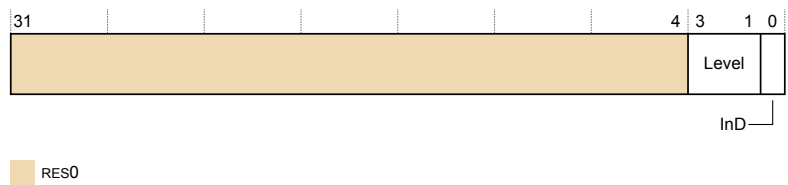


Figure 35: CSSELR_EL1 bit assignments

[31:4]	<i>res0</i>	Reserved.
Level, [3:1]	Cache level of required cache:	
	000	L1.
	001	L2.
	010	L3, if present.
	The combination of Level=001 and InD=1 is reserved.	
	The combinations of Level and InD for 0100 to 1111 are reserved.	
InD, [0]	Instruction not Data bit:	
	0	Data or unified cache.
	1	Instruction cache.
	The combination of Level=001 and InD=1 is reserved.	
	The combinations of Level and InD for 0100 to 1111 are reserved.	
Configurations	If a cache level is missing but CSSELR_EL1 selects this level, then a CCSIDR_EL1 <i>read</i> returns an <i>UNKNOWN</i> value.	
	Bit fields and details not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

DISR_EL1, Deferred Interrupt Status Register, EL1

The DISR_EL1 records the SError interrupts consumed by an ESB instruction.

Bit field descriptions

DISR_EL1 is a 64-bit register, and is part of the registers *Reliability, Availability, Serviceability* (RAS) functional group.

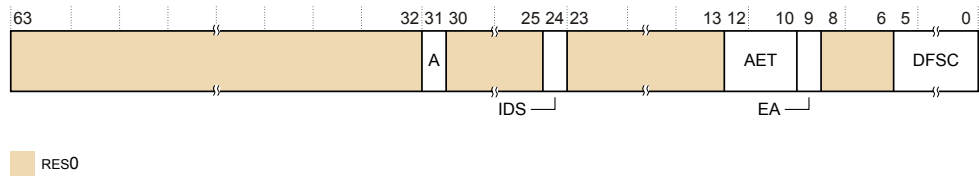


Figure 36: DISR_EL1 bit assignments, DISR_EL1.IDS is 0

RES0, [63:32]	Reserved, <i>RES0</i> .
A, [31]	Set to 1 when ESB defers an asynchronous SError interrupt. If the implementation does not include any synchronizable sources of SError interrupt, this bit is <i>res0</i> .
RES0, [30:25]	Reserved, <i>RES0</i> .
IDS, [24]	Indicates the type of format the deferred SError interrupt uses. The value of this bit is: 0 Deferred error uses architecturally-defined format.
RES0, [23:13]	Reserved, <i>RES0</i> .
AET, [12:10]	Asynchronous Error Type. Describes the state of the <i>core</i> after taking an asynchronous <i>Data Abort exception</i> . The possible values are: 000 Uncontainable error (UC). 001 Unrecoverable error (UEU).
EA, [9]	Reserved, <i>RES0</i> .
RES0, [8:6]	Reserved, <i>RES0</i> .
DFSC, [5:0]	Data Fault Status Code. The possible values of this field are:



Note:
The recovery software must also examine any implemented *fault* records to determine the location and extent of the error.

010001	Asynchronous SError interrupt.
	<div>Note: In <i>AArch32</i> the 010001 code previously meant an Asynchronous External Abort on memory access. With the RAS extension, it extends to include any asynchronous SError interrupt. The Parity Error codes are not used in the RAS extension.</div>

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ERRIDR_EL1, Error ID Register, EL1

The ERRIDR_EL1 defines the number of error record registers.

Bit field descriptions

ERRIDR_EL1 is a 32-bit register, and is part of the registers *Reliability, Availability, Serviceability* (RAS) functional group.

This register is Read Only.

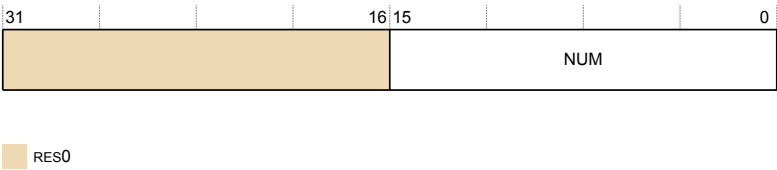


Figure 37: ERRIDR_EL1 bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
NUM, [15:0]		Number of records that can be accessed through the Error Record system registers.
	0002	Two records present, if L3 cache is present.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ERRSELR_EL1, Error Record Select Register, EL1

The ERRSELR_EL1 selects which error record should be accessed through the Error Record system registers. This register is not reset on a warm reset.

Bit field descriptions

ERRSELR_EL1 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

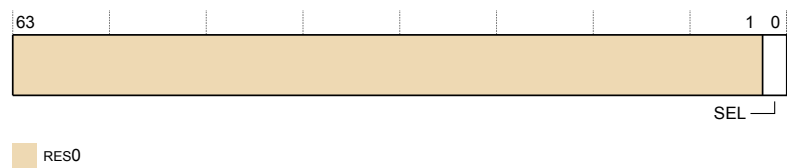


Figure 38: ERRSELR_EL1 bit assignments

RES0, [63:1]	Reserved, <i>RES0</i> .
SEL, [0]	Selects which error record should be accessed.
0	Select record 0 containing errors from Level 1 and Level 2 RAMs located on the Cortex®-A76 <i>core</i> .
1	Select record 1 containing errors from Level 3 RAMs located on the DSU.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ERXADDR_EL1, Selected Error Record Address Register, EL1

Register ERXADDR_EL1 accesses the ERR<n>ADDR address register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXADDR_EL1 accesses the ERR0ADDR register of the core error record. See [ERR0ADDR, Error Record Address Register](#).

If `ERRSELR_EL1.SEL==1`, then `ERXADDR_EL1` accesses the `ERR1ADDR` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ERXCTLR_EL1, Selected Error Record Control Register, EL1

Register `ERXCTLR_EL1` accesses the `ERR<n>CTLR` control register for the error record selected by `ERRSELR_EL1.SEL`.

If `ERRSELR_EL1.SEL==0`, then `ERXCTLR_EL1` accesses the `ERR0CTLR` register of the *core* error record. See [ERR0CTLR, Error Record Control Register](#).

If `ERRSELR_EL1.SEL==1`, then `ERXCTLR_EL1` accesses the `ERR1CTLR` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ERXFR_EL1, Selected Error Record Feature Register, EL1

Register `ERXFR_EL1` accesses the `ERR<n>FR` feature register for the error record selected by `ERRSELR_EL1.SEL`.

If `ERRSELR_EL1.SEL==0`, then `ERXFR_EL1` accesses the `ERR0FR` register of the core error record. See [ERR0FR, Error Record Feature Register](#).

If `ERRSELR_EL1.SEL==1`, then `ERXFR_EL1` accesses the `ERR1FR` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1

Register `ERXMISC0_EL1` accesses the `ERR<n>MISC0` register for the error record selected by `ERRSELR_EL1.SEL`.

If `ERRSELR_EL1.SEL==0`, then `ERXMISC0_EL1` accesses the `ERR0MISC0` register of the *core* error record. See [ERR0MISC0, Error Record Miscellaneous Register 0](#).

If `ERRSELR_EL1.SEL==1`, then `ERXMISC0_EL1` accesses the `ERR1MISC0` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1

Register `ERXMISC1_EL1` accesses the `ERR<n>MISC1` miscellaneous register 1 for the error record selected by `ERRSELR_EL1.SEL`.

If `ERRSELR_EL1.SEL==0`, then `ERXMISC1_EL1` accesses the `ERR0MISC1` register of the *core* error record. See [ERR0MISC1, Error Record Miscellaneous Register 1](#).

If `ERRSELR_EL1.SEL==1`, then `ERXMISC1_EL1` accesses the `ERR1MISC1` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1

Register `ERXPFGCDN_EL1` accesses the `ERR<n>PFGCDN` register for the error record selected by `ERRSELR_EL1.SEL`.

If `ERRSELR_EL1.SEL==0`, then `ERXPFGCDN_EL1` accesses the `ERR0PFGCDN` register of the *core* error record. See [ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register](#).

If `ERRSELR_EL1.SEL==1`, then `ERXPFPGCDN_EL1` accesses the `ERR1PFGCDNR` register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

There are no configuration notes.

Accessing the `ERXPFPGCDN_EL1`

This register can be *read* using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <Xt>, <systemreg>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_2	11	000	1111	0010	010

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_2	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_2	x	0	1	-	RW	RW	RW
S3_0_C15_C2_2	x	1	1	-	n/a	RW	RW
n/a	Not accessible. Executing the <i>PE</i> at this <i>Exception level</i> is not permitted.						

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to *AArch32* state, and see *Synchronous prioritization* for s taken to *AArch64* state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

`ERXPFPGCDN_EL1` is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in `ACTLR_EL2` and `ACTLR_EL3`. See [ACTLR_EL2, Auxiliary Control Register, EL2](#) and [ACTLR_EL3, Auxiliary Control Register, EL3](#).

`ERXPFPGCDN_EL1` is *UNDEFINED* at EL0.

ERXPFPGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1

Register ERXPFPGCTL_EL1 accesses the ERR<n>PFGCTL register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXPFPGCTL_EL1 accesses the ERR0PFGCTLR register of the [core](#) error record. See [ERR0PFGCTL, Error Pseudo Fault Generation Control Register](#).

If ERRSELR_EL1.SEL==1, then ERXPFPGCTL_EL1 accesses the ERR1PFGCTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

There are no configuration notes.

Accessing the ERXPFPGCTL_EL1

This register can be [read](#) using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <Xt>, <systemreg>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_1	11	000	1111	0010	001

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_1	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_1	x	0	1	-	RW	RW	RW
S3_0_C15_C2_1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The [PE](#) cannot be executing at this [Exception level](#), so this access is not possible.

Traps and enables

For a description of the prioritization of any generated [exceptions](#), see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to [AArch32](#) state, and see *Synchronous prioritization* for s taken to [AArch64](#) state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFPGCTL_EL1 is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR_EL2 and ACTLR_EL3. See [ACTLR_EL2, Auxiliary Control Register, EL2](#) and [ACTLR_EL3, Auxiliary Control Register, EL3](#).

ERXPFGCTL_EL1 is *UNDEFINED* at EL0.

If ERXPFGCTL_EL1 is accessible at EL1 and HCR_EL2.TERR == 1, then direct s and writes of ERXPFGCTL_EL1 at Non-secure EL1 generate a Trap to EL2.

If ERXPFGCTL_EL1 is accessible at EL1 or EL2 and SCR_EL3.TERR == 1, then direct s and writes of ERXPFGCTL_EL1 at EL1 or EL2 generate a Trap to EL3.

ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1

Register ERXPFGF_EL1 accesses the ERR<n>PFGF register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXPFGF_EL1 accesses the ERR0PFGF register of the *core* error record. See [ERR0PFGF, Error Pseudo Fault Generation Feature Register](#).

If ERRSELR_EL1.SEL==1, then ERXPFGF_EL1 accesses the ERR1PFGFR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

This has no configuration notes.

Accessing the ERXPFGF_EL1

This register can be *read* using MRS with the following syntax:

```
MRS <Xt>, <systemreg>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_0	11	000	1111	0010	000

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_0	x	x	0	-	RO	n/a	RO
S3_0_C15_C2_0	x	0	1	-	RO	RO	RO
S3_0_C15_C2_0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The *PE* cannot be executing at this *Exception level*, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated *exceptions*, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for s taken to *AArch32* state, and see *Synchronous prioritization* for s taken to *AArch64* state.

Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGR_EL1 is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR_EL2 and ACTLR_EL3. See [ACTLR_EL2, Auxiliary Control Register, EL2](#) and [ACTLR_EL3, Auxiliary Control Register, EL3](#).

ERXPFGR_EL1 is *UNDEFINED* at EL0.

If ERXPFGR_EL1 is accessible at EL1 and HCR_EL2.TERR == 1, then direct s and writes of ERXPFGR_EL1 at Non-secure EL1 generate a Trap to EL2.

If ERXPFGR_EL1 is accessible at EL1 or EL2 and SCR_EL3.TERR == 1, then direct s and writes of ERXPFGR_EL1 at EL1 or EL2 generate a Trap to EL3.

ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1

Register ERXSTATUS_EL1 accesses the ERR<n>STATUS primary status register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXSTATUS_EL1 accesses the ERR0STATUS register of the *core* error record. See [ERR0STATUS, Error Record Primary Status Register](#).

If ERRSELR_EL1.SEL==1, then ERXSTATUS_EL1 accesses the ERR1STATUS register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

ESR_EL1, Exception Syndrome Register, EL1

The ESR_EL1 holds syndrome information for an exception taken to EL1.

Bit field descriptions

ESR_EL1 is a 32-bit register, and is part of the Exception and *fault* handling registers functional group.

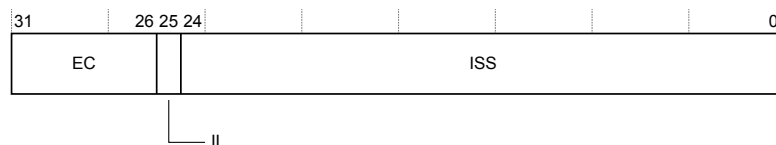


Figure 39: ESR_EL1 bit assignments

EC, [31:26]

Exception Class. Indicates the reason for the *exception* that this register holds information about.

IL, [25]

Instruction Length for synchronous s. The possible values are:

0	16-bit.
1	32-bit.

This field is 1 for the SError interrupt, instruction *aborts*, misaligned PC, Stack pointer misalignment, data s for

When reporting a physical SEI, the following occurs:

- $IDS==0$ (architectural syndrome).
- AET always reports an uncontainable error (UC) with value 000 or an unrecoverable error (UEU) with value 001.
- EA is *RES0*.

When reporting a synchronous *Data Abort*, EA is *RES0*.

See [VSESR_EL2, Virtual SError Exception Syndrome Register \(Ares Specific\)](#).

Configurations

RW fields in this register reset to architecturally *UNKNOWN* values.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ESR_EL3, Exception Syndrome Register, EL3

The ESR_EL3 holds syndrome information for an exception taken to EL3.

Bit field descriptions

ESR_EL3 is a 32-bit register, and is part of the Exception and *fault* handling registers functional group.

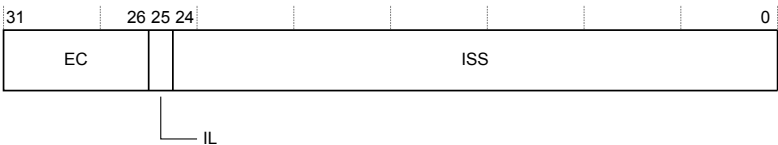


Figure 41: ESR_EL3 bit assignments

EC, [31:26]

Exception Class. Indicates the reason for the *exception* that this register holds information about.

IL, [25]

Instruction Length for synchronous s. The possible values are:

0	16-bit.
1	32-bit.

This field is 1 for the SError interrupt, instruction *aborts*, misaligned PC, Stack pointer misalignment, data s for which the ISV bit is 0, s caused by an *illegal instruction* set state, and s using the 0 Exception Class.

ISS, [24:0]

Syndrome information.

When reporting a virtual SEI, bits[24:0] take the value of VSESR_EL2[24:0].

When reporting a physical SEI, the following occurs:

- $IDS==0$ (architectural syndrome).

- AET always reports an uncontainable error (UC) with value 000 or an unrecoverable error (UEU) with value 001.
- EA is *RES0*.

When reporting a synchronous data , EA is *RES0*.

See [VSESR_EL2, Virtual SError Exception Syndrome Register \(Ares Specific\)](#).

Configurations

RW fields in this register reset to architecturally *unknown* values.

HACR_EL2, Hyp Auxiliary Configuration Register, EL2

HACR_EL2 controls trapping to EL2 of *IMPLEMENTATION DEFINED* aspects of Non-secure EL1 or EL0 operation. This register is not used in the Cortex[®]-A76 core.

Bit field descriptions

HACR_EL2 is a 32-bit register, and is part of Virtualization registers functional group.

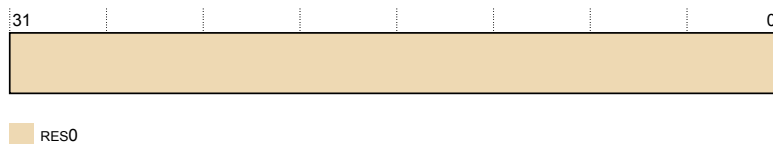


Figure 42: HACR_EL2 bit assignments

RES0, [31:0]

Reserved, *RES0*.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

HCR_EL2, Hypervisor Configuration Register, EL2

The HCR_EL2 provides configuration control for virtualization, including whether various Non-secure operations are trapped to EL2.

Bit field descriptions

HCR_EL2 is a 64-bit register, and is part of the Virtualization registers functional group.

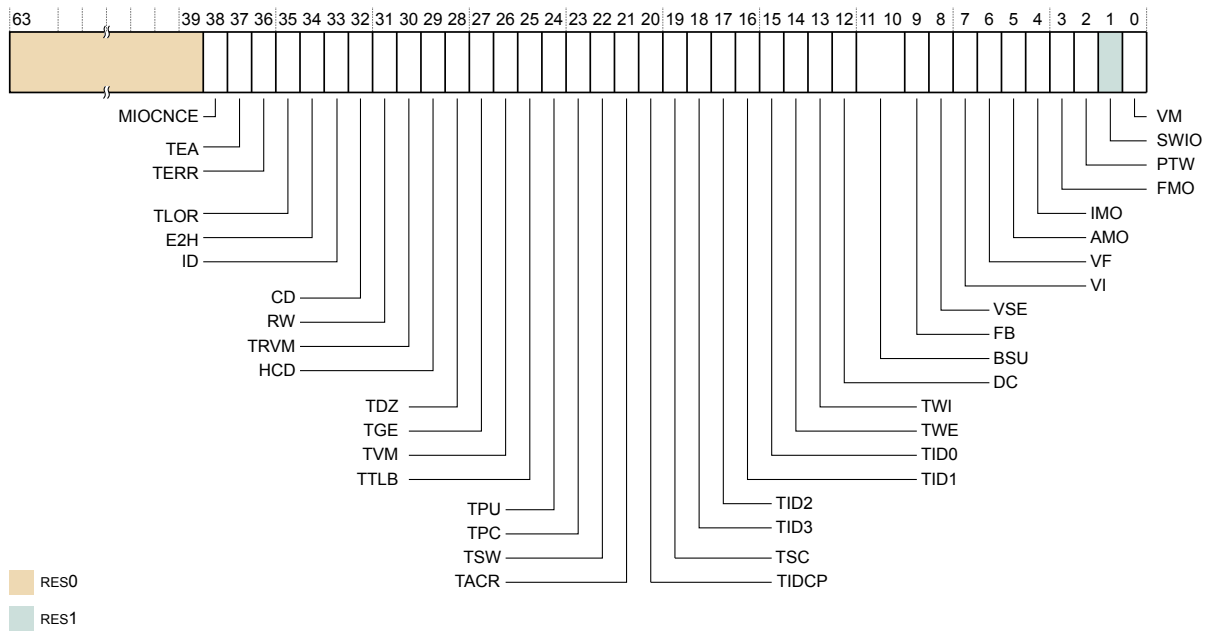


Figure 43: HCR_EL2 bit assignments

RES0, [63:39]

res0

Reserved.

MIOCNCNCE, [38]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the Non-secure EL1 and EL0 translation regime.

RW, [31]

res1

Reserved.

HCD, [29]

res0

Reserved.

TGE, [27]

Traps general [exceptions](#). If this bit is set, and SCR_EL3.NS is set, then:

- All s that would be routed to EL1 are routed to EL2.
- The SCTLRL_EL1.M bit is treated as 0 regardless of its actual state, other than for [reading](#) the bit.
- The HCR_EL2.FMO, IMO, and AMO bits are treated as 1 regardless of their actual state, other than for [ing](#) the bits.
- All virtual interrupts are disabled.
- Any *IMPLEMENTATION DEFINED* mechanisms for signaling virtual interrupts are disabled.
- An return to EL1 is treated as an illegal return.

HCR_EL2.TGE must not be cached in a [TLB](#).

When the value of SCR_EL3.NS is 0 the [core](#) behaves as if this field is 0 for all purposes other than a direct or write access of HCR_EL2.

TID3, [18]

Traps ID group 3 registers. The possible values are:

0	ID group 3 register accesses are not trapped.
1	Reads to ID group 3 registers executed from Non-secure EL1 are trapped to EL2.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for the registers covered by this setting.

Configurations

If EL2 is not implemented, this register is *RES0* from EL3

RW fields in this register reset to architecturally *UNKNOWN* values.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The core does not use this register, ID_AA64AFR0_EL1 is *RES0*.

ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The core does not use this register, ID_AA64AFR0_EL1 is *RES0*.

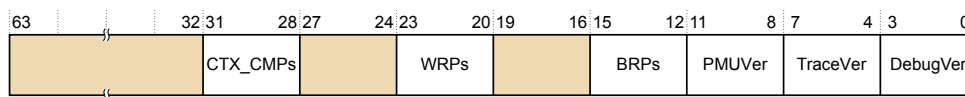
ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1

Provides top-level information about the debug system in AArch64.

Bit field descriptions

ID_AA64DFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.



RES0

Figure 44: ID_AA64DFR0_EL1 bit assignments

RES0, [63:32]

RES0

Reserved.

CTX_CMPs, [31:28]

Number of *breakpoints* that are context-aware, minus 1. These are the highest numbered s:

	1	Two s are context-aware.
RES0, [27:24]	RES0	Reserved.
WRPs, [23:20]	The number of <i>watchpoints</i> minus 1:	
	3	Four points.
RES0, [19:16]	RES0	Reserved.
BRPs, [15:12]	The number of s minus 1:	
	5	Six s.
PMUVer, [11:8]	Performance Monitors Extension version.	
	4	Performance monitor system registers implemented, PMUv3.
TraceVer, [7:4]	Trace extension:	
	0	Trace system registers not implemented.
DebugVer, [3:0]	Debug architecture version:	
	8	Arm®v8-A debug architecture implemented.
Configurations	<p>ID_AA64DFR0_EL1 is architecturally mapped to external register EDDFR.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>	

ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1, EL1

This register is reserved for future expansion of top level information about the debug system in AArch64 state.

ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1

The ID_AA64ISAR0_EL1 provides information about the instructions implemented in AArch64 state, including the instructions that are provided by the Cryptographic Extension.

Bit field descriptions

ID_AA64ISAR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

The optional Cryptographic Extension is not included in the base product of the *core*. Arm requires licensees to have contractual rights to obtain the Cryptographic Extension.

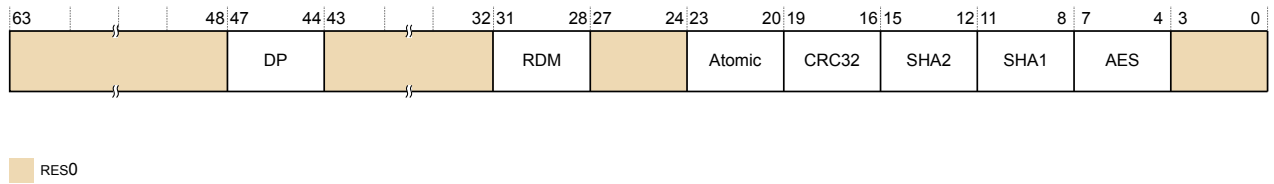


Figure 45: ID_AA64ISAR0_EL1 bit assignments

RES0, [63:48]	<i>RES0</i>	Reserved.
DP, [47:44]		Indicates whether Dot Product support instructions are implemented.
	1	UDOT, SDOT instructions are implemented.
RES0, [43:32]	<i>RES0</i>	Reserved.
RDM, [31:28]		Indicates whether SQRDMLAH and SQRDMLSH instructions in <i>AArch64</i> are implemented.
	1	SQRDMLAH and SQRDMLSH instructions implemented.
RES0, [27:24]	<i>RES0</i>	Reserved.
Atomic, [23:20]		Indicates whether Atomic instructions in are implemented. The value is:
	2	LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions are implemented.
		.
CRC32, [19:16]		Indicates whether CRC32 instructions are implemented. The value is:
	1	CRC32 instructions are implemented.
SHA2, [15:12]		Indicates whether SHA2 instructions are implemented. The possible values are:
	0	No SHA2 instructions are implemented. This is the value if the

		implementation does not include the Cryptographic Extension.
	1	SHA256H, SHA256H2, SHA256U0, and SHA256U1 implemented. This is the value if the implementation includes the Cryptographic Extension.
SHA1, [11:8]	Indicates whether SHA1 instructions are implemented. The possible values are:	
	0	No SHA1 instructions implemented. This is the value if the implementation does not include the Cryptographic Extension.
	1	SHA1C, SHA1P, SHA1M, SHA1SU0, and SHA1SU1 implemented. This is the value if the implementation includes the Cryptographic Extension.
AES, [7:4]	Indicates whether AES instructions are implemented. The possible values are:	
	0	No AES instructions implemented. This is the value if the implementation does not include the Cryptographic Extension.
	2	AESE, AESD, AESMC, and AESIMC implemented, plus PMULL and PMULL2 instructions operating on 64-bit data. This is the value if the implementation includes the Cryptographic Extension.
[3:0]	Reserved, <i>RES0</i> .	
Configurations	<p>ID_AA64ISAR0_EL1 is architecturally mapped to external register ID_AA64ISAR0.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm</i>®</p>	

ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1

The ID_AA64ISAR1_EL1 provides information about the instructions implemented in AArch64 state.

Bit field descriptions

ID_AA64ISAR1_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

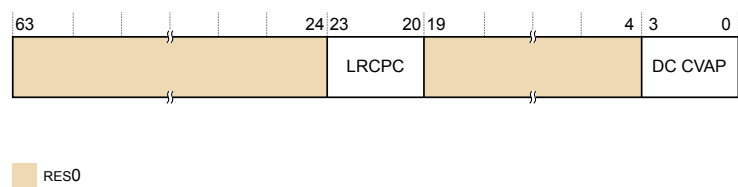


Figure 46: ID_AA64ISAR1_EL1 bit assignments

RES0, [63:24]	RES0	Reserved.
LRCPC, [23:20]	Indicates whether load-acquire (LDA) instructions are implemented for a Release Consistent <i>core</i> consistent RCPC model.	1 The LDAPRB, LDAPRH, and LDAPR instructions are implemented in <i>AArch64</i> .
RES0, [19:4]	RES0	Reserved.
DC CVAP, [3:0]	Indicates whether Data Cache, Clean to the Point of Persistence (DC CVAP) instructions are implemented.	1 DC CVAP is supported in .
Configurations	There are no configuration notes. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

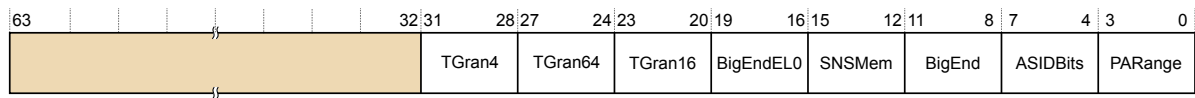
ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1

The ID_AA64MMFR0_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.



RES0

Figure 47: ID_AA64MMFR0_EL1 bit assignments

RES0, [63:32]	<i>RES0</i>	Reserved.
TGran4, [31:28]	Support for 4KB memory translation granule size:	
	0	4KB granule supported.
TGran64, [27:24]	Support for 64KB memory translation granule size:	
	0	64KB granule supported.
TGran16, [23:20]	Support for 16KB memory translation granule size:	
	1	Indicates that the 16KB granule is supported.
BigEndEL0, [19:16]	Mixed-endian support only at EL0.	
	0	No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value.
SNSMem, [15:12]	Secure versus Non-secure Memory distinction:	
	1	Supports a distinction between Secure and Non-secure Memory.
BigEnd, [11:8]	Mixed-endian configuration support:	
	1	Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured.
ASIDBits, [7:4]	Number of ASID bits:	

	2	16 bits.
PARange, [3:0]	Physical address range supported:	
	2	40 bits, 1TB.
	The supported Physical Address Range is 40-bits. Other <i>cores</i> in the DSU may support a different Physical Address Range.	

Configurations	There are no configuration notes.	
	Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1

The ID_AA64MMFR1_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR1_EL1 is a 64-bit register, and is part of the Identification registers functional group. This register is Read Only.

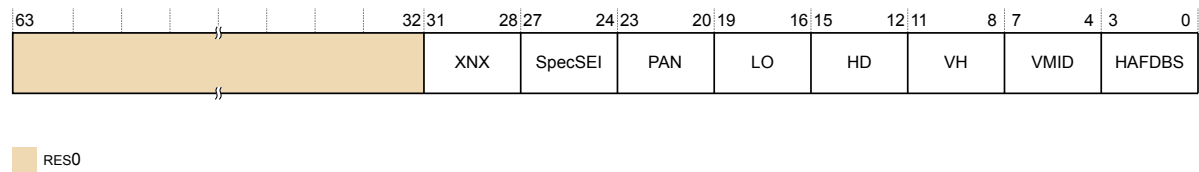


Figure 48: ID_AA64MMFR1_EL1 bit assignments

RES0, [63:32]	RES0	Reserved.
XNX, [31:28]	1	Indicates whether provision of EL0 vs EL1 execute never control at Stage 2 is supported. EL0/EL1 execute control distinction at Stage 2 bit is supported. All other values are reserved.
SpecSEI, [27:24]	0	Describes whether the <i>PE</i> can generate SError interrupt <i>exceptions</i> from speculative <i>reads</i> of memory, including speculative instruction fetches. The never generates an SError interrupt due to

an external *abort* on a speculative .

PAN, [23:20]

Privileged Access Never. Indicates support for the N bit in PSTATE, *SPSR_EL1*, *SR_EL2*, *SR_EL3*, and *DSR_EL0*.

2

N supported and AT S1E1RP and AT S1E1WP instructions supported.

LO, [19:16]

Indicates support for LORegions.

1

LORegions are supported.

HD, [15:12]

Presence of Hierarchical Disables. Enables an operating system or hypervisor to hand over up to 4 bits of the last level page table descriptor (bits[62:59] of the *page table* entry) for use by hardware for *IMPLEMENTATION DEFINED* usage. The value is:

2

Hierarchical Permission Disables and Hardware allocation of bits[62:59] supported.

VH, [11:8]

Indicates whether Virtualization Host Extensions are supported.

1

Virtualization Host Extensions supported.

VMID, [7:4]

Indicates the number of VMID bits supported.

2

16 bits are supported.

HAFDBS, [3:0]

Indicates the support for hardware updates to Access flag and *dirty* state in *translation tables*.

2

Hardware update of both the Access flag and state is supported in hardware.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1

The ID_AA64MMFR2_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR2_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

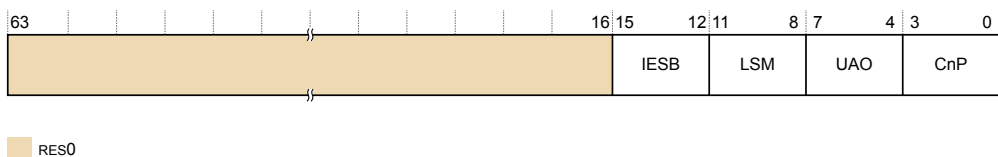


Figure 49: ID_AA64MMFR2_EL1 bit assignments

RES0, [63:16]

RES0

Reserved.

IESB, [15:12]

Indicates whether an implicit Error Synchronization Barrier has been inserted. The value is:

1

SCTLR_ELx.IESB implicit ErrorSynchronizationBarrier control implemented.

LSM, [11:8]

Indicates whether LDM and *STM* ordering control bits are supported. The value is:

0

LSMAOE and nTLSMD bit not supported.

UAO, [7:4]

Indicates the presence of the *User Access Override* (UAO). The value is:

1

UAO is supported.

CnP, [3:0]

Common not Private. Indicates whether a *TLB* entry is pointed at a *translation table base register* that is a member of a common set. The value is:

1

CnP bit is supported.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1

The ID_AA64PFR0_EL1 provides additional information about implemented core features in AArch64.

Bit field descriptions

The optional *Advanced SIMD* and *floating-point* support is not included in the base product of the *core*. Arm requires licensees to have contractual rights to obtain the and support.

ID_AA64PFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

63	60	59	56	55	32	31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0																		
CSV3				CSV2				RAS				GIC				AdvSIMD				FP				EL3 handling				EL2 handling				EL1 handling				EL0 handling			

		implementation has ECC present.
GIC, [27:24]	GIC CPU interface:	
	0	GIC CPU interface is disabled, GICCDISABLE is HIGH, or not implemented.
	1	GIC CPU interface is implemented and enabled, GICCDISABLE is LOW.
Adv, [23:20]	. The possible values are:	
	1	, including Half-precision support, is implemented.
FP, [19:16]	Floating-point. The possible values are:	
	1	Floating-point, including Half-precision support, is implemented.
EL3 handling, [15:12]	EL3 <i>exception</i> handling:	
	1	Instructions can be executed at EL3 in <i>AArch64</i> state only.
EL2 handling, [11:8]	EL2 handling:	
	1	Instructions can be executed at EL3 in state only.
EL1 handling, [7:4]	EL1 handling. The possible values are:	
	1	Instructions can be executed at EL3 in state only.
EL0 handling, [3:0]	EL0 handling. The possible values are:	
	2	Instructions can be executed at EL0 in or <i>AArch32</i> state.
Configurations	ID_AA64PFR0_EL1 is architecturally mapped to External register EDPFR. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1, EL1

The ID_AA64PFR1_EL1 provides additional information about implemented core features in AArch64.

Bit field descriptions

ID_AA64PFR1_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.



RES0

Figure 51: ID_AA64PFR1_EL1 bit assignments

RES0, [63:8]

RES0

Reserved.

SSBS, [7:4]

PSTATE.SSBS. The possible values are:

1

AArch64 provides the PSTATE.SSBS mechanism to mark *regions* that are *Speculative Store Bypassing Safe* (SSBS), but does not implement the MSR/MRS instructions to directly *read* and write the PSTATE.SSBS field.

RES0, [3:0]

RES0

Reserved.

Configurations

ID_AA64PFR1_EL1 is architecturally mapped to External register EDPFR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1

The ID_AFR0_EL1 provides information about the *IMPLEMENTATION DEFINED* features of the PE in AArch32. This register is not used in the Cortex®-A76 core.

Bit field descriptions

ID_AFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

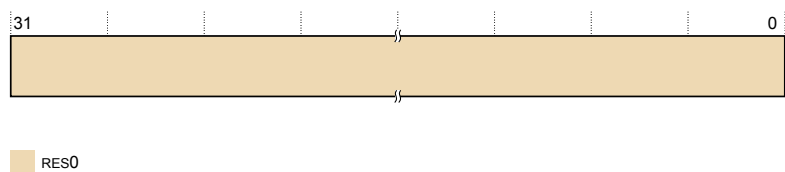


Figure 52: ID_AFR0_EL1 bit assignments

RES0, [31:0] Reserved, *RES0*.

Configurations There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1

The ID_DFR0_EL1 provides top-level information about the debug system in AArch32.

Bit field descriptions

ID_DFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

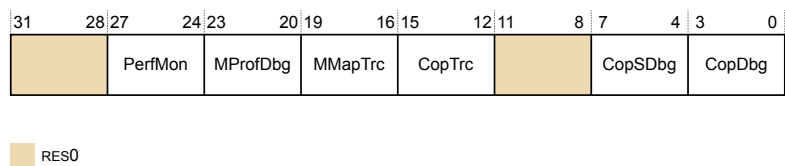


Figure 53: ID_DFR0_EL1 bit assignments

RES0, [31:28] *RES0* Reserved.

PerfMon, [27:24] Indicates support for performance monitor model:
4 Support for *Performance Monitor Unit version 3* (PMUv3) system registers, with a 16-bit evtCount field.

MProfDbg, [23:20] Indicates support for memory-mapped debug model for M profile *cores*:
0 This product does not support M profile Debug architecture.

MMapTrc, [19:16] Indicates support for memory-mapped trace model:

	1	Support for Arm trace architecture, with memory-mapped access.
	In the Trace registers, the <i>ETMIDR</i> gives more information about the implementation.	
CopTrc, [15:12]	Indicates support for <i>coprocessor</i> -based trace model:	
	0	This product does not support Arm trace architecture.
RES0, [11:8]	<i>RES0</i>	Reserved.
CopSDBG, [7:4]	Indicates support for -based Secure debug model:	
	8	This product supports the Armv8.2 Debug architecture.
CopDBG, [3:0]	Indicates support for -based debug model:	
	8	This product supports the Armv8.2 Debug architecture.

Configurations There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1

The ID_ISAR0_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR0_EL1 is a 32-bit register, and is part of the Identification registers functional group. This register is Read Only.

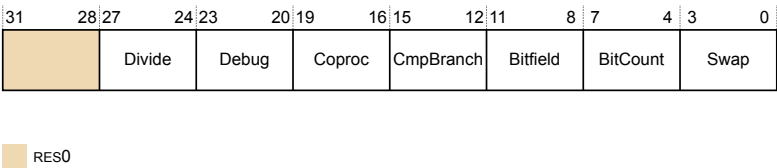


Figure 54: ID_ISAR0_EL1 bit assignments

RES0, [31:28]	<i>RES0</i>	Reserved.
Divide, [27:24]	Indicates the implemented Divide instructions:	

	2	<ul style="list-style-type: none"> SDIV and UDIV in the T32 instruction set. SDIV and UDIV in the A32 instruction set.
Debug, [23:20]	Indicates the implemented Debug instructions:	
	1	BKPT.
Coproc, [19:16]	Indicates the implemented Coprocessor instructions:	
	0	None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.
CmpBranch, [15:12]	Indicates the implemented combined Compare and Branch instructions in the instruction set:	
	1	CBNZ and CBZ.
Bitfield, [11:8]	Indicates the implemented bit field instructions:	
	1	BFC, BFI, SBFX, and UBFX.
BitCount, [7:4]	Indicates the implemented Bit Counting instructions:	
	1	CLZ.
Swap, [3:0]	Indicates the implemented Swap instructions in the instruction set:	
	0	None implemented.
Configurations	<p>In an AArch64-only implementation, this register is <i>UNKNOWN</i>.</p> <p>Must be interpreted with ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, and ID_ISAR6_EL1. See:</p> <ul style="list-style-type: none"> ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1. ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1. ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1. ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1. ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1. 	

- [ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1](#).

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1

The ID_ISAR1_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
Jazelle				Interwork				Immediate				IfThen			
								Extend				Except_AR			
												Except			
												Endian			

Figure 55: ID_ISAR1_EL1 bit assignments

Jazelle, [31:28]

Indicates the implemented *Jazelle state* instructions:

- 1 Adds the BXJ instruction, and the J bit in the *PSR*.

Interwork, [27:24]

Indicates the implemented Interworking instructions:

- 3
 - The BX instruction, and the T bit in the .
 - The BLX instruction. The PC loads have BX-like behavior.
 - Data-processing instructions in the *A32* instruction set with the PC as the destination and the S bit clear, have BX-like behavior.

Immediate, [23:20]

Indicates the implemented data-processing instructions with long immediates:

- 1
 - The MOV_T instruction.
 - The MOV instruction encodings with zero-extended 16-bit immediates.
 - The *T32* ADD and SUB instruction encodings with zero-extended 12-bit immediates, and other ADD, ADR, and SUB encodings cross-referenced by the

		pseudocode for those encodings.
IfThen, [19:16]	Indicates the implemented If-Then instructions in the instruction set:	
	1	The <code>IT</code> instructions, and the <code>IT</code> bits in the <code>s</code> .
Extend, [15:12]	Indicates the implemented Extend instructions:	
	2	<ul style="list-style-type: none"> The <code>SXTB</code>, <code>SXTH</code>, <code>UXTB</code>, and <code>UXTH</code> instructions. The <code>SXTB16</code>, <code>SXTAB</code>, <code>SXTAB16</code>, <code>SXTAH</code>, <code>UXTB16</code>, <code>UXTAB</code>, <code>UXTAB16</code>, and <code>UXTAH</code> instructions.
Except_AR, [11:8]	Indicates the implemented A profile <i>exception</i> -handling instructions:	
	1	The <code>SRS</code> and <code>RFE</code> instructions, and the A profile forms of the <code>CPS</code> instruction.
Except, [7:4]	Indicates the implemented -handling instructions in the instruction set:	
	1	The <code>LDM (return)</code> , <code>LDM (user registers)</code> , and <code>STM (user registers)</code> instruction versions.
Endian, [3:0]	Indicates the implemented Endian instructions:	
	1	The <code>SETEND</code> instruction, and the <code>E</code> bit in the <code>s</code> .
Configurations	<p>In an <i>AArch64</i>-only implementation, this register is <i>unknown</i>.</p> <p>Must be interpreted with <code>ID_ISAR0_EL1</code>, <code>ID_ISAR2_EL1</code>, <code>ID_ISAR3_EL1</code>, <code>ID_ISAR4_EL1</code>, <code>ID_ISAR5_EL1</code>, and <code>ID_ISAR6_EL1</code>. See:</p> <ul style="list-style-type: none"> <code>ID_ISAR0_EL1</code>, <i>AArch32</i> Instruction Set Attribute Register 0, EL1. <code>ID_ISAR2_EL1</code>, Instruction Set Attribute Register 2, EL1. <code>ID_ISAR3_EL1</code>, Instruction Set Attribute Register 3, EL1. <code>ID_ISAR4_EL1</code>, Instruction Set Attribute Register 4, EL1. 	

- [ID_ISAR5_EL1](#), Instruction Set Attribute Register 5, EL1.
- [ID_ISAR6_EL1](#), Instruction Set Attribute Register 6, EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1

The ID_ISAR2_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR2_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
Reversal	PSR_AR		MultU		MultS		Mult				MemHint		LoadStore		

MultiAccessInt —

Figure 56: ID_ISAR2_EL1 bit assignments

Reversal, [31:28]

Indicates the implemented Reversal instructions:

2

The REV, REV16, REVSH, and RBIT instructions.

[PSR_AR](#), [27:24]

Indicates the implemented A and R profile instructions to manipulate the :

1

The MRS and MSR instructions, and the [exception](#) return forms of data-processing instructions.

The return forms of the data-processing instructions are:

- In the [A32](#) instruction set, data-processing instructions with the PC as the destination and the S bit set.
- In the [T32](#) instruction set, the SUBSPC, LR, #N instruction.

MultU, [23:20]

Indicates the implemented advanced unsigned Multiply instructions:

2

The UMULL, UMLAL, and UMAAL instructions.

MultS, [19:16]

Indicates the implemented advanced signed Multiply instructions.

3

- The SMULL and SMLAL instructions.

- The SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT instructions, and the Q bit in the s.
- The SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLS LD, SMLS LDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSD X instructions.

Mult, [15:12]

Indicates the implemented additional Multiply instructions:

2

The MUL, MLA and MLS instructions.

MultiAccessInt, [11:8]

Indicates the support for interruptible multi-access instructions:

0

No support. This means the LDM and *STM* instructions are not interruptible.

MemHint, [7:4]

Indicates the implemented memory *hint instructions*:

4

The PLD, PLI, and PLDW instructions.

LoadStore, [3:0]

Indicates the implemented additional load/store instructions:

2

The LDRD and STRD instructions.

The Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, and LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, and STLEXD) instructions.

Configurations

In an *AArch64*-only implementation, this register is *UNKNOWN*.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, and ID_ISAR6_EL1. See:

- ID_ISAR0_EL1, *AArch32* Instruction Set Attribute Register 0, EL1.
- ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1.
- ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1.
- ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1.
- ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1.
- ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1

The ID_ISAR3_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR3_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0																
T32EE				TrueNOP				T32Copy				TabBranch				SynchPrim				SVC				SIMD				Saturate			

	1	Support for instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register.
TabBranch, [19:16]	Indicates the implemented Table Branch instructions in the instruction set.	
	1	The TBB and TBH instructions.
SynchPrim, [15:12]	Indicates the implemented Synchronization Primitive instructions:	
	2	<ul style="list-style-type: none"> • The LDREX and STREX instructions. • The CLREX, LDREXB, STREXB, and STREXH instructions. • The LDREXD and STREXD instructions.
SVC, [11:8]	Indicates the implemented instructions:	
	1	The instruction.
SIMD, [7:4]	Indicates the implemented <i>Single Instruction Multiple Data</i> () instructions.	
	3	<ul style="list-style-type: none"> • The SSAT and USAT instructions, and the Q bit in the <i>PSRs</i>. • The PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8,

USAX, UXTAB16,
UXTB16 instructions,
and the GE[3:0] bits in
the s.

The field relates
only to implemented
instructions that perform
operations on the *general-
purpose registers*. In
an implementation that
supports *Advanced* and
floating-point instructions,
MVFR0, MVFR1, and
MVFR2 give information
about the implemented
instructions.

Saturate, [3:0]

Indicates the implemented Saturate instructions:

1

The QADD, QDADD,
QDSUB, QSUB Q bit in the
s.

Configurations

In an *AArch64*-only implementation, this register is
UNKNOWN.

Must be interpreted with ID_ISAR0_EL1,
ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR4_EL1,
ID_ISAR5_EL1, and ID_ISAR6_EL1. See:

- ID_ISAR0_EL1, *AArch32* Instruction Set Attribute Register 0, EL1.
- ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1.
- ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1.
- ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1.
- ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1.
- ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

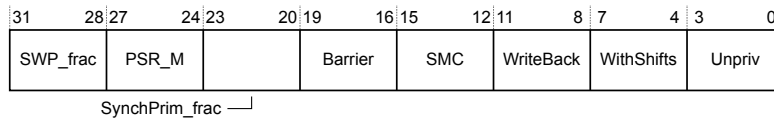
ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1

The ID_ISAR4_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR4_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

**Figure 58: ID_ISAR4_EL1 bit assignments****SWP_frac, [31:28]**

Indicates support for the memory system locking the bus for SWP or SWPB instructions:

- 0** SWP and SWPB instructions not implemented.

PSR_M, [27:24]

Indicates the implemented M profile instructions to modify the s:

- 0** None implemented.

SynchPrim_frac, [23:20]

This field is used with the ID_ISAR3.SynchPrim field to indicate the implemented Synchronization Primitive instructions:

- 0**
- The LDREX and STREX instructions.
 - The CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions.
 - The LDREXD and STREXD instructions.

Barrier, [19:16]

Indicates the supported Barrier instructions in the [A32](#) and [T32](#) instruction sets:

- 1** The DMB, DSB, and ISB barrier instructions.

SMC, [15:12]

Indicates the implemented SMC instructions:

- 0** None implemented.

WriteBack, [11:8]

Indicates the support for -Back [addressing modes](#):

- 1** Core supports all the -Back s as defined in Arm® v8-A.

WithShifts, [7:4]

Indicates the support for instructions with shifts.

- 4**
- Support for shifts of loads and stores over the range LSL 0-3.
 - Support for other constant shift options,

- both on load/store and other instructions.
- Support for register-controlled shift options.

Unpriv, [3:0]	Indicates the implemented unprivileged instructions.
2	<ul style="list-style-type: none">• The LDRBT, LDRT, STRBT, and STRT instructions.• The LDRHT, LDRSBT, LDRSHT, and STRHT instructions.

Configurations	<p>In an <i>AArch64</i>-only implementation, this register is <i>UNKNOWN</i>.</p> <p>Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR5_EL1, and ID_ISAR6_EL1. See:</p> <ul style="list-style-type: none">• ID_ISAR0_EL1, <i>AArch32</i> Instruction Set Attribute Register 0, EL1.• ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1.• ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1.• ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1.• ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1.• ID_ISAR6_EL1, Instruction Set Attribute Register 6, EL1.
----------------	---

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1

The ID_ISAR5_EL1 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR5_EL1 is a 32-bit register, and is part of the Identification registers functional group. This register is Read Only.

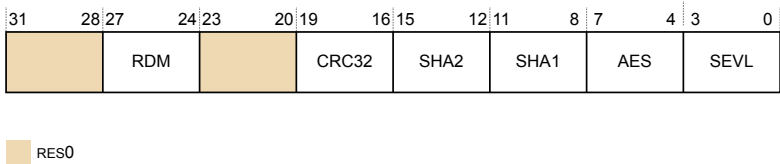


Figure 59: ID_ISAR5_EL1 bit assignments

RES0, [31:28]	RES0	Reserved.
RDM, [27:24]	VQRDMLAH and VQRDMLSH instructions in AArch32 . The value is:	
	1	VQRDMLAH and VQRDMLSH instructions are implemented.
[23:20]	RES0	Reserved.
CRC32, [19:16]	Indicates whether CRC32 instructions are implemented in state. The value is:	
	1	CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions are implemented.
SHA2, [15:12]	Indicates whether SHA2 instructions are implemented in state. The possible values are:	
	0	No SHA2 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.
	1	SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 instructions are implemented. This is the value when the Cryptographic Extensions are implemented and enabled.
SHA1, [11:8]	Indicates whether SHA1 instructions are implemented in state. The possible values are:	
	0	No SHA1 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.
	1	SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 instructions are implemented. This is the value when the Cryptographic Extensions

are implemented and enabled.

AES, [7:4]

Indicates whether AES instructions are implemented in state. The possible values are:

- | | |
|----------|---|
| 0 | No AES instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled. |
| 2 | <ul style="list-style-type: none"> • AESE, AESD, AESMC, and AESIMC implemented. • PMULL and PMULL2 instructions operating on 64-bit data. <p>This is the value when the Cryptographic Extensions are implemented and enabled.</p> |

SEVL, [3:0]

Indicates whether the SEVL instruction is implemented:

- | | |
|----------|---------------------------------------|
| 1 | SEVL implemented to send event local. |
|----------|---------------------------------------|

Configurations

ID_ISAR5 must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR6_EL1. See:

- [ID_ISAR0_EL1](#), Instruction Set Attribute Register 0, EL1.
- [ID_ISAR1_EL1](#), Instruction Set Attribute Register 1, EL1.
- [ID_ISAR2_EL1](#), Instruction Set Attribute Register 2, EL1.
- [ID_ISAR3_EL1](#), Instruction Set Attribute Register 3, EL1.
- [ID_ISAR4_EL1](#), Instruction Set Attribute Register 4, EL1.
- [ID_ISAR6_EL1](#), Instruction Set Attribute Register 6, EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1

The ID_ISAR6_EL1 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR6_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.



RES0

Figure 60: ID_ISAR6_EL1 bit assignments

RES0, [31:8]	RES0	Reserved.
DP, [7:4]	UDOT and SDOT instructions. The value is: 0001	UDOT and SDOT instructions are implemented.
RES0, [3:0]	RES0	Reserved.
Configurations	There is one copy of this register that is used in both Secure and Non-secure states.	

ID_ISAR6_EL1 must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1. See:

- [ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1.](#)
- [ID_ISAR1_EL1, Instruction Set Attribute Register 1, EL1.](#)
- [ID_ISAR2_EL1, Instruction Set Attribute Register 2, EL1.](#)
- [ID_ISAR3_EL1, Instruction Set Attribute Register 3, EL1.](#)
- [ID_ISAR4_EL1, Instruction Set Attribute Register 4, EL1.](#)
- [ID_ISAR5_EL1, Instruction Set Attribute Register 5, EL1.](#)

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1

The ID_MMFR0_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28:27	24:23	20:19	16:15	12:11	8	7	4	3	0
InnerShr	FCSE	AuxReg	TCM	ShareLvl	OuterShr	PMSA		VMSA		

Figure 61: ID_MMFR0_EL1 bit assignments**InnerShr, [31:28]**

Indicates the innermost shareability domain implemented:

1 Implemented with hardware coherency support.

FCSE, [27:24]

Indicates support for *Fast Context Switch Extension* (FCSE):

0 Not supported.

AuxReg, [23:20]

Indicates support for Auxiliary registers:

2 Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

TCM, [19:16]

Indicates support for s and associated DMAs:

0 Not supported.

ShareLvl, [15:12]

Indicates the number of shareability levels implemented:

1 Two levels of shareability implemented.

OuterShr, [11:8]

Indicates the outermost shareability domain implemented:

1 Implemented with hardware coherency support.

PMSA, [7:4]

Indicates support for a *Protected Memory System Architecture* (PMSA):

0 Not supported.

VMSA, [3:0]

Indicates support for a *Virtual Memory System Architecture* (VMSA).

5 Support for:

- VMSAv7, with support for *remapping* and the Access flag.
- The PXN bit in the Short-descriptor

[translation table](#)

format descriptors.

- The Long-descriptor format.

Configurations

Must be interpreted with ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- [ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1](#).
- [ID_MMFR2_EL1, Memory Model Feature Register 2, EL1](#).
- [ID_MMFR3_EL1, Memory Model Feature Register 3, EL1](#).
- [ID_MMFR4_EL1, Memory Model Feature Register 4, EL1](#).

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1

The ID_MMFR1_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
BPred		L1TstCln		L1Uni		L1Hvd		L1UniSW		L1HvdSW		L1UniVA		L1HvdVA	

	0	None supported.
L1UniSW, [15:12]	Indicates the supported L1 cache line maintenance operations by set/way, for a unified cache implementation:	
	0	None supported.
L1HvdSW, [11:8]	Indicates the supported L1 cache line maintenance operations by set/way, for a Harvard cache implementation:	
	0	None supported.
L1UniVA, [7:4]	Indicates the supported L1 cache line maintenance operations by <i>MVA</i> , for a unified cache implementation:	
	0	None supported.
L1HvdVA, [3:0]	Indicates the supported L1 cache line maintenance operations by , for a Harvard cache implementation:	
	0	None supported.
Configurations	<p>Must be interpreted with ID_MMFR0_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:</p> <ul style="list-style-type: none"> • ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1. • ID_MMFR2_EL1, Memory Model Feature Register 2, EL1. • ID_MMFR3_EL1, Memory Model Feature Register 3, EL1. • ID_MMFR4_EL1, Memory Model Feature Register 4, EL1. 	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1

The ID_MMFR2_EL1 provides information about the implemented memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR2_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
HWAaccFlg		WFIStall		MemBarr		UniTLB		HvdTLB		LL1HvdRng		L1HvdBG		L1HvdFG	

Figure 63: ID_MMFR2_EL1 bit assignments

HWAccFlg, [31:28]

Hardware access flag. Indicates support for a hardware access flag, as part of the VMSAv7 implementation:

0 Not supported.

WFISall, [27:24]

Wait For Interrupt Stall. Indicates the support for *Wait For Interrupt* (WFI) stalling:

1 Support for WFI stalling.

MemBarr, [23:20]

Memory Barrier. Indicates the supported CP15 memory barrier operations.

2 Supported CP15 memory barrier operations are:

- *Data Synchronization Barrier* (DSB).
- *Instruction Synchronization Barrier* (ISB).
- *Data Memory Barrier* (DMB).

Uni^{TLB}, [19:16]

Unified . Indicates the supported maintenance operations, for a unified implementation.

6 Supported unified maintenance operations are:

- Invalidate all entries in the .
- Invalidate entry by [MVA](#).
- Invalidate entries by ASID match.
- Invalidate instruction and data entries by All ASID. This is a shared unified operation.
- Invalidate Hyp mode unified entry by .
- Invalidate entire Non-secure EL1 and EL0 unified .
- Invalidate entire Hyp mode unified .
- ILIS, IALIS, ILHIS, IL, IAL, and ILH.
- IIPAS2IS, IIPAS2LIS,

	IIPAS2, and IIPAS2L.
Hvd, [15:12]	Harvard . Indicates the supported maintenance operations, for a Harvard implementation: 0 Not supported.
LL1HvdRng, [11:8]	L1 Harvard cache Range. Indicates the supported L1 cache maintenance range operations, for a Harvard cache implementation: 0 Not supported.
L1HvdBG, [7:4]	L1 Harvard cache Background fetch. Indicates the supported L1 cache background prefetch operations, for a Harvard cache implementation: 0 Not supported.
L1HvdFG, [3:0]	L1 Harvard cache Foreground fetch. Indicates the supported L1 cache foreground prefetch operations, for a Harvard cache implementation: 0 Not supported.
Configurations	Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See: <ul style="list-style-type: none"> • ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1. • ID_MMFR1_EL1, Memory Model Feature Register 1, EL1. • ID_MMFR3_EL1, Memory Model Feature Register 3, EL1. • ID_MMFR4_EL1, Memory Model Feature Register 4, EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1

The ID_MMFR4_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR4_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	24	23	20	19	16	15	12	11	8	7	4	3	0
RAZ			LSM		HD		CNP		XNX		AC2		SpecSEI

Figure 64: ID_MMFR4_EL1 bit assignments**RAZ, [31:24]**

Read-As-Zero.

LSM, [23:20]

Load/Store Multiple. Indicates whether adjacent loads or stores can be combined. The value is:

0

LSMAOE and nTLSMD bit not supported.

HD, [19:16]Presence of Hierarchical Disables. Enables an operating system or hypervisor to hand over up to 4 bits of the last level *page table* descriptor (bits[62:59] of the entry) for use by hardware for *IMPLEMENTATION DEFINED* usage. The value is:**2**

Hierarchical Permission Disables and Hardware allocation of bits[62:59] supported.

CNP, [15:12]Common Not Private. Indicates support for selective sharing of *TLB* entries across multiple *PEs*. The value is:**1**

CnP bit supported.

XNX, [11:8]Execute Never. Indicates whether the stage 2 *translation tables* allows the stage 2 control of whether memory is executable at EL1 independent of whether memory is executable at EL0. The value is:**1**

EL0/EL1 execute control distinction at stage2 bit supported.

AC2, [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. The value is:

1

ACTLR2 and HACTLR2 are implemented.

SpecSEI, [3:0]Describes whether the *core* can generate SError interrupt *exceptions* from speculative *reads* of memory, including speculative instruction fetches. The value is:**0**The never generates an SError interrupt due to an external *abort* on a speculative .**Configurations**

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR3_EL1. See:

- ID_MMFR0_EL1, *AArch32 Memory Model Feature Register 0, EL1*.
- ID_MMFR1_EL1, *Memory Model Feature Register 1, EL1*.
- ID_MMFR2_EL1, *Memory Model Feature Register 2, EL1*.
- ID_MMFR3_EL1, *Memory Model Feature Register 3, EL1*.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1

The ID_PFR0_EL1 provides top-level information about the instruction sets supported by the core in AArch32.

Bit field descriptions

ID_PFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	28	27				20	19		16	15		12	11		8	7		4	3		0
RAS									CSV2		State3		State2		State1		State0				

RES0

Figure 65: ID_PFR0_EL1 bit assignments

RAS, [31:28]		RAS extension version. The value is:
	1	Version 1 of the RAS extension is present.
RES0, [27:20]	RES0	Reserved.
CSV2, [19:16]	0	This <i>device</i> does not disclose whether branch targets trained in one context can affect speculative execution in a different context.
	1	Branch targets trained in one context cannot affect speculative execution in a different hardware described context. This is the reset value.

State3, [15:12]	Indicates support for <i>Thumb Execution Environment (T32EE)</i> instruction set. This value is:
	0Core does not support the EE instruction set.
State2, [11:8]	Indicates support for Jazelle. This value is:
	1Core supports trivial implementation of Jazelle.
State1, [7:4]	Indicates support for instruction set. This value is:
	3Core supports encoding after the introduction of <i>Thumb-2</i> technology, and for all 16-bit and 32-bit basic instructions.
State0, [3:0]	Indicates support for <i>A32</i> instruction set. This value is:
	1instruction set implemented.
Configurations	There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1

The ID_PFR1_EL1 provides information about the programmers model and architecture extensions supported by the core.

Bit field descriptions

ID_PFR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

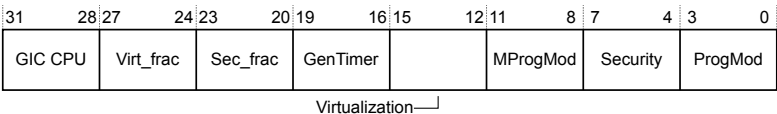


Figure 66: ID_PFR1_EL1 bit assignments

GIC CPU, [31:28]	GIC CPU support:
	0GIC CPU interface is disabled, GICCDISABLE is HIGH, or not implemented.

	1	GIC CPU interface is implemented and enabled, GICCDISABLE is LOW.
Virt_frac, [27:24]	0	No features from the Armv7 Virtualization Extensions are implemented.
Sec_frac, [23:20]	0	No features from the Armv7 Virtualization Extensions are implemented.
GenTimer, [19:16]	Generic Timer support:	
	1	Generic Timer supported.
Virtualization, [15:12]	Virtualization support:	
	0	Virtualization not implemented.
MProgMod, [11:8]	M profile programmers model support:	
	0	Not supported.
Security, [7:4]	Security support:	
	0	Security not implemented.
ProgMod, [3:0]	Indicates support for the standard programmers model for Armv4 and later. Model must support User, FIQ , IRQ , Supervisor, Abort, Undefined, and System modes:	
	0	Not supported.
Configurations	There are no configuration notes.	

ID_PFR2_EL1, AArch32 Processor Feature Register 2, EL1

The ID_PFR2_EL1 provides information about the programmers model and architecture extensions supported by the core.

Bit field descriptions

ID_PFR2_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

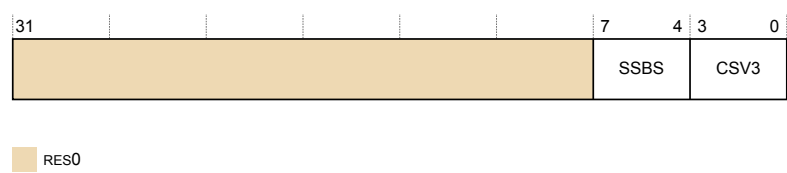


Figure 67: ID_PFR2_EL1 bit assignments

RES0, [31:8]	RES0	Reserved.
SSBS, [7:4]	1	AArch32 provides the PSTATE.SSBS mechanism to mark <i>regions</i> that are <i>Speculative Store Bypassing Safe</i> (SSBS).
CSV3, [3:0]	1	Data loaded under speculation with a permission or domain <i>fault</i> cannot be used to form an address or generate condition codes to be used by instructions newer than the load in the speculative sequence. This is the reset value.

Configurations

There are no configuration notes.

LORC_EL1, LORegion Control Register, EL1

The LORC_EL1 register enables and disables LORegions, and selects the current LORegion descriptor.

Bit field descriptions

LORC_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

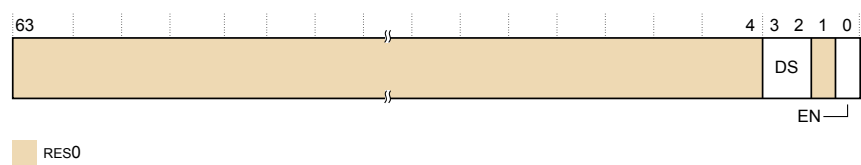


Figure 68: LORC_EL1 bit assignments

[63:4]	Reserved, RES0.
DS, [3:2]	Descriptor Select. Number that selects the current LORegion descriptor accessed by the LORSA_EL1, LOREA_EL1, and LORN_EL1 registers.

[1]	Reserved, <i>RES0</i> .				
EN, [0]	Enable. The possible values are: <table><tr><td>0</td><td>Disabled. This is the reset value.</td></tr><tr><td>1</td><td>Enabled.</td></tr></table>	0	Disabled. This is the reset value.	1	Enabled.
0	Disabled. This is the reset value.				
1	Enabled.				

Configurations	<p>RW fields in this register reset to architecturally <i>UNKNOWN</i> values.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>
----------------	---

LORID_EL1, LORegion ID Register, EL1

The LORID_EL1 ID register indicates the supported number of LORegions and LORegion descriptors.

Bit field descriptions

LORID_EL1 is a 64-bit register.

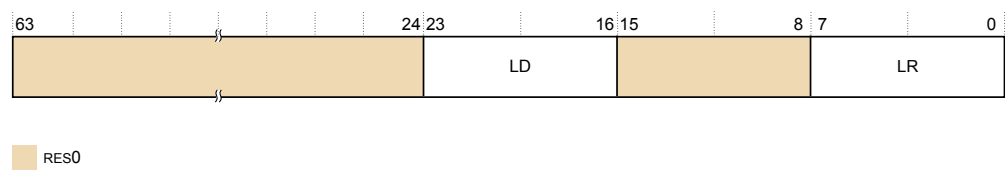


Figure 69: LORID_EL1 bit assignments

[63:24]	Reserved, <i>RES0</i> .		
LD, [23:16]	Number of LORegion descriptors supported by the implementation, expressed as binary 8-bit number. The value is: <table><tr><td>04</td><td>Four LORegion descriptors are supported.</td></tr></table>	04	Four LORegion descriptors are supported.
04	Four LORegion descriptors are supported.		
[15:8]	Reserved, <i>RES0</i> .		
LR, [7:0]	Number of LORegions supported by the implementation, expressed as a binary 8-bit number. The value is: <table><tr><td>04</td><td>Four LORegions are supported.</td></tr></table>	04	Four LORegions are supported.
04	Four LORegions are supported.		

Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>
----------------	--

LORN_EL1, LORegion Number Register, EL1

The LORN_EL1 register holds the number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bit field descriptions

LORN_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

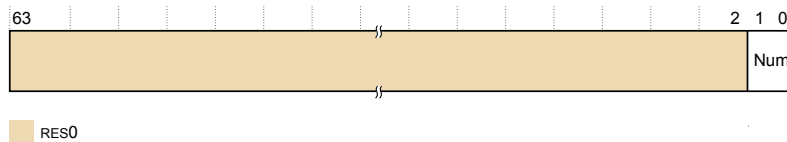


Figure 70: LORN_EL1 bit assignments

[63:2]

Reserved, *RES0*.

Num, [1:0]

Indicates the LORegion number.

Configurations

RW fields in this register reset to architecturally *UNKNOWN* values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MDCR_EL3, Monitor Debug Configuration Register, EL3

The MDCR_EL3 provides configuration options for Security to self-hosted debug.

Bit field descriptions

MDCR_EL3 is a 32-bit register, and is part of:

- The Debug registers functional group.
- The Security registers functional group.

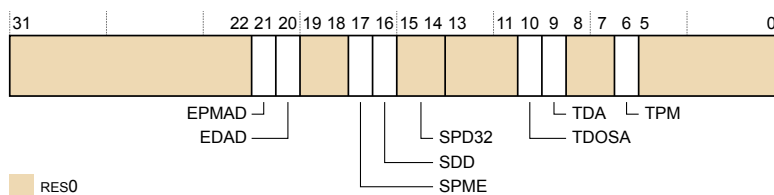


Figure 71: MDCR_EL3 bit assignments

RES0, [31:22]

RES0

Reserved.

EPMAD, [21]

External debugger access to Performance Monitors registers disabled. This disables access to these registers by an external debugger. The possible values are:

- | | |
|----------|---|
| 0 | Access to Performance Monitors registers from external debugger is permitted. |
| 1 | Access to Performance Monitors registers from external debugger is disabled, unless overridden by authentication interface. |

EDAD, [20]

External debugger access to *breakpoint* and *watchpoint* registers disabled. This disables access to these registers by an external debugger. The possible values are:

- | | |
|----------|--|
| 0 | Access to and point registers from external debugger is permitted. |
| 1 | Access to and point registers from external debugger is disabled, unless overridden by authentication interface. |

SPME, [17]

Secure performance monitors enable. This enables event counting *exceptions* from Secure state. The possible values are:

- | | |
|----------|--|
| 0 | Event counting prohibited in Secure state. |
| 1 | Event counting allowed in Secure state. |

SPD32, [15:14]

res0 Reserved.

TDOSA, [10]

Trap accesses to the OS debug system registers, OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1 OS.

- | | |
|----------|---|
| 0 | Accesses are not trapped. |
| 1 | Accesses to the OS debug system registers are trapped to EL3. |

The reset value is *UNKNOWN*.

TDA, [9]

Trap accesses to the remaining sets of debug registers to EL3.

- | | |
|----------|---------------------------|
| 0 | Accesses are not trapped. |
|----------|---------------------------|

1 Accesses to the remaining debug system registers are trapped to EL3.

The reset value is *UNKNOWN*.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MIDR_EL1, Main ID Register, EL1

The MIDR_EL1 provides identification information for the core, including an implementer code for the device and a device ID number.

Bit field descriptions

MIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

31	24	23	20	19	16	15	4	3	0
Implementer		Variant		Architecture		PartNum			Revision

Figure 72: MIDR_EL1 bit assignments

Implementer, [31:24]	Indicates the implementer code. This value is:
41	ASCII character 'A' - implementer is Arm® Limited.
Variant, [23:20]	Indicates the variant number of the <i>core</i> . This is the major revision number <i>x</i> in the rx part of the rxy description of the product revision status. This value is:
0x4	r4p1.
Architecture, [19:16]	Indicates the architecture code. This value is:
F	Defined by CPUID scheme.
PartNum, [15:4]	Indicates the primary part number. This value is:
0xD0B	Cortex®-A76 .
Revision, [3:0]	Indicates the minor revision number of the . This is the minor revision number <i>y</i> in the py part of the rxy description of the product revision status. This value is:
0x0	r4p1.

Configurations

The MIDR_EL1 is architecturally mapped to external MIDR_EL1 register.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MPIDR_EL1, Multiprocessor Affinity Register, EL1

The MPIDR_EL1 provides an additional core identification mechanism for scheduling purposes in a cluster.

Bit field descriptions

MPIDR_EL1 is a 64-bit register, and is part of the Other system control registers functional group.

This register is Read Only.

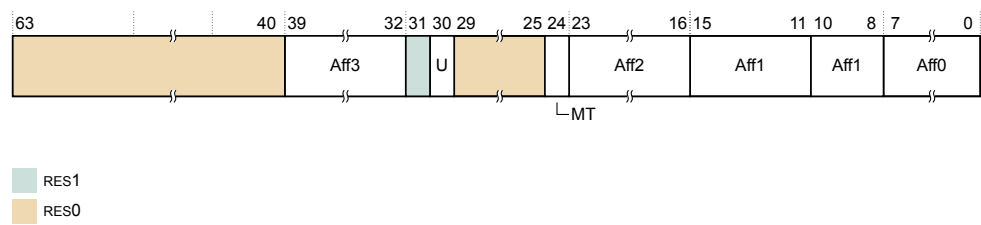


Figure 73: MPIDR_EL1 bit assignments

RES0, [63:40]	RES0	Reserved.
Aff3, [39:32]	CLUSTERID	Indicates the value <i>read</i> in the CLUSTERIDAFF3 configuration signal.
RES1, [31]	RES1	Reserved.
U, [30]		Indicates a single <i>core</i> system, as distinct from 0 in a <i>cluster</i> . This value is: 0 Core is part of a multiprocessor system. This is the value for implementations with more than one , and for implementations with an <i>ACE</i> or CHI master interface.
RES0, [29:25]	RES0	Reserved.

MT, [24]	Indicates whether the lowest level of affinity consists of logical s that are implemented using a multithreading type approach. This value is:
1	Performance of <i>PEs</i> at the lowest affinity level is very interdependent. Affinity0 represents this. Cortex [®] -A76 is not multithreaded, but may be in a system with other s that are multithreaded.
Aff2, [23:16]	Affinity level 2. Second highest level affinity field.
CLUSTERID	Indicates the value in the CLUSTERIDAFF2 configuration signal.
Aff1, [15:11]	Part of Affinity level 1. Third highest level affinity field.
RAZ	Read-As-Zero.
Aff1, [10:8]	Part of Affinity level 1. Third highest level affinity field.
CPUID	Identification number for each CPU in the :
0	MP1: CPUID: 0. to
7	MP8: CPUID: 7.
Aff0, [7:0]	Affinity level 0. The level identifies individual this within a multithreaded . The Cortex [®] -A76 is single-threaded, so this field has the value 00.
Configurations	MPIDR_EL1[31:0] is mapped to external register EDDEVAF0. MPIDR_EL1[63:32] is mapped to external register EDDEVAF1. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .

PAR_EL1, Physical Address Register, EL1

The PAR_EL1 returns the output address from an address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Bit field descriptions, *PAR_EL1.F* is 0

The following figure shows the R bit assignments when R.F is 0.

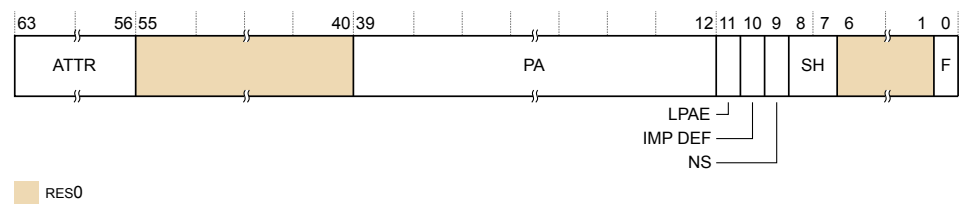


Figure 74: R bit assignments, *R_EL1.F* is 0

IMP DEF, [10]

IMPLEMENTATION DEFINED. Bit[10] is *RES0*.

F, [0]

Indicates whether the instruction performed a successful address translation.

0	Address translation completed successfully.
1	Address translation <i>aborted</i> .

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Bit field descriptions, *R_EL1.F* is 1

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

REVIDR_EL1, Revision ID Register, EL1

The REVIDR_EL1 provides revision information, additional to MIDR_EL1, that identifies minor fixes (errata) which might be present in a specific implementation of the Cortex®-A76 core.

Bit field descriptions

REVIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.



Figure 75: REVIDR_EL1 bit assignments

<i>implementation defined, [31:0]</i>	<i>IMPLEMENTATION DEFINED.</i>
Configurations	<p>There are no configuration notes.</p> <p>Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>

RMR_EL3, Reset Management Register

The RMR_EL3 controls the execution state that the core boots into and allows request of a Warm reset.

Bit field descriptions

RMR_EL3 is a 32-bit register, and is part of the Reset management registers functional group.

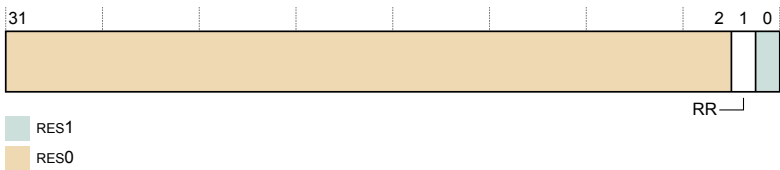


Figure 76: RMR_EL3 bit assignments

RES0, [31:2]	<i>RES0</i>	Reserved.
RR, [1]	<p>Reset Request. The possible values are:</p> <p>0 This is the reset value on both a Warm and a <i>Cold reset</i>.</p> <p>1 Requests a <i>Warm reset</i>.</p> <p>The bit is strictly a request.</p>	
RES1, [0]	<i>RES1</i>	Reserved.

Configurations	<p>There are no configuration notes.</p> <p>Details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i>.</p>
----------------	---

RVBAR_EL3, Reset Vector Base Address Register, EL3

RVBAR_EL3 contains the *IMPLEMENTATION DEFINED* address that execution starts from after reset.

Bit field descriptions

RVBAR_EL3 is a 64-bit register, and is part of the Reset management registers functional group.

This register is Read Only.

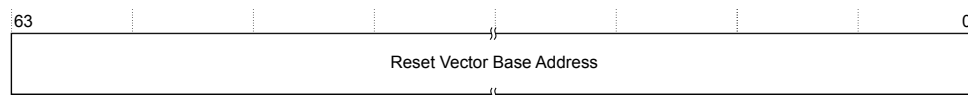


Figure 77: RVBAR_EL3 bit assignments

RVBA, [63:0]

Reset Vector Base Address. The address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and bits [63:40] are 000000 because the address must be within the *physical address* size supported by the *core*.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

SCTLR_EL1, System Control Register, EL1

The SCTLR_EL1 provides top-level control of the system, including its memory system, at EL1 and EL0.

Bit field descriptions

SCTLR_EL1 is a 32-bit register, and is part of the Other system control registers functional group.

This register resets to 0000000030D50838.

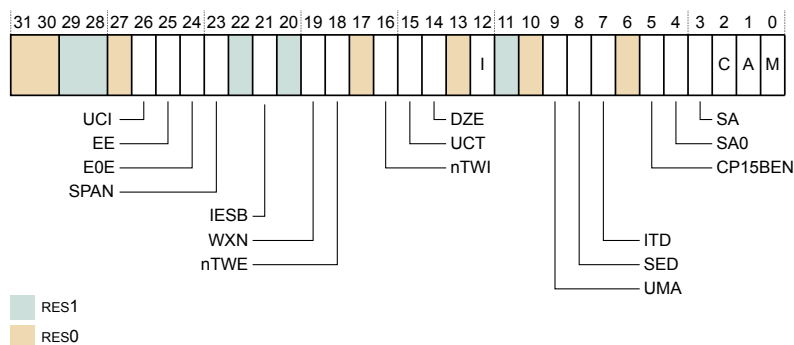


Figure 78: SCTLR_EL1 bit assignments

RES0, [31:30]	<i>RES0</i>	Reserved.
RES1, [29:28]	<i>RES1</i>	Reserved.
RES0, [27]	<i>RES0</i>	Reserved.
EE, [25]	Exception <i>endianness</i> . The value of this bit controls the for explicit data accesses at EL1. This value also indicates the of the <i>translation table</i> data for lookups. The possible values of this bit are:	
	0	Little-endian.
	1	Big-endian.
ITD, [7]	This field is <i>RAZ/WI</i> .	
RES0, [6]	<i>RES0</i>	Reserved.
CP15BEN, [5]	CP15 barrier enable. The possible values are:	
	0	CP15 barrier operations disabled. Their encodings are <i>UNDEFINED</i> .
	1	CP15 barrier operations enabled.
M, [0]	<i>MMU</i> enable. The possible values are:	
	0	EL1 and EL0 stage 1 disabled.
	1	EL1 and EL0 stage 1 enabled.
Configurations	There are no configuration notes. Bit fields and details that are not provided in this description are architecturally defined. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> .	

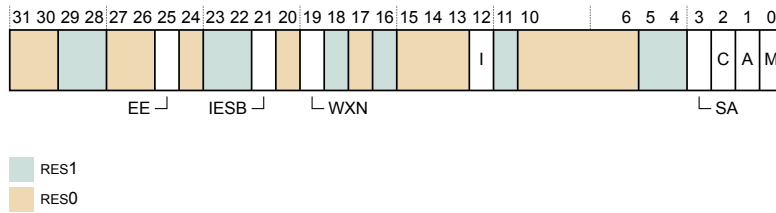
SCTLR_EL3, System Control Register, EL3

The SCTLR_EL3 provides top-level control of the system, including its memory system at EL3.

Bit field descriptions

SCTLR_EL3 is a 32-bit register, and is part of the Other system control registers functional group.

This register resets to 30C50838.

**Figure 79: SCTLR_EL3 bit assignments****RES0, [31:30]****RES0**

Reserved.

RES1, [29:28]**RES1**

Reserved.

RES0, [27:26]**RES0**

Reserved.

EE, [25]Exception *endianness*. This bit controls the for:

- Explicit data accesses at EL3.
- Stage 1 *translation table* walks at EL3.

The possible values are:

- | | |
|----------|----------------|
| 0 | Little endian. |
| 1 | Big endian. |

The reset value is determined by the CFGEND configuration signal.

I, [12]

Global instruction cache enable. The possible values are:

- | | |
|----------|---|
| 0 | Instruction caches disabled. This is the reset value. |
| 1 | Instruction caches enabled. |

C, [2]

Global enable for data and unified caches. The possible values are:

- | | |
|----------|--|
| 0 | Disables data and unified caches. This is the reset value. |
| 1 | Enables data and unified caches. |

M, [0]Global enable for the EL3 *MMU*. The possible values are:

- | | |
|----------|---|
| 0 | Disables EL3 . This is the reset value. |
| 1 | Enables EL3 . |

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

TCR_EL1, Translation Control Register, EL1

The TCR_EL1 determines which Translation Base registers define the base address register for a translation table walk required for stage 1 translation of a memory access from EL0 or EL1 and holds cacheability and shareability information.

Bit field descriptions

TCR_EL1 is a 64-bit register, and is part of the Virtual memory control registers functional group.

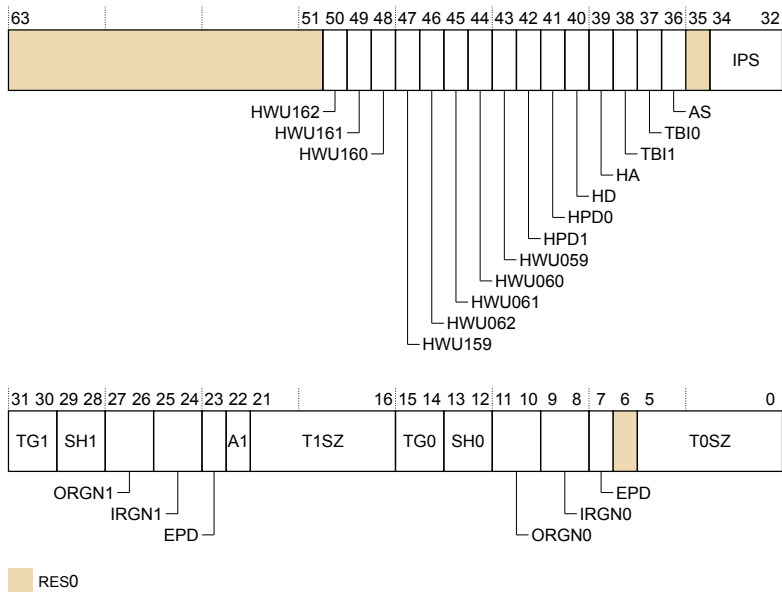


Figure 80: TCR_EL1 bit assignments

Note: Bits[50:39], architecturally defined, are implemented in the *core*.

HD, [40]	Hardware management of <i>dirty</i> state in stage 1 translations from EL0 and EL1. The possible values are:
0	Stage 1 hardware management of state disabled.
1	Stage 1 hardware management of state enabled, only if the HA bit is also set to 1.
HA, [39]	Hardware Access flag update in stage 1 translations from EL0 and EL1. The possible values are:

0	Stage 1 Access flag update disabled.
1	Stage 1 Access flag update enabled.

Configurations

RW fields in this register reset to *UNKNOWN* values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

TCR_EL2, Translation Control Register, EL2

The TCR_EL2 controls translation table walks required for stage 1 translation of a memory access from EL2 and holds cacheability and shareability information.

Bit field descriptions

TCR_EL2 is a 64-bit register.

TCR_EL2 is part of:

- The Virtual memory control registers functional group.
- The Hypervisor and virtualization registers functional group.

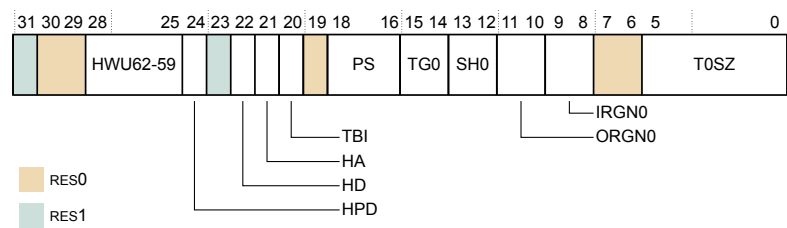


Figure 81: TCR_EL2 bit assignments



Note:
Bits[28:21], architecturally defined, are implemented in the *core*.

HD, [22]	Dirty bit update. The possible values are:
0	Dirty bit update is disabled.
1	Dirty bit update is enabled.
HA, [21]	Stage 1 Access flag update. The possible values are:
0	Stage 1 Access flag update is disabled.
1	Stage 1 Access flag update is enabled.

Configurations

When the Virtualization Host Extension is activated, TCR_EL2 has the same bit assignments as TCR_EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

TCR_EL3, Translation Control Register, EL3

The TCR_EL3 controls translation table walks required for stage 1 translation of memory accesses from EL3 and holds cacheability and shareability information for the accesses.

Bit field descriptions

TCR_EL3 is a 32-bit register and is part of the Virtual memory control registers functional group.

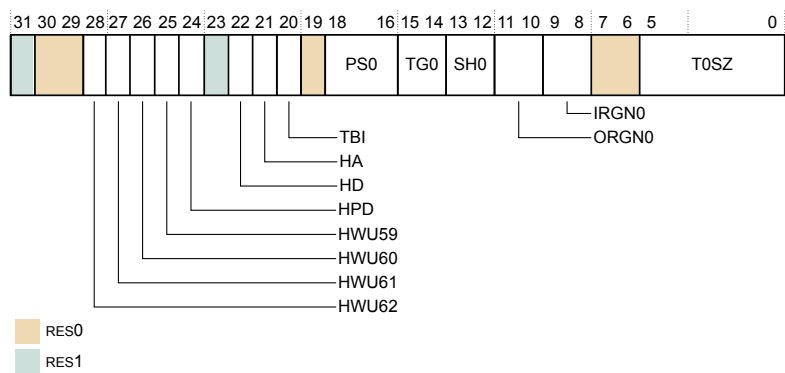


Figure 82: TCR_EL3 bit assignments



Note:
Bits[28:21], architecturally defined, are implemented in the *core*.

HD, [22]	Dirty bit update. The possible values are:	
	0	Dirty bit update is disabled.
	1	Dirty bit update is enabled.
HA, [21]	Stage 1 Access flag update. The possible values are:	
	0	Stage 1 Access flag update is disabled.
	1	Stage 1 Access flag update is enabled.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm®*

Architecture Reference Manual Armv8, for Armv8-A architecture profile.

TTBR1_EL1, Translation Table Base Register 1, EL1

The TTBR1_EL1 holds the base address of translation table 1, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses at EL0 and EL1.

Bit field descriptions

TTBR1_EL1 is a 64-bit register.

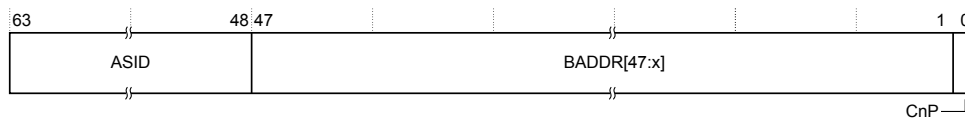


Figure 83: TTBR1_EL1 bit assignments

ASID, [63:48]

An ASID for the *translation table* base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

BADDR[47:x], [47:1]

Translation table base address, bits[47:x]. Bits [x-1:0] are *res0*.

x is based on the value of TCR_EL1.T0SZ, the stage of translation, and the memory translation granule size.

For instructions on how to calculate it, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The value of x determines the required alignment of the translation table, that must be aligned to 2^x bytes.

If bits [x-1:1] are not all zero, this is a misaligned Translation Table Base Address. Its effects are *constrained unpredictable*, where bits [x-1:1] are treated as if all the bits are zero. The value *read* back from those bits is the value written.

CnP, [0]

Common not Private. The possible values are:

0	CnP is not supported.
1	CnP is supported.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

TTBR1_EL2, Translation Table Base Register 1, EL2

TTBR1_EL2 has the same format and contents as TTBR1_EL1.

See [TTBR1_EL1, Translation Table Base Register 1, EL1](#).

VDSIR_EL2, Virtual Deferred Interrupt Status Register, EL2

The VDSIR_EL2 records that a virtual SError interrupt has been consumed by an ESB instruction executed at Non-secure EL1.

Bit field descriptions

VDSIR_EL2 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

Configurations

See [VDSIR_EL2 at EL1 using AArch64](#).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

.

VDSIR_EL2 at EL1 using AArch64

VDSIR_EL2 has a specific format when written at EL1.

The following figure shows the VDSIR_EL2 bit assignments when written at EL1 using [AArch64](#):

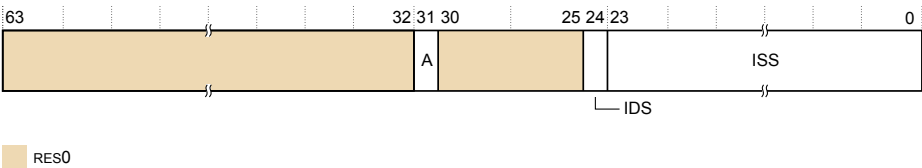


Figure 84: VDSIR_EL2 at EL1 using

RES0, [63:32]

RES0

Reserved.

A, [31]

Set to 1 when ESB defers an asynchronous SError interrupt.

RES0, [30:25]

RES0

Reserved.

IDS, [24]

Contains the value from VSESR_EL2.IDS.

ISS, [23:0]

Contains the value from VSESR_EL2, bits[23:0].

VESR_EL2, Virtual SError Exception Syndrome Register

The VESR_EL2 provides the syndrome value reported to software on taking a virtual SError interrupt exception.

Bit field descriptions

VESR_EL2 is a 64-bit register, and is part of:

- The Exception and *fault* handling registers functional group.
- The Virtualization registers functional group.

If the virtual SError interrupt is taken to EL1, VESR_EL2 provides the syndrome value reported in ESR_EL1.

VESR_EL2 bit assignments

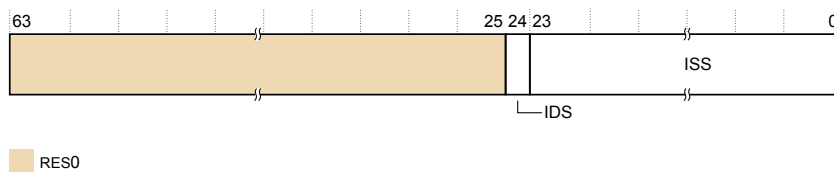


Figure 85: VESR_EL2 bit assignments

RES0, [63:25]

RES0

Reserved.

IDS, [24]

Indicates whether the deferred SError interrupt was of an *IMPLEMENTATION DEFINED* type. See ESR_EL1.IDS for a description of the functionality.

On taking a virtual SError interrupt to EL1 using because HCR_EL2.VSE == 1, ESR_EL1[24] is set to VESR_EL2.IDS.

ISS, [23:0]

Syndrome information. See ESR_EL1.ISS for a description of the functionality.

On taking a virtual SError interrupt to EL1 using *AArch32* due to HCR_EL2.VSE == 1, ESR_EL1 [23:0] is set to VESR_EL2.ISS.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

VTCTR_EL2, Virtualization Translation Control Register, EL2

The VTCTR_EL2 controls the translation table walks required for the stage 2 translation of memory accesses from Non-secure EL0 and EL1.

Bit field descriptions

It also holds cacheability and shareability information for the accesses.

VTCR_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

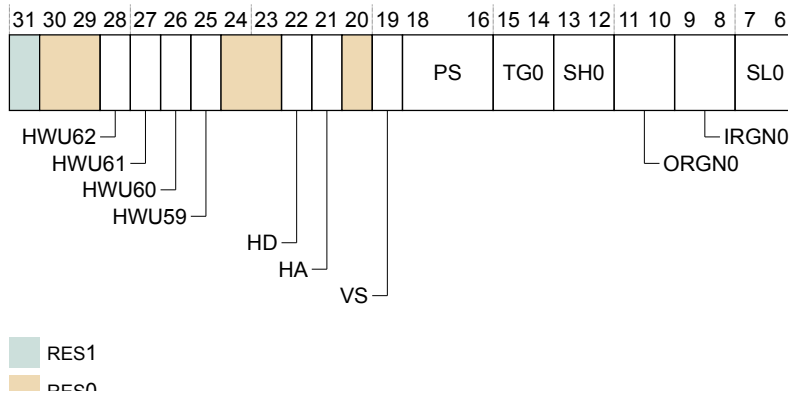


Figure 86: VTCR_EL2 bit assignments



Note:

Bits[28:25] and bits[22:21], architecturally defined, are implemented in the *core*.

TG0, [15:14]

TTBR0_EL2 granule size. The possible values are:

00	4KB.
01	64KB.
10	16KB.
11	Reserved.

All other values are not supported.

Configurations

RW fields in this register reset to architecturally *UNKNOWN* values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

VTTBR_EL2, Virtualization Translation Table Base Register, EL2

VTTBR_EL2 holds the base address of the translation table for the stage 2 translation of memory accesses from Non-secure EL0 and EL1.

Bit field descriptions

VTTBR_EL2 is a 64-bit register.

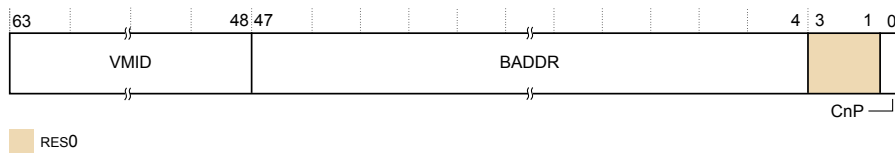


Figure 87: VTTBR_EL2 bit assignments

CnP, [0]

Common not Private. The possible values are:

- | | |
|----------|-----------------------|
| 0 | CnP is not supported. |
| 1 | CnP is supported. |

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter

13

Error system registers

Topics:

- [Error system register summary](#)
- [ERR0ADDR, Error Record Address Register](#)
- [ERR0CTLR, Error Record Control Register](#)
- [ERR0FR, Error Record Feature Register](#)
- [ERR0MISC1, Error Record Miscellaneous Register 1](#)
- [ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register](#)
- [ERR0PFGCTL, Error Pseudo Fault Generation Control Register](#)
- [ERR0PFGF, Error Pseudo Fault Generation Feature Register](#)
- [ERR0STATUS, Error Record Primary Status Register](#)

This chapter describes the error registers accessed by the AArch64 error registers.

Error system register summary

This section identifies the ERR0* core error record registers accessed by the AArch64 ERX* error registers.

The ERR0* registers are agnostic to the architectural state. For example, this means that for ERRSELR==0 and ERRSELR_EL1==0, ERXPGFGR and ERXPGGF_EL1 will both access ERR0PFGF.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The following table describes the architectural error record registers.

Table 53: Architectural error system register summary

Register mnemonic	Size	Register name	Access aliases from <i>AArch64</i>
ERR0ADDR	64	ERR0ADDR, Error Record Address Register	ERXADDR_EL1, Selected Error Record Address Register, EL1
ERR0CTLR	64	ERR0CTLR, Error Record Control Register	ERXCTLR_EL1, Selected Error Record Control Register, EL1
ERR0FR	64	ERR0FR, Error Record Feature Register	ERXFR_EL1, Selected Error Record Feature Register, EL1
ERR0MISC0	64	ERR0MISC0, Error Record Miscellaneous Register 0	ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1
ERR0MISC1	64	ERR0MISC1, Error Record Miscellaneous Register 1	ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1
ERR0STATUS	32	ERR0STATUS, Error Record Primary Status Register	ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1

The following table describes the error record registers that are *IMPLEMENTATION DEFINED*.

Table 54: IMPLEMENTATION DEFINED error system register summary

Register mnemonic	Size	Register name	Access aliases from
ERR0PFGCDN	32	ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register	ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1
ERR0PFGCTL	32	ERR0PFGCTL, Error Pseudo Fault Generation Control Register	ERXPFGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1

Register mnemonic	Size	Register name	Access aliases from
ERR0PFGF	32	ERR0PFGF, Error Pseudo Fault Generation Feature Register	ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1

ERR0ADDR, Error Record Address Register

The ERR0ADDR stores the address that is associated to an error that is recorded.

Bit field descriptions

ERR0ADDR is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

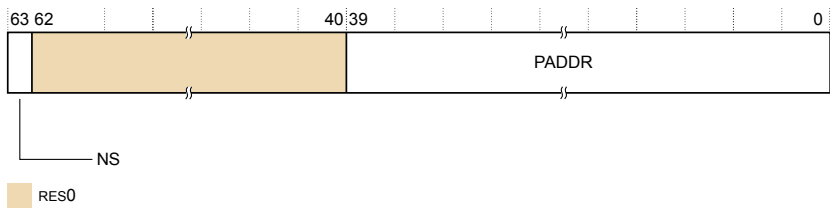


Figure 88: ERR0ADDR bit assignments

NS, [63]	Non-secure attribute. The possible values are:	
	0	The <i>physical address</i> is Secure.
	1	The is Non-secure.
RES0, [62:40]	RES0	Reserved.
PADDR, [39:0]	Physical address.	
Configurations	ERR0ADDR resets to <i>UNKNOWN</i> . When ERRSELR.SEL==0, this register is accessible from ERXADDR_EL1, Selected Error Record Address Register, EL1 .	

ERR0CTLR, Error Record Control Register

The ERR0CTLR contains enable bits for the node that writes to this record:

- Enabling error detection and correction.
- Enabling an error recovery interrupt.
- Enabling a *fault* handling interrupt.
- Enabling error recovery reporting as a *read* or write error response.

Bit field descriptions

ERR0CTLR is a 64-bit register and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

ERR0CTLR resets to CFI [8], FI [3], and UI [2] are *UNKNOWN*. The rest of the register is *RES0*.

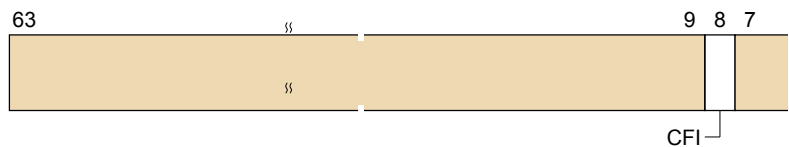




Figure 89: ERR0CTLR bit assignments

RES0, [63:9]	RES0	Reserved.				
CFI, [8]	<p>Fault handling interrupt for corrected errors enable.</p> <p>The handling interrupt is generated when one of the standard CE counters on ERR0MISC0 overflows and the overflow bit is set. The possible values are:</p> <table><tr><td>0</td><td>Fault handling interrupt not generated for corrected errors.</td></tr><tr><td>1</td><td>Fault handling interrupt generated for corrected errors.</td></tr></table> <p>The interrupt is generated even if the error status is overwritten because the error record already records a higher priority error.</p> <p> Note: This applies to both reads and writes.</p>		0	Fault handling interrupt not generated for corrected errors.	1	Fault handling interrupt generated for corrected errors.
0	Fault handling interrupt not generated for corrected errors.					
1	Fault handling interrupt generated for corrected errors.					
RES0, [7:4]	RES0	Reserved.				
FI, [3]	<p>Fault handling interrupt enable.</p> <p>The handling interrupt is generated for all detected Deferred errors and Uncorrected errors. The possible values are:</p> <table><tr><td>0</td><td>Fault handling interrupt disabled.</td></tr><tr><td>1</td><td>Fault handling interrupt enabled.</td></tr></table>		0	Fault handling interrupt disabled.	1	Fault handling interrupt enabled.
0	Fault handling interrupt disabled.					
1	Fault handling interrupt enabled.					
UI, [2]	<p>Uncorrected error recovery interrupt enable. When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred. The possible values are:</p>					

	0	Error recovery interrupt disabled.
	1	Error recovery interrupt enabled.
		Note: Applies to both s and writes.
RES0, [1]	RES0	Reserved.
ED, [0]	Error Detection and correction enable. The possible values are:	
	0	Error detection and correction disabled.
	1	Error detection and correction enabled.
Configurations	This register is accessible from the following registers when ERRSELR.SEL==0: ERXCTLR_EL1 , Selected Error Record Control Register , EL1 .	

ERR0FR, Error Record Feature Register

The ERR0FR defines which of the common architecturally defined features are implemented and, of the implemented features, which are software programmable.

Bit field descriptions

ERR0FR is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group. The register is Read Only.

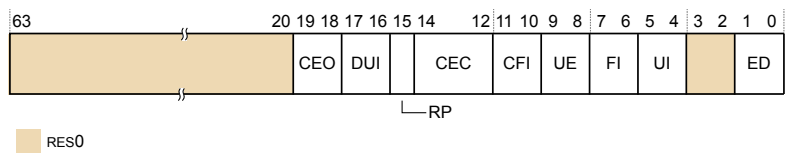


Figure 90: ERR0FR bit assignments

[63:20]	RES0	Reserved.
CEO, [19:18]	Corrected Error Overwrite. The value is:	
	00	Counts CE if a counter is implemented and keeps the previous error status. If the counter overflows,

		ERR0STATUS.OF is set to 1.
DUI, [17:16]	Error recovery interrupt for deferred errors. The value is:	
	00	The <i>core</i> does not support this feature.
RP, [15]	Repeat counter. The value is:	
	1	A first repeat counter and a second other counter are implemented. The repeat counter is the same size as the primary error counter.
CEC, [14:12]	Corrected Error Counter. The value is:	
	010	The node implements an 8-bit standard CE counter in ERR0MISC0[39:32].
CFI, [11:10]	Fault handling interrupt for corrected errors. The value is:	
	10	The node implements a control for enabling <i>fault</i> handling interrupts on corrected errors.
UE, [9:8]	In-band uncorrected error reporting. The value is:	
	01	The node implements in-band uncorrected error reporting, that is external <i>aborts</i> .
FI, [7:6]	Fault handling interrupt. The value is:	
	10	The node implements a handling interrupt and implements controls for enabling and disabling.
UI, [5:4]	Error recovery interrupt for uncorrected errors. The value is:	
	10	The node implements an error recovery interrupt and implements controls for enabling and disabling.
[3:2]	RES0	Reserved.
ED, [1:0]	Error detection and correction. The value is:	

10

The node implements controls for enabling or disabling error detection and correction.

Configurations

ERR0FR resets to 000000000000A9A2

ERR0FR is accessible from the following registers when ERRSELR.SEL==0:

[ERXFR_EL1, Selected Error Record Feature Register, EL1.](#)

ERR0MISC1, Error Record Miscellaneous Register 1

This register is unused in the Cortex®-A76 core and marked as *RES0*.

Configurations

When ERRSELR.SEL==0, ERR0MISC1 is accessible from [ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1.](#)

ERR0PFGCDN, Error Pseudo Fault Generation Count Down Register

ERR0PFGCDN is the Cortex®-A76 node register that generates one of the errors that are enabled in the corresponding ERR0PFGCTL register.

Bit field descriptions

ERR0PFGCDN is a 32-bit register and is RW.

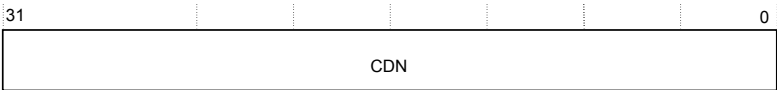


Figure 91: ERR0PFGCDN bit assignments

CDN, [31:0]

Count Down value. The reset value of the Error Generation Counter is used for the countdown.

Configurations

There are no configuration options.

ERR0PFGCDN resets to *UNKNOWN*.

When ERRSELR.SEL==0, ERR0PFGCDN is accessible from [ERXPFGCDN_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1.](#)

ERR0PFGCTL, Error Pseudo Fault Generation Control Register

The ERR0PFGCTL is the Cortex®-A76 node register that enables controlled fault generation.

Bit field descriptions

ERR0PFGCTL is a 32-bit *read*/write register.

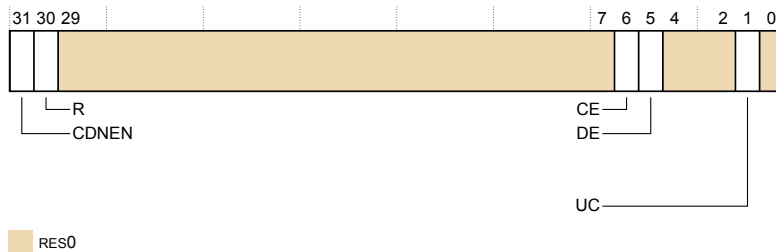


Figure 92: ERR0PFGCTL bit assignments

CDNEN, [31]

Count down enable. This bit controls transfers from the value that is held in the ERR0PFGCDN into the Error Generation Counter and enables this counter to start counting down. The possible values are:

- | | |
|----------|---|
| 0 | The Error Generation Counter is disabled. |
| 1 | The value that is held in the ERR0PFGCDN register is transferred into the Error Generation Counter. The Error Generation Counter counts down. |

R, [30]

Restartable bit. When it reaches 0, the Error Generation Counter restarts from the ERR0PFGCDN value or stops. The possible values are:

- | | |
|----------|---|
| 0 | When it reaches 0, the counter stops. |
| 1 | When it reaches 0, the counter reloads the value that is stored in ERR0PFGCDN and starts counting down again. |

[29:7]

Reserved, *RES0*.

CE, [6]

Corrected error generation enable. The possible values are:

- | | |
|----------|----------------------------------|
| 0 | No corrected error is generated. |
|----------|----------------------------------|

	1	A corrected error might be generated when the Error Generation Counter is triggered.
DE, [5]	Deferred Error generation enable. The possible values are:	
	0	No deferred error is generated.
	1	A deferred error might be generated when the Error Generation Counter is triggered.
[4:2]	Reserved, <i>RES0</i> .	
UC, [1]	Uncontainable error generation enable. The possible values are:	
	0	No uncontainable error is generated.
	1	An uncontainable error might be generated when the Error Generation Counter is triggered.
[0]	Reserved, <i>RES0</i> .	
Configurations	<p>There are no configuration notes.</p> <p>ERR0PFGCTL resets to 00000000.</p> <p>ERR0PFGCTL is accessible from the following registers when ERRSELR.SEL==0:</p> <ul style="list-style-type: none"> • ERXPFGCTL_EL1, Selected Error Pseudo Fault Generation Control Register, EL1. 	

ERR0PFGF, Error Pseudo Fault Generation Feature Register

The ERR0PFGF is the Cortex®-A76 node register that defines which fault generation features are implemented.

Bit field descriptions

ERR0PFGF is a 32-bit register and is RO.

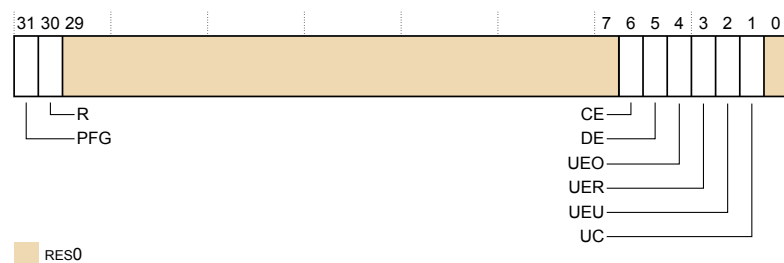


Figure 93: ERR0PFGF bit assignments

PFG, [31]

Pseudo Fault Generation. The value is:

1 The node implements a *fault* injection mechanism.

R, [30]

Restartable bit. When it reaches zero, the Error Generation Counter restarts from the ERR0PFGCDN value or stops. The value is:

1 This feature is controllable.

[29:7]

RES0 Reserved.

CE, [6]

Corrected Error generation. The value is:

1 This feature is controllable.

DE, [5]

Deferred Error generation. The value is:

1 This feature is controllable.

UEO, [4]

Latent or Restartable Error generation. The value is:

0 The node does not support this feature.

UER, [3]

Signaled or Recoverable Error generation. The value is:

0 The node does not support this feature.

UEU, [2]

Unrecoverable Error generation. The value is:

0 The node does not support this feature.

UC, [1]

Uncontainable Error generation. The value is:

1 This feature is controllable.

[0]

RES0 Reserved.

Configurations

There are no configuration notes.

ERR0PFGF resets to C0000062.

When ERRSELR.SEL==0, ERR0PFGF is accessible from [ERXPFGF_EL1, Selected Pseudo Fault Generation Feature Register, EL1](#).

ERR0STATUS, Error Record Primary Status Register

The ERR0STATUS contains information about the error record.

The register indicates:

- Whether any error has been detected.
- Whether any detected error was not corrected and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether a second error of the same type was detected before software handled the first error.
- Whether any error has been reported.
- Whether the other error record registers contain valid information.

Bit field descriptions

ERR0STATUS is a 32-bit register.

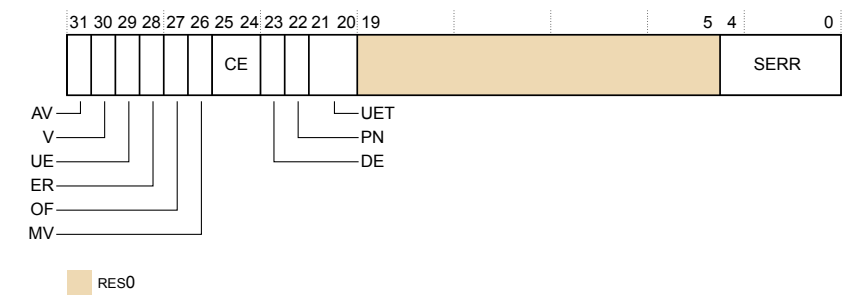


Figure 94: ERR0STATUS bit assignments

AV, [31]	Address Valid. The possible values are:
0	ERR0ADDR is not valid.
1	ERR0ADDR contains an address associated with the highest priority error recorded by this record.
V, [30]	Status Register valid. The possible values are:
0	ERR0STATUS is not valid.
1	ERR0STATUS is valid. At least one error has been recorded.

UE, [29]

Uncorrected error. The possible values are:

- | | |
|----------|--|
| 0 | No error that could not be corrected or deferred has been detected. |
| 1 | At least one error that could not be corrected or deferred has been detected. If error recovery interrupts are enabled, then the interrupt signal is asserted until this bit is cleared. |

ER, [28]

Error reported. The possible values are:

- | | |
|----------|--|
| 0 | No external <i>abort</i> has been reported. |
| 1 | The node has reported an external to the master that is in access or making a transaction. |

OF, [27]

Overflow. The possible values are:

- | | |
|----------|---|
| 0 | <ul style="list-style-type: none"> • If UE == 1, then no error status for an Uncorrected error has been discarded. • If UE == 0 and DE == 1, then no error status for a Deferred error has been discarded. • If UE == 0, DE == 0, and CE != 00, then:

The corrected error counter has not overflowed. |
| 1 | More than one error has occurred and so details of the other error have been discarded. |

MV, [26]

Miscellaneous Registers Valid. The possible values are:

- | | |
|----------|--|
| 0 | ERR0MISC0 and ERR0MISC1 are not valid. |
| 1 | This bit indicates that ERR0MISC0 contains additional information about any error that is recorded by this record. |

CE, [25:24]

Corrected error. The possible values are:

	00	No corrected error recorded.
	10	At least one corrected error recorded.
DE, [23]	Deferred error. The possible values are:	
	0	No errors were deferred.
	1	At least one error was not corrected and deferred by poisoning.
PN, [22]	Poison. The value is:	
	0	The Cortex [®] -A76 <i>core</i> cannot distinguish a poisoned value from a corrupted value.
UET, [21:20]	Uncorrected Error Type. The value is:	
	00	Uncontainable.
[19:5]	RES0.	Reserved.
SERR, [4:0]	Primary error code. The possible values are:	
	0	No error.
	1	Errors due to <i>fault</i> injection.
	2	ECC error from internal data buffer.
	6	ECC error on cache data RAM.
	7	ECC error on cache tag or <i>dirty</i> RAM.
	8	Parity error on <i>TLB</i> data RAM.
	12	Error response for a cache copyback.
	15	Deferred error from slave not supported at the consumer. For example, poisoned data received from a slave by a master that cannot defer the error further.

Configurations

There are no configuration notes.

ERR0STATUS resets to 00000000.

ERR0STATUS is accessible from the following registers
when ERRSEL.RSEL==0:

- [ERXSTATUS_EL1](#), Selected Error Record Primary Status Register, EL1.

Chapter

14

GIC registers

Topics:

- [CPU interface registers](#)
- [AArch64 physical GIC CPU interface system register summary](#)
- [ICC_AP0R0_EL1, Interrupt Controller Active Priorities Group 0 Register 0, EL1](#)
- [ICC_AP1R0_EL1, Interrupt Controller Active Priorities Group 1 Register 0 EL1](#)
- [ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1](#)
- [ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1](#)
- [ICC_CTLR_EL1, Interrupt Controller Control Register, EL1](#)
- [ICC_CTLR_EL3, Interrupt Controller Control Register, EL3](#)
- [ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1](#)
- [ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2](#)
- [ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3](#)
- [AArch64 virtual GIC CPU interface register summary](#)
- [ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Register 0, EL1](#)
- [ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Register 0, EL1](#)

This chapter describes the GIC registers.

- [ICV_BPR0_EL1](#), Interrupt Controller Virtual Binary Point Register 0, EL1
- [ICV_BPR1_EL1](#), Interrupt Controller Virtual Binary Point Register 1, EL1
- [ICV_CTLR_EL1](#), Interrupt Controller Virtual Control Register, EL1
- [AArch64 virtual interface control system register summary](#)
- [ICH_AP0R0_EL2](#), Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2
- [ICH_AP1R0_EL2](#), Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2
- [ICH_HCR_EL2](#), Interrupt Controller Hyp Control Register, EL2
- [ICH_VMCR_EL2](#), Interrupt Controller Virtual Machine Control Register, EL2
- [ICH_VTR_EL2](#), Interrupt Controller VGIC Type Register, EL2

CPU interface registers

Each CPU interface block provides the interface for the Cortex®-A76 core that interfaces with a GIC distributor within the system.

The Cortex®-A76 *core* only supports system register access to the *GIC* CPU interface registers. The following table lists the three types of CPU interface system registers supported in the Cortex®-A76 .

Table 55: CPU interface system register types supported in the Cortex®-A76 .

Register prefix	Register type
ICC	Physical CPU interface system registers.
ICV	Virtual CPU interface system registers.
ICH	Virtual interface control system registers.

Access to virtual CPU interface system registers is only possible at Non-secure EL1.

Access to ICC registers or the equivalent ICV registers is determined by HCR_EL2. See [HCR_EL2, Hypervisor Configuration Register, EL2](#).

For more information on the CPU interface, see the *Arm® Generic Interrupt Controller Architecture Specification*.

AArch64 physical GIC CPU interface system register summary

The following table lists the AArch64 physical GIC CPU interface system registers that have *IMPLEMENTATION DEFINED* bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of *AArch64* physical *GIC* CPU interface system registers.

Table 56: physical CPU interface system register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICC_AP0R03EL1	0	12	8	4	RW		ICC_AP0R0_EL1 , Interrupt Controller Active Priorities Group 0 Register 0, EL1
ICC_AP1R03EL1	0	12	9	0	RW		ICC_AP1R0_EL1 , Interrupt Controller Active Priorities Group 1 Register 0 EL1

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICC_BPR0_EL1	0	12	8	3	RW		ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1
ICC_BPR1_EL1	0	12	12	3	RW		ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1
ICC_CTLR_EL1	0	12	12	4	RW		ICC_CTLR_EL1, Interrupt Controller Control Register, EL1
ICC_CTLR_EL3	6	12	12	4	RW		ICC_CTLR_EL3, Interrupt Controller Control Register, EL3
ICC_SRE_EL1	0	12	12	5	RW		ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1
ICC_SRE_EL2	4	12	9	5	RW		ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2
ICC_SRE_EL3	6	12	12	5	RW		ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3

ICC_AP0R0_EL1, Interrupt Controller Active Priorities Group 0 Register 0, EL1

The ICC_AP0R0_EL1 provides information about Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The [GIC](#) system registers functional group.
- The control registers functional group.

The [core](#) implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_AP1R0_EL1, Interrupt Controller Active Priorities Group 1 Register 0, EL1

The ICC_AP1R0_EL1 provides information about Group 1 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The [GIC](#) system registers functional group.
- The control registers functional group.

The [core](#) implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1

ICC_BPR0_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0_EL1 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The control registers functional group.

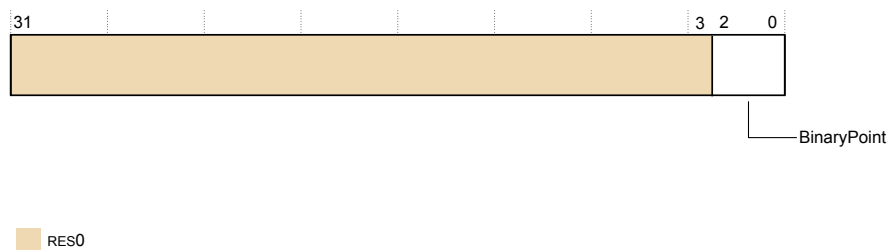


Figure 95: ICC_BPR0_EL1 bit assignments

RES0, [31:3]	RES0	Reserved.
BinaryPoint, [2:0]	The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value that is implemented is: 2	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1

ICC_BPR1_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Bit field descriptions

ICC_BPR1_EL1 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The control registers functional group.

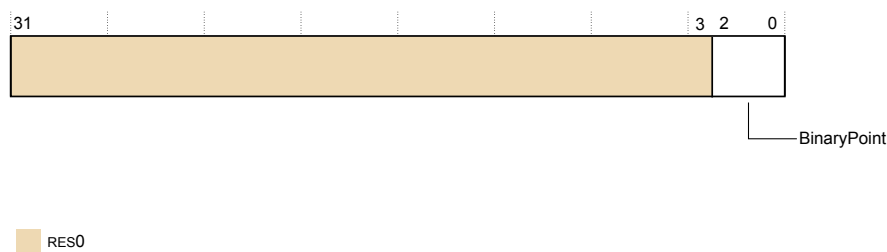


Figure 96: ICC_BPR1_EL1 bit assignments

RES0, [31:3]	<i>RES0</i>	Reserved.
BinaryPoint, [2:0]	<p>The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.</p> <p>The minimum value implemented of ICC_BPR1_EL1 Secure register is 2.</p> <p>The minimum value implemented of ICC_BPR1_EL1 Non-secure register is 3.</p>	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_CTLR_EL1, Interrupt Controller Control Register, EL1

ICC_CTLR_EL1 controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_CTLR_EL1 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The control registers functional group.

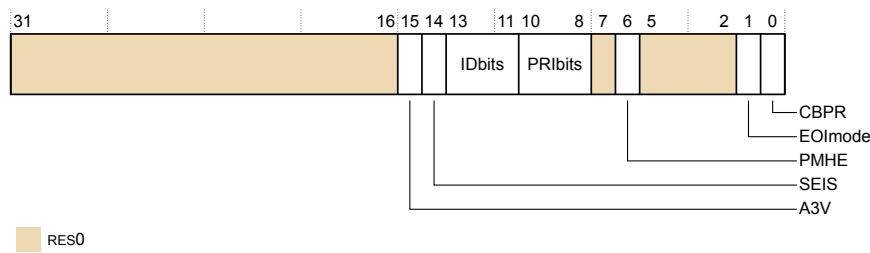


Figure 97: ICC_CTLR_EL1 bit assignments

RES0, [31:16]	<i>RES0</i>	Reserved.
A3V, [15]	Affinity 3 Valid. The value is:	

	1	The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.
SEIS, [14]	SEI Support. The value is:	
	0	The CPU interface logic does not support local generation of SEIs.
IDbits, [13:11]	Identifier bits. The value is:	
	0	The number of physical interrupt identifier bits supported is 16 bits.
	This field is an alias of ICC_CTLR_EL3.IDbits.	
PRibits, [10:8]	Priority bits. The value is:	
	4	The <i>core</i> supports 32 levels of physical priority with 5 priority bits.
RES0, [7]	RES0	Reserved.
PMHE, [6]	Priority Mask Hint Enable. This bit is <i>read</i> only and is an alias of ICC_CTLR_EL3.PMHE. The possible values are:	
	0	Disables use of ICC_PMR as a hint for interrupt distribution.
	1	Enables use of ICC_PMR as a hint for interrupt distribution.
RES0, [5:2]	RES0	Reserved.
EOImode, [1]	End of interrupt mode for the current security state. The possible values are:	
	0	ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are <i>UNPREDICTABLE</i> .
	1	ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides

		interrupt deactivation functionality.
CBPR, [0]	Common Binary Point Register. Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupt. The possible values are:	
	0	ICC_BPR0 determines the preemption group for Group 0 interrupts. ICC_BPR1 determines the preemption group for Group 1 interrupts.
	1	ICC_BPR0 determines the preemption group for Group 0 and Group 1 interrupts.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_CTLR_EL3, Interrupt Controller Control Register, EL3

ICC_CTLR_EL3 controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_CTLR_EL3 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Security registers functional group.
- The control registers functional group.

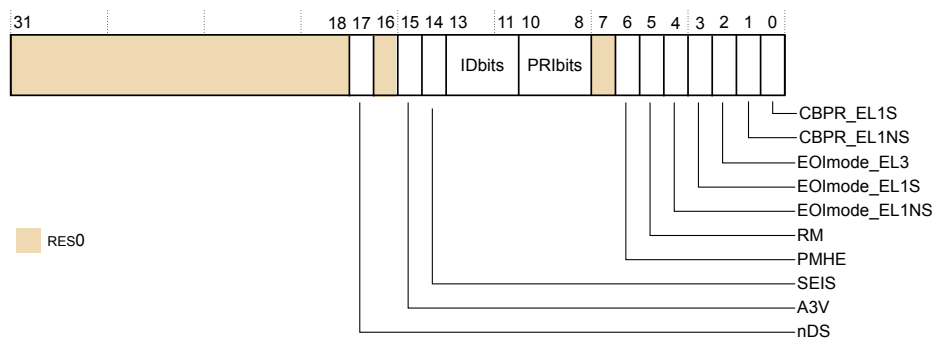


Figure 98: ICC_CTLR_EL3 bit assignments

RES0, [31:18]

RES0

Reserved.

nDS, [17]

Disable Security not supported. Read-only and writes are *ignored*. The value is:

	1	The CPU interface logic does not support disabling of security, and requires that security is not disabled.
RES0, [16]	RES0	Reserved.
A3V, [15]		Affinity 3 Valid. This bit is <i>RAO</i> /WI.
SEIS, [14]	SEI Support. The value is:	
	0	The CPU interface logic does not support generation of SEIs.
IDbits, [13:11]	Identifier bits. The value is:	
	0	The number of physical interrupt identifier bits supported is 16 bits.
	This field is an alias of ICC_CTLR_EL3.IDbits.	
PRBits, [10:8]	Priority bits. The value is:	
	4	The <i>core</i> supports 32 levels of physical priority with 5 priority bits. Accesses to ICC_AP0R{1—3} and ICC_AP1R{1—3} are <i>UNDEFINED</i> .
RES0, [7]		Reserved, <i>RES0</i> .
PMHE, [6]	Priority Mask Hint Enable. The possible values are:	
	0	Disables use of ICC_PMR as a hint for interrupt distribution.
	1	Enables use of ICC_PMR as a hint for interrupt distribution.
RM, [5]		Routing Modifier. This bit is <i>RAZ</i> /WI.
EOImode_EL1NS, [4]	EOI mode for interrupts handled at Non-secure EL1 and EL2.	

	Controls whether a write to an End of Interrupt register also deactivates the interrupt.
EOImode_EL1S, [3]	EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt.
EOImode_EL3, [2]	EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.
CBPR_EL1NS, [1]	Common Binary Point Register, EL1 Non-secure. Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.
CBPR_EL1S, [0]	Common Binary Point Register, EL1 Secure. Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupt at EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1

ICC_SRE_EL1 controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

Bit field descriptions

ICC_SRE_EL1 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The control registers functional group.

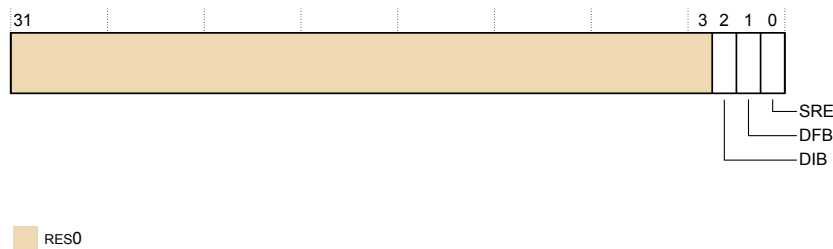


Figure 99: ICC_SRE_EL1 bit assignments

RES0, [31:3]	<i>RES0</i>	Reserved.
DIB, [2]		Disable <i>IRQ</i> bypass. The possible values are: 0 bypass enabled.

	1	bypass disabled.
	This bit is an alias of ICC_SRE_EL3.DIB	
DFB, [1]	Disable <i>FIQ</i> bypass. The possible values are:	
	0	bypass enabled.
	1	bypass disabled.
	This bit is an alias of ICC_SRE_EL3.DFB	
SRE, [0]	System Register Enable. The value is:	
	1	The System register interface for the current Security state is enabled.
	This bit is <i>RAO</i> /WI. The <i>core</i> only supports a system register interface to the CPU interface.	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2

ICC_SRE_EL2 controls whether the system register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Bit field descriptions

ICC_SRE_EL2 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Virtualization registers functional group.
- The control registers functional group.

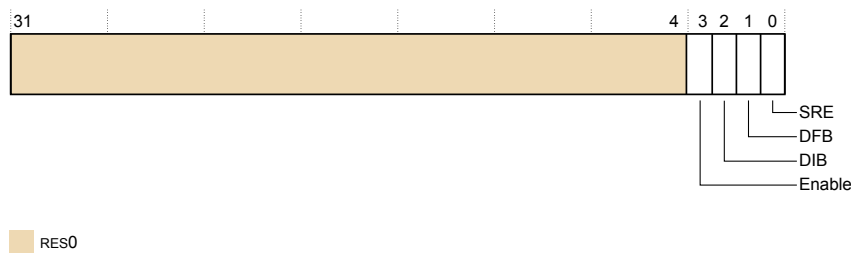


Figure 100: ICC_SRE_EL2 bit assignments

RES0, [31:4]	<i>RES0</i>	Reserved.
Enable, [3]	Enables lower <i>Exception level</i> access to ICC_SRE_EL1. The value is:	

	1	Non-secure EL1 accesses to ICC_SRE_EL1 do not trap to EL2.
	This bit is <i>RAO</i> /WI.	
DIB, [2]	Disable <i>IRQ</i> bypass. The possible values are:	
	0	bypass enabled.
	1	bypass disabled.
	This bit is an alias of ICC_SRE_EL3.DIB	
DFB, [1]	Disable <i>FIQ</i> bypass. The possible values are:	
	0	bypass enabled.
	1	bypass disabled.
	This bit is an alias of ICC_SRE_EL3.DFB	
SRE, [0]	System Register Enable. The value is:	
	1	The System register interface for the current Security state is enabled.
	This bit is /WI. The <i>core</i> only supports a system register interface to the CPU interface.	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3

ICC_SRE_EL3 controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Bit field descriptions

ICC_SRE_EL3 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Security registers functional group.
- The control registers functional group.

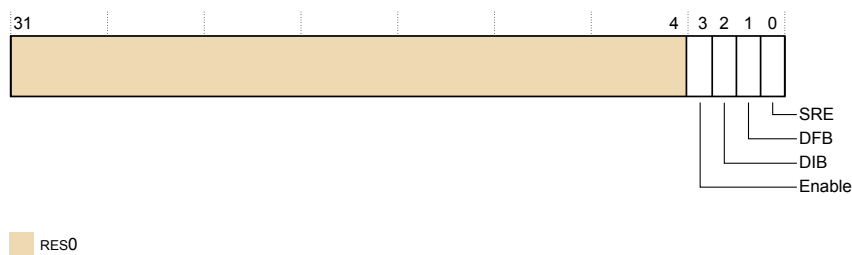


Figure 101: ICC_SRE_EL3 bit assignments

RES0, [31:4]	RES0	Reserved.
Enable, [3]	Enables lower <i>Exception level</i> access to ICC_SRE_EL1 and ICC_SRE_EL2. The value is:	
	1	<ul style="list-style-type: none">Secure EL1 accesses to Secure ICC_SRE_EL1 do not trap to EL3.EL2 accesses to Non-secure ICC_SRE_EL1 and ICC_SRE_EL2 do not trap to EL3.Non-secure EL1 accesses to ICC_SRE_EL1 do not trap to EL3.
	This bit is <i>RAO</i> /WI.	
DIB, [2]	Disable <i>IRQ</i> bypass. The possible values are:	
	0	bypass enabled.
	1	bypass disabled.
DFB, [1]	Disable <i>FIQ</i> bypass. The possible values are:	
	0	bypass enabled.
	1	bypass disabled.
SRE, [0]	System Register Enable. The value is:	
	1	The System register interface for the current Security state is enabled.
	This bit is /WI. The <i>core</i> only supports a system register interface to the CPU interface.	

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

AArch64 virtual GIC CPU interface register summary

The following table describes the AArch64 virtual GIC CPU interface system registers that have *IMPLEMENTATION DEFINED* bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of *AArch64* virtual *GIC* CPU interface system registers.

Table 57: virtual CPU interface register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICV_AP0R03EL1	0	12	8	4		RW	ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0, Register 0, EL1
ICV_AP1R03EL1	0	12	9	0		RW	ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1, Register 0, EL1
ICV_BRP0_BL1	0	12	8	3		RW	ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1
ICV_BPR1_BL1	0	12	12	3		RW	ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1
ICV_CTLR_EL1	0	12	12	4		RW	ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1

ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Register 0, EL1

The ICV_AP0R0_EL1 register provides information about virtual Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of the virtual *GIC* system registers functional group.

The *core* implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Register 0, EL1

The ICV_AP1R0_EL1 register provides information about virtual Group 1 active priorities.

Bit descriptions

This register is a 32-bit register and is part of the virtual *GIC* system registers functional group.

The *core* implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1

ICV_BPR0_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0_EL1 is a 32-bit register and is part of the virtual *GIC* system registers functional group.

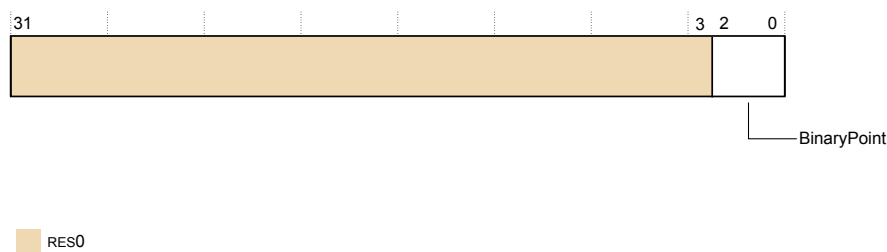


Figure 102: ICV_BPR0_EL1 bit assignments

RES0, [31:3]	Reserved, <i>RES0</i> .
BinaryPoint, [2:0]	The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value that is implemented is: 2

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1

ICV_BPR1_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Bit field descriptions

ICV_BPR1_EL1 is a 32-bit register and is part of the virtual *GIC* system registers functional group.

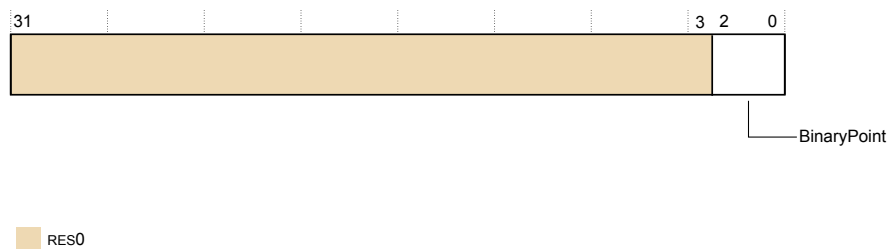


Figure 103: ICV_BPR1_EL1 bit assignments

RES0, [31:3]	<i>RES0</i>	Reserved.
BinaryPoint, [2:0]	The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value that is implemented of ICV_BPR1_EL1 Secure register is 2.	

The minimum value that is implemented of
ICV_BPR1_EL1 Non-secure register is 3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1

ICV_CTLR_EL1 controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Bit field descriptions

ICV_CTLR_EL1 is a 32-bit register and is part of the virtual *GIC* system registers functional group.

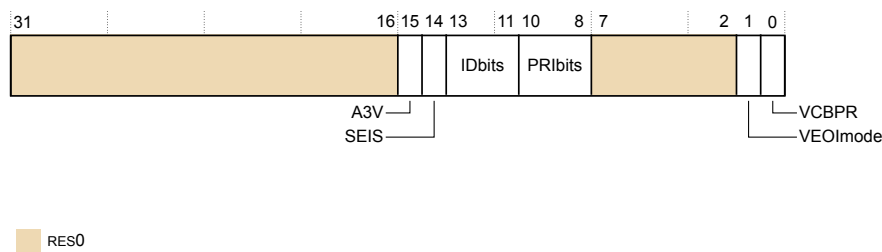


Figure 104: ICV_CTLR_EL1 bit assignments

RES0, [31:16]	<i>RES0</i>	Reserved.
A3V, [15]	Affinity 3 Valid. The value is: 1	The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.
SEIS, [14]	SEI Support. The value is: 0	The virtual CPU interface logic does not support local generation of SEIs.
IDbits, [13:11]	Identifier bits. The value is: 0	The number of physical interrupt identifier bits supported is 16 bits.
PRIbits, [10:8]	Priority bits. The value is:	

	4	Support 32 levels of physical priority (5 priority bits).
RES0, [7:2]	RES0	Reserved.
VEOImode, [1]	Virtual EOI mode. The possible values are:	
	0	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are <i>UNPREDICTABLE</i> .
	1	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.
VCBPR, [0]	Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts. The possible values are:	
	0	ICV_BPR0_EL1 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.
	1	ICV_BPR0_EL1 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts. Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 111. <i>Writes</i> to ICV_BPR1_EL1 are <i>ignored</i> .

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

AArch64 virtual interface control system register summary

The following table lists the AArch64 virtual interface control system registers that have *IMPLEMENTATION DEFINED* bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of *AArch64* virtual interface control system registers.

Table 58: virtual interface control system register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICH_AP0R03EL1	0	12	8	4		RW	ICH_AP0R0_EL2, Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2
ICH_AP1R03EL1	0	19	9	0		RW	ICH_AP1R0_EL2, Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2
ICH_HCR_EL2	4	12	11	0		RW	ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2
ICH_VTR_EL2	4	12	11	1		RO	ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2
ICH_VMCR3EL2	4	12	11	7		RW	ICH_VTR_EL2, Interrupt Controller V Type Register, EL2

ICH_AP0R0_EL2, Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2

The ICH_AP0R0_EL2 provides information about Group 0 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The [GIC](#) system registers functional group.
- The Virtualization registers functional group.
- The [host](#) interface control registers functional group.

The [core](#) implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICH_AP1R0_EL2, Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2

The ICH_AP1R0_EL2 provides information about Group 1 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The [GIC](#) system registers functional group.
- The Virtualization registers functional group.
- The [host](#) interface control registers functional group.

The [core](#) implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

00000000	No interrupt active. This is the reset value.
00000001	Interrupt active for priority 0.
00000002	Interrupt active for priority 8.
...	...
80000000	Interrupt active for priority F8.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2

ICH_HCR_EL2 controls the environment for VMs.

Bit field descriptions

ICH_HCR_EL2 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Virtualization registers functional group.
- The *host* interface control registers functional group.

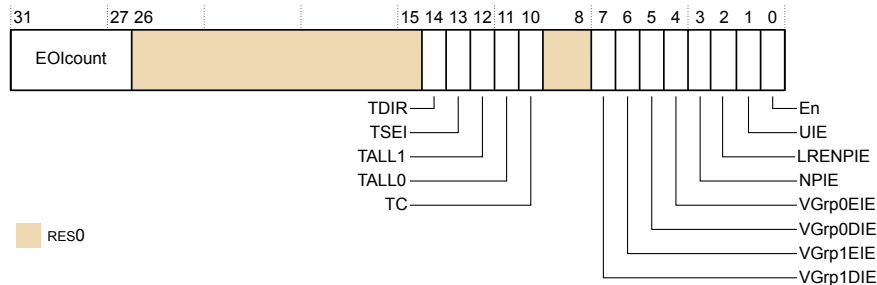


Figure 105: ICH_HCR_EL2 bit assignments

EOICount, [31:27]		Number of outstanding deactivates.
RES0, [26:15]	RES0	Reserved.
TDIR, [14]	0	Non-secure EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are not trapped to EL2, unless trapped by other mechanisms.
	1	Non-secure EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are trapped to EL2.
TSEI, [13]	0	Trap all locally generated SEIs. The value is: Locally generated SEIs do not cause a trap to EL2.
TALL1, [12]		Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2. The possible values are:

	0	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.
	1	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.
TALL0, [11]	Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2. The possible values are:	
	0	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.
	1	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.
TC, [10]	Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2. The possible values are:	
	0	Non-secure EL1 accesses to common registers proceed as normal.
	1	Non-secure EL1 accesses to common registers trap to EL2.
RES0, [9:8]	RES0	Reserved.
VGrp1DIE, [7]	VM Group 1 Disabled Interrupt Enable. The possible values are:	
	0	Maintenance interrupt disabled.
	1	Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0.
VGrp1EIE, [6]	VM Group 1 Enabled Interrupt Enable. The possible values are:	

	0 1	Maintenance interrupt disabled. Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1.
VGrp0DIE, [5]	VM Group 0 Disabled Interrupt Enable. The possible values are:	
	0 1	Maintenance interrupt disabled. Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0.
VGrp0EIE, [4]	VM Group 0 Enabled Interrupt Enable. The possible values are:	
	0 1	Maintenance interrupt disabled. Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1.
NPIE, [3]	No Pending Interrupt Enable. The possible values are:	
	0 1	Maintenance interrupt disabled. Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.
LRENPIE, [2]	List Register Entry Not Present Interrupt Enable. The possible values are:	
	0 1	Maintenance interrupt disabled. Maintenance interrupt is asserted while the EOICount field is not 0.
UIE, [1]	Underflow Interrupt Enable. The possible values are:	

En, [0]	0	Maintenance interrupt disabled.
	1	Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.
	Enable. The possible values are:	
	0	Virtual CPU interface operation disabled.
	1	Virtual CPU interface operation enabled.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2

ICH_VMCR_EL2 enables the hypervisor to save and restore the virtual machine view of the GIC state.

Bit field descriptions

ICH_VMCR_EL2 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Virtualization registers functional group.
- The *host* interface control registers functional group.

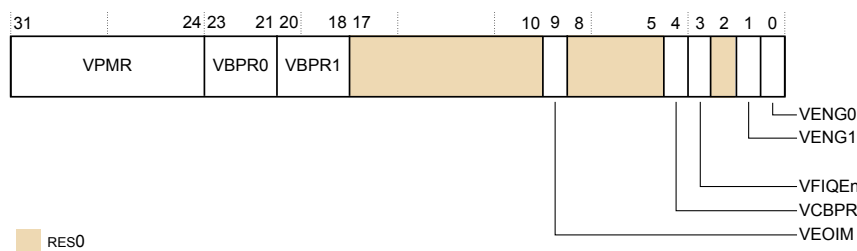


Figure 106: ICH_VMCR_EL2 bit assignments

VPMR, [31:24]	Virtual Priority Mask.	This field is an alias of ICV_PMR_EL1.Priority.
VBPR0, [23:21]	Virtual Binary Point Register, Group 0. The minimum value is:	
	2	This field is an alias of ICV_BPR0_EL1.BinaryPoint.

VBPR1, [20:18]

Virtual Binary Point Register, Group 1. The minimum value is:

3 This field is an alias of ICV_BPR1_EL1.BinaryPoint.

RES0, [17:10]

RES0 Reserved.

VEOIM, [9]

Virtual EOI mode. The possible values are:

0 ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are *UNPREDICTABLE*.

1 ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality.

This bit is an alias of ICV_CTLR_EL1.EOImode.

RES0, [8:5]

RES0 Reserved.

VCBPR, [4]

Virtual Common Binary Point Register. The possible values are:

0 ICV_BPR0_EL1 determines the preemption group for virtual Group 0 interrupts only.

ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.

1 ICV_BPR0_EL1 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts.

Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 111. *Writes* to ICV_BPR1_EL1 are *ignored*.

VFIQEn, [3]

Virtual enable. The value is:

	1	Group 0 virtual interrupts are presented as virtual s.
RES0, [2]	RES0	Reserved.
VENG1, [1]	Virtual Group 1 interrupt enable. The possible values are:	
	0	Virtual Group 1 interrupts are disabled.
	1	Virtual Group 1 interrupts are enabled.
VENG0, [0]	Virtual Group 0 interrupt enable. The possible values are:	
	0	Virtual Group 0 interrupts are disabled.
	1	Virtual Group 0 interrupts are enabled.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

ICH_VTR_EL2, Interrupt Controller VGIC Type Register, EL2

ICH_VTR_EL2 reports supported GIC virtualization features.

Bit field descriptions

ICH_VTR_EL2 is a 32-bit register and is part of:

- The *GIC* system registers functional group.
- The Virtualization registers functional group.
- The *host* interface control registers functional group.

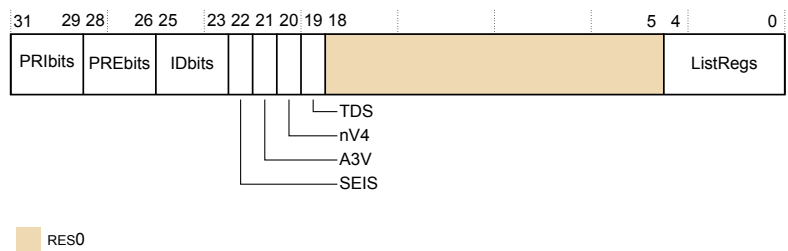


Figure 107: ICH_VTR_EL2 bit assignments

PRIbits, [31:29]	Priority bits. The number of virtual priority bits implemented, minus one.	
	4	Priority implemented is 5-bit.

PREbits, [28:26]

The number of virtual preemption bits implemented, minus one. The value is:

4

Virtual preemption implemented is 5-bit.

IDbits, [25:23]

The number of virtual interrupt identifier bits supported. The value is:

0

Virtual interrupt identifier bits that are implemented is 16-bit.

SEIS, [22]

SEI Support. The value is:

0

The virtual CPU interface logic does not support generation of SEIs.

A3V, [21]

Affinity 3 Valid. The value is:

1

The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

nV4, [20]

Direct injection of virtual interrupts not supported. The value is:

0

The CPU interface logic supports direct injection of virtual interrupts.

TDS, [19]

Separate trapping of Non-secure EL1 writes to ICH_DIR_EL1 supported. The value is:

1

Implementation supports ICH_HCR_EL2.TDIR.

RES0, [18:5]**RES0**

Reserved.

ListRegs, [4:0]**3**

The number of implemented List registers, minus one.

The *core* implements 4 list registers. Accesses to ICH_LR_EL2[x] (x>3) in *AArch64* are *UNDEFINED*.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® [Generic Interrupt Controller Architecture Specification](#)*.

Chapter

15

Advanced SIMD and floating-point registers

Topics:

- [AArch64 register summary](#)
- [FPCR, Floating-point Control Register](#)
- [FPSR, Floating-point Status Register](#)
- [MVFR0_EL1, Media and VFP Feature Register 0, EL1](#)
- [MVFR1_EL1, Media and VFP Feature Register 1, EL1](#)
- [MVFR2_EL1, Media and VFP Feature Register 2, EL1](#)
- [AArch32 register summary](#)
- [FPSCR, Floating-Point Status and Control Register](#)

This chapter describes the Advanced SIMD and floating-point registers.

AArch64 register summary

The core has several Advanced SIMD and floating-point system registers in the AArch64 execution state. Each register has a specific purpose, specific usage constraints, configurations, and attributes.

The following table gives a summary of the Cortex[®]-A76 *core Advanced SIMD* and *floating-point* system registers in the *AArch64* execution state.

Table 59: and system registers

Name	Type	Reset	Description
FPCR	RW	00000000	See FPCR , Floating-point Control Register.
FPSR	RW	<i>UNKNOWN</i>	See FPSR , Floating-point Status Register.
MVFR0_EL1	RO	10110222	See MVFR0_EL1 , Media and <i>VFP</i> Feature Register 0, EL1.
MVFR1_EL1	RO	13211111	See MVFR1_EL1 , Media and Feature Register 1, EL1.
MVFR2_EL1	RO	00000043	See MVFR2_EL1 , Media and Feature Register 2, EL1.

FPCR, Floating-point Control Register

The FPCR controls floating-point behavior.

Bit field descriptions

FPCR is a 32-bit register.

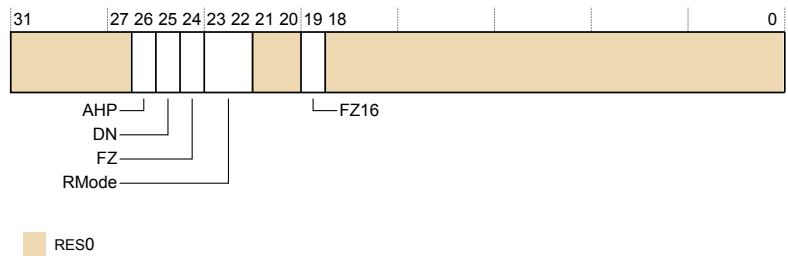


Figure 108: FPCR bit assignments

RES0, [31:27]	<i>res0</i>	Reserved.
AHP, [26]	0	IEEE half-precision format selected. This is the reset value.
	1	Alternative half-precision format selected.
DN, [25]		Default <i>NaN</i> mode control bit. The possible values are:

	0	operands propagate through to the output of a <i>floating-point</i> operation. This is the reset value.
	1	Any operation involving one or more s returns the Default .
FZ, [24]	Flush-to-zero mode control bit. The possible values are:	
	0	Flush-to-zero mode disabled. Behavior of the system is fully compliant with the IEEE 754 standard. This is the reset value.
	1	Flush-to-zero mode enabled.
RMode, [23:22]	Rounding Mode control field. The encoding of this field is:	
	00	<i>Round to Nearest</i> (RN) mode. This is the reset value.
	01	<i>Round towards Plus Infinity</i> (RP) mode.
	10	<i>Round towards Minus Infinity</i> (RM) mode.
	11	<i>Round towards Zero</i> (RZ) mode.
RES0, [21:20]	<i>res0</i>	Reserved.
FZ16, [19]	Flush-to-zero mode control bit on half-precision data-processing instructions. The possible values are:	
	0	Flush-to-zero mode disabled. Behavior of the system is fully compliant with the IEEE 754 standard. This is the default value.
	1	Flush-to-zero mode enabled.
RES0, [18:0]	<i>res0</i>	Reserved.
Configurations	The named fields in this register map to the equivalent fields in the <i>AArch32</i> FPSCR. See FPSCR, Floating-Point Status and Control Register .	

Usage constraints

Accessing the FPCR

To access the FPCR:

```
MRS <Xt>, FPCR ; Read FPCR into Xt
MSR FPCR, <Xt> ; Write Xt to FPCR
```

Register access is encoded as follows:

Table 60: FPCR access encoding

op0	op1	CRn	CRm	op2	
11	011	0100	0100	000	

Accessibility			This register is accessible as follows:		
EL0	EL1	EL1	EL2	EL3	EL3
	(NS)	(S)		(SCR.NS = 1)	(SCR.NS = 0)
RW	RW	RW	RW	RW	RW

FPSR, Floating-point Status Register

The FPSR provides floating-point system status information.

Bit field descriptions

FPSR is a 32-bit register.

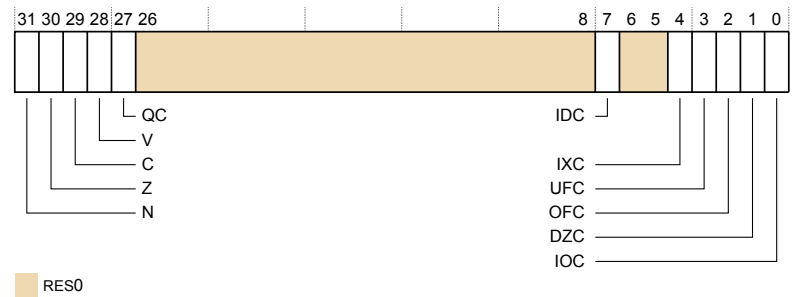


Figure 109: FPSR bit assignments

- N, [31]

Negative condition flag for *AArch32 floating-point* comparison operations. *AArch64* comparisons set the PSTATE.N flag instead.
- Z, [30]

Zero condition flag for comparison operations. comparisons set the PSTATE.Z flag instead.
- C, [29]

Carry condition flag for comparison operations. comparisons set the PSTATE.C flag instead.

V, [28]	Overflow condition flag for comparison operations. comparisons set the PSTATE.V flag instead.
QC, [27]	Cumulative saturation bit. This bit is set to 1 to indicate that an <i>Advanced SIMD</i> integer operation has saturated since a 0 was last written to this bit.
RES0, [26:8]	Reserved, <i>res0</i> .
IDC, [7]	Input Denormal cumulative <i>exception</i> bit. This bit is set to 1 to indicate that the Input Denormal has occurred since 0 was last written to this bit.
RES0, [6:5]	Reserved, <i>res0</i> .
IXC, [4]	Inexact cumulative bit. This bit is set to 1 to indicate that the Inexact has occurred since 0 was last written to this bit.
UFC, [3]	Underflow cumulative bit. This bit is set to 1 to indicate that the Underflow has occurred since 0 was last written to this bit.
OFC, [2]	Overflow cumulative bit. This bit is set to 1 to indicate that the Overflow has occurred since 0 was last written to this bit.
DZC, [1]	Division by Zero cumulative bit. This bit is set to 1 to indicate that the Division by Zero has occurred since 0 was last written to this bit.
IOC, [0]	Invalid Operation cumulative bit. This bit is set to 1 to indicate that the Invalid Operation has occurred since 0 was last written to this bit.
Configurations	The named fields in this register map to the equivalent fields in the FPSCR. See FPSCR, Floating-Point Status and Control Register .

Usage constraints

Accessing the FPSR

To access the FPSR:

```
MRS <Xt>, FPSR; Read FPSR into Xt
MSR FPSR, <Xt>; Write Xt to FPSR
```

Register access is encoded as follows:

Table 61: FPSR access encoding

op0	op1	CRn	CRm	op2
11	011	0100	0100	001

Accessibility

This register is accessible as follows:

EL0	EL1	EL1	EL2	EL3	EL3
	(NS)	(S)		(SCR.NS = 1)	(SCR.NS = 0)
RW	RW	RW	RW	RW	RW

MVFR0_EL1, Media and VFP Feature Register 0, EL1

The MVFR0_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR0_EL1 is a 32-bit register.

31	28:27	24:23	20:19	16:15	12:11	8	7	4	3	0
FPRound	FPSHVec	FPSqrt	FPDivide	FPTrap	FPDP	FPSP				SIMDReg

Figure 110: MVFR0_EL1 bit assignments

FPRound, [31:28]

Indicates the *rounding modes* supported by the *floating-point* hardware:

1 All s supported.

FPSHVec, [27:24]

Indicates the hardware support for short vectors:

0 Not supported.

FPSqrt, [23:20]

Indicates the hardware support for square root operations:

1 Supported.

FPDivide, [19:16]

Indicates the hardware support for divide operations:

1 Supported.

FPTrap, [15:12]

Indicates whether the hardware implementation supports *exception* trapping:

0 Not supported.

FPDP, [11:8]

Indicates the hardware support for double-precision operations:

2 Supported, *VFPv3* or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

FPSP, [7:4]

Indicates the hardware support for single-precision operations:

2 Supported, v3 or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

SIMDReg, [3:0]

Indicates support for the *Advanced* register bank:

2 Supported, 32 x 64-bit registers supported.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Configurations

There are no configuration notes.

Usage constraints

Accessing the MVFR0_EL1

To access the MVFR0_EL1:

```
MRS <Xt>, MVFR0_EL1 ; Read MVFR0_EL1 into Xt
```

Register access is encoded as follows:

Table 62: MVFR0_EL1 access encoding

op0	op1	CRn	CRm	op2
11	000	0000	0011	000

Accessibility

This register is accessible as follows:

EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

MVFR1_EL1, Media and VFP Feature Register 1, EL1

The MVFR1_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR1_EL1 is a 32-bit register.

31	28:27	24:23	20:19	16:15	12:11	8	7	4	3	0
SIMDFMAC	FPHP	SIMDHP	SIMDSP	SIMDInt	SIMDLS	FPDNaN		FPFtZ		

Figure 111: MVFR1_EL1 bit assignments

SIMDFMAC, [31:28]

Indicates whether the *Advanced* and *floating-point* unit supports fused multiply accumulate operations:

1 Implemented.

FPHP, [27:24]

Indicates whether the and unit supports half-precision conversion instructions:

3 Floating-point half precision conversion and data processing instructions implemented.

HP, [23:20]

Indicates whether the and unit supports half-precision conversion operations:

2 half precision conversion and data processing instructions implemented.

SP, [19:16]

Indicates whether the and unit supports single-precision operations:

1 Implemented.

Int, [15:12]

Indicates whether the and unit supports integer operations:

1 Implemented.

LS, [11:8]

Indicates whether the and unit supports load/store instructions:

1 Implemented.

FPDNaN, [7:4]

Indicates whether the hardware implementation supports only the *Default mode*:

1 Hardware supports propagation of values.

FPFtZ, [3:0]

Indicates whether the hardware implementation supports only the *Flush-to-zero mode* of operation:

1 Hardware supports full denormalized number arithmetic.

Configurations

There are no configuration notes.

Usage constraints**Accessing the MVFR1_EL1**

To access the MVFR1_EL1:

```
MRS <Xt>, MVFR1_EL1 ; Read MVFR1_EL1
into Xt
```

Register access is encoded as follows:

Table 63: MVFR1_EL1 access encoding

op0	op1	CRn	CRm	op2	
11	000	0000	0011	001	
Accessibility					
			This register is accessible as follows:		
EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

MVFR2_EL1, Media and VFP Feature Register 2, EL1

The MVFR2_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR2_EL1 is a 32-bit register.

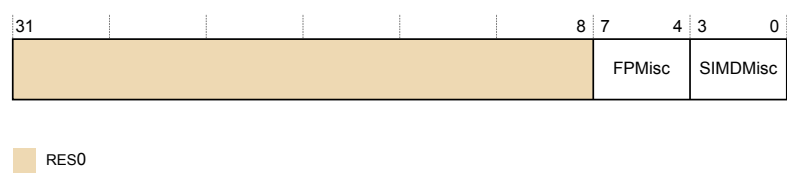


Figure 112: MVFR2_EL1 bit assignments

[31:8]	RES0	Reserved.
FPMisc, [7:4]	Indicates support for miscellaneous <i>floating-point</i> features.	4 Supports: <ul style="list-style-type: none">Floating-point selection.Floating-point Conversion to Integer with Directed Rounding modes.Floating-point Round to Integral Floating-point.Floating-point MaxNum and MinNum.
<i>SIMD</i> Misc, [3:0]	Indicates support for miscellaneous <i>Advanced</i> features.	3 Supports:

- Floating-point Conversion to Integer with Directed Rounding modes.
- Floating-point Round to Integral Floating-point.
- Floating-point MaxNum and MinNum.

Configurations

There are no configuration notes.

Usage constraints

Accessing the MVFR2_EL1

To access the MVFR2_EL1:

```
MRS <Xt>, MVFR2_EL1 ; Read MVFR2_EL1
into Xt
```

Register access is encoded as follows:

Table 64: MVFR2_EL1 access encoding

op0	op1	CRn	CRm	op2
11	000	0000	0011	010

Accessibility

This register is accessible as follows:

EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

AArch32 register summary

The core has one Advanced SIMD and floating-point system registers in the AArch32 execution state.

The following table gives a summary of the Cortex[®]-A76 *core Advanced SIMD* and *floating-point* system registers in the *AArch32* execution state.

Table 65: and system registers

Name	Type	Reset	Description
FPSCR	RW	UNKNOWN	See FPSCR, Floating-Point Status and Control Register .

See the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile* for information on permitted accesses to the and system registers.

FPSCR, Floating-Point Status and Control Register

The FPSCR provides floating-point system status information and control.

Bit field descriptions

FPSCR is a 32-bit register.

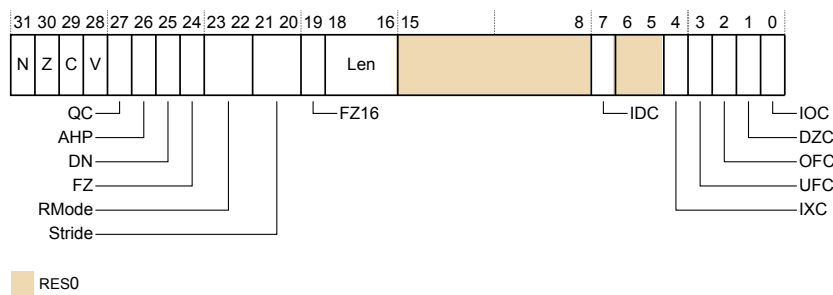


Figure 113: FPSCR bit assignments

N, [31]	Floating-point Negative condition code flag. Set to 1 if a <i>floating-point</i> comparison operation produces a less than result.				
Z, [30]	Floating-point Zero condition code flag. Set to 1 if a comparison operation produces an equal result.				
C, [29]	Floating-point Carry condition code flag. Set to 1 if a comparison operation produces an equal, greater than, or unordered result.				
V, [28]	Floating-point Overflow condition code flag. Set to 1 if a comparison operation produces an unordered result.				
QC, [27]	Cumulative saturation bit. This bit is set to 1 to indicate that an <i>Advanced SIMD</i> integer operation has saturated after 0 was last written to this bit.				
AHP, [26]	Alternative Half-Precision control bit: <table><tr><td>0</td><td>IEEE half-precision format selected. This is the reset value.</td></tr><tr><td>1</td><td>Alternative half-precision format selected.</td></tr></table>	0	IEEE half-precision format selected. This is the reset value.	1	Alternative half-precision format selected.
0	IEEE half-precision format selected. This is the reset value.				
1	Alternative half-precision format selected.				
DN, [25]	Default <i>NaN</i> mode control bit:				

0	operands propagate through to the output of a operation. This is the reset value.
1	Any operation involving one or more s returns the Default .

The value of this bit only controls arithmetic. [AArch32](#) arithmetic always uses the Default setting, regardless of the value of the DN bit.

FZ, [24]

Flush-to-zero mode control bit:

0	Flush-to-zero mode disabled. Behavior of the system is fully compliant with the IEEE 754 standard. This is the reset value.
1	Flush-to-zero mode enabled.

The value of this bit only controls arithmetic. arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.

[RMode](#), [23:22]

Rounding Mode control field:

00	<i>Round to Nearest</i> (RN) mode. This is the reset value.
01	<i>Round towards Plus Infinity</i> (RP) mode.
10	<i>Round towards Minus Infinity</i> () mode.
11	<i>Round towards Zero</i> (RZ) mode.

The specified [rounding mode](#) is used by almost all instructions. arithmetic always uses the setting, regardless of the value of the ode bits.

Stride, [21:20]

<i>res0</i>	Reserved.
-------------	-----------

FZ16, [19]

Flush-to-zero mode control bit on half-precision data-processing instructions:

0	Flush-to-zero mode disabled. Behavior of the system is fully compliant with the IEEE 754 standard.
---	--

	1	Flush-to-zero mode enabled.
Len, [18:16]	<i>res0</i>	Reserved.
RES0, [15:8]	<i>res0</i>	Reserved.
IDC, [7]	Input Denormal cumulative <i>exception</i> bit. This bit is set to 1 to indicate that the Input Denormal has occurred since 0 was last written to this bit.	
RES0, [6:5]	<i>res0</i>	Reserved.
IXC, [4]	Inexact cumulative bit. This bit is set to 1 to indicate that the Inexact has occurred since 0 was last written to this bit.	
UFC, [3]	Underflow cumulative bit. This bit is set to 1 to indicate that the Underflow has occurred since 0 was last written to this bit.	
OFC, [2]	Overflow cumulative bit. This bit is set to 1 to indicate that the Overflow has occurred since 0 was last written to this bit.	
DZC, [1]	Division by Zero cumulative bit. This bit is set to 1 to indicate that the Division by Zero has occurred since 0 was last written to this bit.	
IOC, [0]	Invalid Operation cumulative bit. This bit is set to 1 to indicate that the Invalid Operation has occurred since 0 was last written to this bit.	

Configurations

There is one copy of this register that is used in both Secure and Non-secure states.

The named fields in this register map to the equivalent fields in the *AArch64* FPCR and FPSR. See [FPCR, Floating-point Control Register](#) and [FPSR, Floating-point Status Register](#).

Usage constraints

Accessing the FPSCR


To access the FPSCR:

```
VMRS <Rt>, FPSCR ; Read FPSCR into
  Rt
VMSR FPSCR, <Rt> ; Write Rt to FPSCR
```

Register access is encoded as follows:

Table 66: FPSCR access encoding

spec_reg
0001

 **Note:** The Cortex®-A76 *core* implementation does not support the deprecated *VFP* short vector feature. Attempts to execute the associated data-processing instructions result in an *UNDEFINED* Instruction .

Accessibility				This register is accessible as follows:		
EL0	EL0	EL1	EL1	EL2	EL3	EL3
(NS)	(S)	(NS)	(S)		(SCR.NS = 1)	(SCR.NS = 0)
Config	RW	-	-	-	-	-

Access to this register depends on the values of CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, CPTR_EL3.TFP, and HCR_EL2.{E2H, TGE}. For details of which values of these fields allow access at which *Exception levels*, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Part III

Debug descriptions

Topics:

- [Debug](#)
- [Performance Monitor Unit](#)
- [Activity Monitor Unit](#)
- [Embedded Trace Macrocell](#)

This part describes the debug functionality of the Cortex[®]-A76 core.

Chapter

16

Debug

Topics:

- [About debug methods](#)
- [Debug register interfaces](#)
- [Debug events](#)
- [External debug interface](#)

This chapter describes the Cortex[®]-A76 core debug registers and shows examples of how to use them.

About debug methods

The core is part of a debug system and supports both self-hosted and external debug.

The following figure shows a typical external debug system.

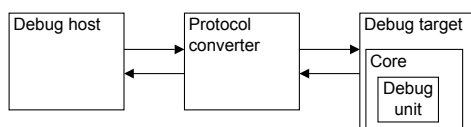


Figure 114: External debug system

Debug *host*

A computer, for example a personal computer, that is running a software debugger such as the [DS-5](#) Debugger. With the debug , you can issue high-level commands, such as setting a *breakpoint* at a certain location or examining the contents of a memory address.

Protocol converter

The debug sends messages to the debug target using an interface such as Ethernet. However, the debug target typically implements a different interface protocol. A *device* such as DSTREAM is required to convert between the two protocols.

Debug target

The lowest level of the system implements system support for the protocol converter to access the debug unit using the *Advanced Peripheral Bus* (APB) slave interface. An example of a debug target is a development system with a test chip or a silicon part with a *core*.

Debug unit

Helps debugging software that is running on the :

- Hardware systems that are based on the .
- Operating systems.
- Application software.

With the debug unit, you can:

- Stop program execution.
- Examine and alter process and *coprocessor* state.
- Examine and alter memory and the state of the input or output peripherals.
- Restart the .

For self-ed debug, the debug target runs additional debug monitor software that runs on the Cortex®-A76 itself. This way, it does not require expensive interface hardware to connect a second computer.

Debug register interfaces

The Debug architecture defines a set of debug registers.

The debug register interfaces provide access to these registers from:

- Software running on the *core*.
- An external debugger.

The Cortex®-A76 implements the Armv8 Debug architecture and debug events as described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*. It also implements improvements to Debug introduced in Armv8.1 and Armv8.2.

-
-
-
-

Core interfaces

System register access allows the core to directly access certain debug registers.

Debug registers

This function is system register based and memory-mapped. You can access the debug register map using the [APB](#) slave port. The external [debug interface](#) enables both external and self-hosted debug agents to access debug registers. Access to the debug registers is partitioned as follows:

Performance monitor

This function is system register based and memory-mapped. You can access the performance monitor registers using the slave port.

Activity monitor

This function is system register based and memory-mapped. You can access the activity monitor registers using the slave port.

Trace registers

This function is memory-mapped.

ELA registers

This function is memory-mapped.

Related reference

[External debug interface](#) on page 309

For information about external debug interface, including debug memory map and debug signals, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Breakpoints and watchpoints

The core supports six breakpoints, four watchpoints, and a standard *Debug Communications Channel* (DCC).

A [breakpoint](#) consists of a control register and a value register. These two registers are referred to as a *Breakpoint Register Pair* (BRP).

Four of the s (BRP 0-3) match only to [virtual address](#) and the other two (BRP 4 and 5) match against either or context ID, or VMID. All the watchpoints can be linked to two s (BRP 4 and 5) to enable a memory request to be trapped in a given process context.

Effects of resets on debug registers

The core has the following reset signals that affect the debug registers:

nCPUPORESET

This signal initializes the [core](#) logic, including the debug, [ETM](#) trace unit, [breakpoint](#), [watchpoint](#) logic, and performance monitors logic. This maps to a [Cold reset](#) that covers reset of the logic and the integrated debug functionality.

nCORERESET

This signal resets some of the debug and performance monitor logic. This maps to a [Warm reset](#) that covers reset of the logic.

External access permissions to debug registers

External access permission to the debug registers is subject to the conditions at the time of the access.

The following table describes the *core* response to accesses through the external *debug interface*.

Table 67: External access conditions to registers

Name	Condition	Description
Off	EDPRSR.PU is 0	Core power domain is completely off, or in a low-power state where the power domain registers cannot be accessed. If debug power is off, then all external debug and memory-mapped register accesses return an error.
DLK	DoubleLockStatus() == TRUE (EDPRSR.DLK is 1)	OS Double Lock is locked.
OSLK	OSLSR_EL1.OSLK is 1	OS Lock is locked.
EDAD	AllowExternalDebugAccess() == FALSE	External debug access is disabled. When an error is returned because of an EDAD condition code, and this is the highest priority error condition, EDPRSR.SDAD is set to 1. Otherwise SDAD is unchanged.
Default	-	None of the conditions apply, normal access.

The following table shows an example of external register access condition codes for access to a performance monitor register. To determine the access permission for the register, scan the columns from left to right. Stop at the first column a condition is true, the entry gives the access permission of the register and scanning stops.

Table 68: External register condition code example

Off	DLK	OSLK	EDAD	Default
-	-	-	-	RO

Debug events

A debug event can be a software debug event or a halting debug event.

A *core* responds to a debug event in one of the following ways:

- Ignores the debug event.
- Takes a debug *exception*.
- Enters debug state.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information about the debug events.

-
-

Related reference

[External debug interface](#) on page 309

For information about external debug interface, including debug memory map and debug signals, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

[About clocks, resets, and input synchronization](#) on page 40

The Cortex®-A76 core supports hierarchical clock gating.

Watchpoint debug events

In the Cortex®-A76 core, watchpoint debug events are always synchronous.

Memory [hint instructions](#) and cache [clean](#) operations, except DC ZVA and DC IVAC, do not generate [watchpoint](#) debug events. Store exclusive instructions generate a point debug event even when the check for the control of exclusive monitor fails. Atomic CAS instructions generate a point debug event even when the compare operation fails.

Debug OS Lock

Debug OS Lock is set by the powerup reset, nCPUPORESET.

For normal behavior of debug events and debug register accesses, Debug OS Lock must be cleared. For more information, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

External debug interface

For information about external debug interface, including debug memory map and debug signals, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Chapter

17

Performance Monitor Unit

Topics:

- [About the PMU](#)
- [PMU functional description](#)
- [PMU events](#)
- [PMU interrupts](#)
- [Exporting PMU events](#)

This chapter describes the *Performance Monitor Unit* (PMU) and the registers that it uses.

About the PMU

The Cortex®-A76 core includes performance monitors that enable you to gather various statistics on the operation of the core and its memory system during runtime. These provide useful information about the behavior of the core that you can use when debugging or profiling code.

The PMU provides six counters. Each counter can count any of the events available in the [core](#). The absolute counts recorded might vary because of pipeline effects. This has negligible effect except in cases where the counters are enabled for a very short time.

Related reference

[PMU events \(Ares/Enyo/Deimos Specific\)](#) on page 313

The following table shows the events that are generated and the numbers that the PMU uses to reference the events. The table also shows the bit position of each event on the event bus. Event reference numbers that are not listed are reserved.

PMU functional description

This section describes the functionality of the PMU.

The PMU includes the following interfaces and counters:

Event interface

Events from all other units from across the design are provided to the PMU.

System register and [APB](#) interface

You can program the PMU registers using the system registers or the external interface.

Counters

The PMU has 32-bit counters that increment when they are enabled, based on events, and a 64-bit cycle counter.

PMU register interfaces

The Cortex®-A76 [core](#) supports access to the performance monitor registers from the internal system register interface and a memory-mapped interface.

.

External register access permissions

Whether or not access is permitted to a register depends on:

- If the [core](#) is powered up.
- The state of the OS Lock and OS Double Lock.
- The state of External Performance Monitors access disable.
- The state of the debug authentication inputs to the .

The behavior is specific to each register and is not described in this document. For a detailed description of these features and their effects on the registers, see the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*.

The register descriptions provided in this manual describe whether each register is [read](#)/write or -only.

PMU events

The following table shows the events that are generated and the numbers that the PMU uses to reference the events. The table also shows the bit position of each event on the event bus. Event reference numbers that are not listed are reserved.

Table 69: PMU *Events*

number	PMU event bus (to trace)	mnemonic	description
0x0	[00]	SW_INCR	Software increment. Instruction architecturally executed (<i>condition code check</i> pass).
0x1	[01]	L1I_CACHE_REFILL	L1 instruction cache refill. This event counts any instruction fetch which misses in the cache. The following instructions are not counted: <ul style="list-style-type: none"> • Cache maintenance instructions. • Non-<i>cacheable</i> accesses.
0x2	[02]	L1I_ <i>TLB</i> _REFILL	L1 instruction refill. This event counts any refill of the instruction L1 from the L2 . This includes refills that result in a translation <i>fault</i> . The following instructions are not counted: <ul style="list-style-type: none"> • maintenance instructions. This event counts regardless of whether the <i>MMU</i> is enabled.

number	PMU event bus (to trace)	mnemonic	description
0x3	[167]	L1D_CACHE_REFILL	<p>L1 data cache refill. This event counts any load or store operation or <i>page table</i> walk access which causes data to be <i>read</i> from outside the L1, including accesses which do not allocate into L1.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Stores of an entire cache line, even if they make a coherency request outside the L1. • Partial cache line writes which do not allocate into the L1 cache. • Non- accesses. <p>This event counts the sum of L1D_CACHE_REFILL_RD and L1D_CACHE_REFILL_WR.</p>
0x4	[05:03]	L1D_CACHE	<p>L1 data cache access. This event counts any load or store operation or walk access which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL event causes this event to count.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Non- accesses. <p>This event counts the sum of L1D_CACHE_RD and L1D_CACHE_WR.</p>

number	PMU event bus (to trace)	mnemonic	description
0x5	[07:06]	L1D__REFILL	<p>L1 data refill. This event counts any refill of the data L1 from the L2 . This includes refills that result in a translation . The following instructions are not counted:</p> <ul style="list-style-type: none"> • maintenance instructions. <p>This event counts regardless of whether the is enabled.</p>
0x8	[11:08]	INST_RETIRE	<p>Instruction architecturally executed. This event counts all retired instructions, including those that fail their condition check.</p>
0x9	[12]	EXC_TAKEN	<p>Exception taken.</p>
0x0A	[13]	EXC_RETURN	<p>Instruction architecturally executed, pass, <i>exception</i> return.</p>
0x0B	[156]	CID_WRITE_RETIRE	<p>Instruction architecturally executed, pass, write to CONTEXTIDR. This event only counts writes to CONTEXTIDR in <i>AArch32</i> state, and via the CONTEXTIDR_EL1 mnemonic in <i>AArch64</i> state.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • <i>Writes</i> to CONTEXTIDR_EL12 and CONTEXTIDR_EL2.
0x10	[14]	BR_MIS_PRED	<p>Mispredicted or not predicted branch speculatively executed. This event counts any predictable branch instruction which is mispredicted either due to dynamic misprediction or because the is off and the branches are statically predicted not taken.</p>

number	PMU event bus (to trace)	mnemonic	description
0x11	[15]	CPU_CYCLES	Cycle
0x12	[16]	BR_PRED	Predictable branch speculatively executed. This event counts all predictable branches.
0x13	[19:17]	MEM_ACCESS	<p>Data memory access. This event counts memory accesses due to load or store instructions.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Instruction fetches. • Cache maintenance instructions. • Translation table walks or prefetches. <p>This event counts the sum of MEM_ACCESS_RD and MEM_ACCESS_WR.</p>
0x14	[20]	L1I_CACHE	<p>Level 1 instruction cache access or Level 0 Macro-op cache access. This event counts any instruction fetch which accesses the L1 instruction cache or L0 Macro-op cache.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions. • Non- accesses.

number	PMU event bus (to trace)	mnemonic	description
0x15	[21]	L1D_CACHE_WB	<p>L1 data cache -Back. This event counts any <i>write-back</i> of data from the L1 data cache to L2 or L3. This counts both <i>victim</i> line evictions and snoops, including cache maintenance operations.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> Invalidations which do not result in data being transferred out of the L1. Full-line writes which write to L2 without writing L1, such as write streaming mode.
0x16	[24:22]	L2D_CACHE	<p>L2 data cache access. This event counts any transaction from L1 which looks up in the L2 cache, and any from the L1 to the L2. Snoops from outside the <i>core</i> and cache maintenance operations are not counted.</p>
0x17	[27:25]	L2D_CACHE_REFILL	<p>L2 data cache refill. This event counts any transaction from L1 which causes data to be from outside the . L2 refills caused by stashes into L2 should not be counted.</p>
0x18	[30:28]	L2D_CACHE_WB	<p>L2 data cache . This event counts any of data from the L2 cache to outside the . This includes snoops to the L2 which return data, regardless of whether they cause an invalidation. Invalidations from the L2 which do not write data outside of the and snoops which return data from the L1 are not counted.</p>

number	PMU event bus (to trace)	mnemonic	description
0x19	[32:31]	BUS_ACCESS	Bus access. This event counts for every beat of data transferred over the data channels between the and the SCU. If both and write data beats are transferred on a given cycle, this event is counted twice on that cycle. This event counts the sum of BUS_ACCESS_RD and BUS_ACCESS_WR.
0x1A	[33]	MEMORY_ERROR	Local memory error. This event counts any correctable or uncorrectable memory error (ECC or parity) in the protected RAMs.
0x1B	[36:34]	INST_SPEC	Operation speculatively executed
0x1C	[37]	TTBR_WRITE_RETIRED	<p>Instruction architecturally executed, pass, write to TTBR. This event only counts writes to TTBR0/TTBR1 in state and TTBR0_EL1/TTBR1_EL1 in state.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Accesses to TTBR0_EL12/TTBR1_EL12 or TTBR0_EL2/TTBR1_EL2.
0x1D	[38]	BUS_CYCLES	Bus cycles. This event duplicates CPU_CYCLES.
0x1E	[39]	CHAIN	For odd-numbered counters, increments the count by one for each overflow of the preceding even-numbered counter. For even-numbered counters, there is no increment.

number	PMU event bus (to trace)	mnemonic	description
0x20	[41:40]	L2D_CACHE_ALLOCATE	L2 data cache allocation without refill. This event counts any full cache line write into the L2 cache which does not cause a linefill, including s from L1 to L2 and full-line writes which do not allocate into L1.
0x21	[42]	BR_RETIRE	Instruction architecturally executed, branch. This event counts all branches, taken or not. This excludes entries, debug entries and CCFAIL branches.
0x22	[43]	BR_MIS_PRED_RETIRE	Instruction architecturally executed, mispredicted branch. This event counts any branch counted by BR_RETIRE which is not correctly predicted and causes a pipeline flush.
0x23	[44]	STALL_FRONTEND	No operation issued because of the frontend. The counter counts on any cycle when there are no fetched instructions available to dispatch.
0x24	[45]	STALL_BACKEND	No operation issued because of the backend. The counter counts on any cycle fetched instructions are not dispatched due to resource constraints.
0x25	[48:46]	L1D_	Level 1 data access. This event counts any load or store operation which accesses the data L1 . If both a load and a store are executed on a cycle, this event counts twice. This event counts regardless of whether the is enabled.

number	PMU event bus (to trace)	mnemonic	description
0x26	[168]	L1I_	Level 1 instruction access. This event counts any instruction fetch which accesses the instruction L1. This event counts regardless of whether the is enabled.
0x29	[157]	L3D_CACHE_ALLOCATE	Attributable L3 data or unified cache allocation without refill. This event counts any full cache line write into the L3 cache which does not cause a linefill, including s from L2 to L3 and full-line writes which do not allocate into L2.
0x2A	[159:158]	L3D_CACHE_REFILL	Attributable Level 3 unified cache refill. This event counts for any transaction returning data from the SCU for which the data source was outside the <i>cluster</i> . Transactions such as ReadUnique are counted here as " transactions, even though they can be generated by store instructions.
0x2B	[160]	L3D_CACHE	Attributable Level 3 unified cache access. This event counts for any transaction returning data from the SCU, or for any write to the SCU.
0x2D	[49]	L2D__REFILL	Attributable L2 data or unified refill. This event counts on any refill of the L2 , caused by either an instruction or data access. This event does not count if the is disabled.

number	PMU event bus (to trace)	mnemonic	description
0x2F	[51:50]	L2D_	Attributable L2 data or unified access. This event counts on any access to the L2 (caused by a refill of any of the L1 s). This event does not count if the is disabled.
0x31	[161]	REMOTE_ACCESS	Access to another socket in a multi-socket system.
0x34	[52]	D_WALK	Access to data that caused a walk. This event counts on any data access which causes L2D__REFILL to count.
0x35	[53]	I_WALK	Access to instruction that caused a walk. This event counts on any instruction access which causes L2D__REFILL to count.
0x36	[163:162]	LL_CACHE_RD	<p>Last level cache access, .</p> <ul style="list-style-type: none"> • If CPUECTLR.EXTLLC is set: This event counts any transaction which returns a data source of 'interconnect cache'. • If CPUECTLR.EXTLLC is not set: This event is a duplicate of the L*D_CACHE_RD event corresponding to the last level of cache implemented – L3D_CACHE_RD if both per- L2 and L3 are implemented, L2D_CACHE_RD if only one is implemented, or L1D_CACHE_RD if neither is implemented.

number	PMU event bus (to trace)	mnemonic	description
0x37	[165:164]	LL_CACHE_MISS_RD	<p>Last level cache miss, .</p> <ul style="list-style-type: none"> • If CPUECTLR.EXTLLC is set: This event counts any transaction which returns a data source of 'DRAM', 'remote' or 'inter- peer'. • If CPUECTLR.EXTLLC is not set: This event is a duplicate of the L*D_CACHE_REFILL_RD event corresponding to the last level of cache implemented – L3D_CACHE_REFILL_RD if both per- L2 and L3 are implemented, L2D_CACHE_REFILL_RD if only one is implemented, or L1D_CACHE_REFILL_RD if neither is implemented.
0x40	[]	L1D_CACHE_RD	<p>L1 data cache access, .</p> <p>This event counts any load operation or walk access which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL_RD event causes this event to count.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Non- accesses.

number	PMU event bus (to trace)	mnemonic	description
0x41	[57:56]	L1D_CACHE_WR	<p>L1 data cache access, write. This event counts any store operation which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL_WR event causes this event to count.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Non- accesses.
0x42	[58]	L1D_CACHE_REFILL_RD	<p>L1 data cache refill, . This event counts any load operation or walk access which causes data to be from outside the L1, including accesses which do not allocate into L1.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Non- accesses.
0x43	[59]	L1D_CACHE_REFILL_WR	<p>L1 data cache refill, write. This event counts any store operation which causes data to be from outside the L1, including accesses which do not allocate into L1.</p> <p>The following instructions are not counted:</p> <ul style="list-style-type: none"> • Cache maintenance instructions and prefetches. • Stores of an entire cache line, even if they make a coherency request outside the L1. • Partial cache line writes which do not allocate into the L1 cache. • Non- accesses.

number	PMU event bus (to trace)	mnemonic	description
0x44	[60]	L1D_CACHE_REFILL_INNER	L1 data cache refill, inner. This event counts any L1 D-cache linefill (as counted by L1D_CACHE_REFILL) which hits in the L2 cache, L3 cache or another in the .
0x45	[61]	L1D_CACHE_REFILL_OUTER	L1 data cache refill, outer. This event counts any L1 D-cache linefill (as counted by L1D_CACHE_REFILL) which does not hit in the L2 cache, L3 cache or another in the , and instead obtains data from outside the .
0x46	[62]	L1D_CACHE_WB_VICTIM	L1 data cache ,
0x47	[63]	L1D_CACHE_WB_CLEAN	L1 data cache <i>cleaning</i> and coherency
0x48	[64]	L1D_CACHE_INVALID	L1 data cache <i>invalidate</i> .
0x4C	[65]	L1D__REFILL_RD	L1 data refill, .
0x4D	[66]	L1D__REFILL_WR	L1 data refill, write.
0x4E	[68:67]	L1D__RD	L1 data access, .
0x4F	[70:69]	L1D__WR	L1 data access, write.
0x50	[72:71]	L2D_CACHE_RD	L2 data cache access, . This event counts any transaction from L1 which looks up in the L2 cache. Snoops from outside the are not counted.
0x51	[74:73]	L2D_CACHE_WR	L2 data cache access, write. This event counts any write transaction from L1 which looks up in the L2 cache or any from L1 which allocates into the L2 cache. Snoops from outside the are not counted.

number	PMU event bus (to trace)	mnemonic	description
0x52	[76:75]	L2D_CACHE_REFILL_RD	L2 data cache refill, . This event counts any transaction from L1 which causes data to be from outside the . L2 refills caused by stashes into L2 should not be counted. Transactions such as ReadUnique are counted here as " transactions, even though they can be generated by store instructions.
0x53	[78:77]	L2D_CACHE_REFILL_WR	L2 data cache refill, write. This event counts any write transaction from L1 which causes data to be from outside the . L2 refills caused by stashes into L2 should not be counted. Transactions such as ReadUnique are not counted as write transactions.
0x56	[80:79]	L2D_CACHE_WB_VICTIM	L2 data cache , .
0x57	[82:81]	L2D_CACHE_WB_CLEAN	L2 data cache , ing and coherency.
0x58	[84:83]	L2D_CACHE_INVALID	L2 data cache .
0x5C	[85]	L2D__REFILL_RD	L2 data or unified refill, .
0x5D	[86]	L2D__REFILL_WR	L2 data or unified refill, write.
0x5E	[88:87]	L2D__RD	L2 data or unified access, .
0x5F	[89]	L2D__WR	L2 data or unified access, write.
0x60	[90]	BUS_ACCESS_RD	Bus access . This event counts for every beat of data transferred over the data channel between the and the SCU.
0x61	[91]	BUS_ACCESS_WR	Bus access write. This event counts for every beat of data transferred over the write data channel between the and the SCU.

number	PMU event bus (to trace)	mnemonic	description
0x66	[93:92]	MEM_ACCESS_RD	Data memory access, . This event counts memory accesses due to load instructions. The following instructions are not counted: <ul style="list-style-type: none"> • Instruction fetches. • Cache maintenance instructions. • Translation table walks. • Prefetches.
0x67	[95:94]	MEM_ACCESS_WR	Data memory access, write. This event counts memory accesses due to store instructions. The following instructions are not counted: <ul style="list-style-type: none"> • Instruction fetches. • Cache maintenance instructions. • Translation table walks. • Prefetches.
0x68	[97:96]	UNALIGNED_LD_SPEC	Unaligned access,
0x69	[99:98]	UNALIGNED_ST_SPEC	Unaligned access, write
0x6A	[102:100]	UNALIGNED_LDST_SPEC	Unaligned access
0x6C	[103]	LDREX_SPEC	Exclusive operation speculatively executed, LDREX or LDX.
0x6D	[104]	STREX_PASS_SPEC	Exclusive operation speculatively executed, STREX or STX pass.
0x6E	[105]	STREX_FAIL_SPEC	Exclusive operation speculatively executed, STREX or STX fail.
0x6F	[106]	STREX_SPEC	Exclusive operation speculatively executed, STREX or STX.
0x70	[109:107]	LD_SPEC	Operation speculatively executed, load.
0x71	[112:110]	ST_SPEC	Operation speculatively executed, store.

number	PMU event bus (to trace)	mnemonic	description
0x72	[114:113]	LDST_SPEC	Operation speculatively executed, load or store. This event counts the sum of LD_SPEC and ST_SPEC.
0x73	[117:115]	DP_SPEC	Operation speculatively executed, integer data-processing.
0x74	[120:118]	ASE_SPEC	Operation speculatively executed, <i>Advanced SIMD</i> instruction.
0x75	[123:121]	VFP_SPEC	Operation speculatively executed, <i>floating-point</i> instruction.
0x76	[125:124]	PC_WRITE_SPEC	Operation speculatively executed, software change of the PC.
0x77	[128:126]	CRYPTO_SPEC	Operation speculatively executed, Cryptographic instruction.
0x78	[129]	BR_IMMED_SPEC	Branch speculatively executed, immediate branch.
0x79	[130]	BR_RETURN_SPEC	Branch speculatively executed, procedure return.
0x7A	[131]	BR_INDIRECT_SPEC	Branch speculatively executed, indirect branch.
0x7C	[132]	<i>ISB_SPEC</i>	Barrier speculatively executed, .
0x7D	[134:133]	DSB_SPEC	Barrier speculatively executed, DSB.
0x7E	[136:135]	DMB_SPEC	Barrier speculatively executed, DMB.
0x81	[137]	EXC_UNDEF	Counts the number of undefined s taken locally.
0x82	[138]	EXC_SVC	Exception taken locally, <i>Supervisor Call</i> .
0x83	[139]	EXC_PABORT	Exception taken locally, <i>Instruction Abort</i> .
0x84	[140]	EXC_DABORT	Exception taken locally, <i>Data Abort</i> and SError.
0x86	[141]	EXC_ <i>IRQ</i>	Exception taken locally, .
0x87	[142]	EXC_ <i>FIQ</i>	Exception taken locally, .

number	PMU event bus (to trace)	mnemonic	description
0x88	[143]	EXC_SMC	Exception taken locally, Secure Monitor Call.
0x8A	[144]	EXC_HVC	Exception taken locally, Hypervisor Call.
0x8B	[145]	EXC_TRAP_PABORT	Exception taken, not taken locally.
0x8C	[146]	EXC_TRAP_DABORT	Exception taken, or SError not taken locally.
0x8D	[147]	EXC_TRAP_OTHER	Exception taken, Other traps not taken locally.
0x8E	[148]	EXC_TRAP_	Exception taken, not taken locally.
0x8F	[149]	EXC_TRAP_	Exception taken, not taken locally.
0x90	[152:150]	RC_LD_SPEC	Release consistency operation speculatively executed, load-acquire.
0x91	[155:153]	RC_ST_SPEC	Release consistency operation speculatively executed, store-release.
0xA0	[166]	L3_CACHE_RD	L3 cache .

PMU interrupts

The Cortex[®]-A76 core asserts the nPMUIRQ signal when the PMU generates an interrupt.

You can route this signal to an external interrupt controller for prioritization and masking. This is the only mechanism that signals this interrupt to the [core](#).

This interrupt is also driven as a trigger input to the [CTI](#). See the *Arm[®] DynamIQ[™] Shared Unit Technical Reference Manual* for more information.

Exporting PMU events

Some of the PMU events are exported to the ETM trace unit to be monitored.



Note: The PMUEVENT bus is not exported to external components. This is because the event bus cannot safely cross an asynchronous boundary when events can be generated on every cycle.

Chapter 18

Activity Monitor Unit

Topics:

- [About the AMU](#)
- [Accessing the activity monitors](#)
- [AMU counters](#)
- [AMU events](#)

This chapter describes the *Activity Monitor Unit* (AMU).

About the AMU

The Cortex®-A76 core includes activity monitoring. It has features in common with performance monitoring, but is intended for system management use whereas performance monitoring is aimed at user and debug applications.

The activity monitors provide useful information for system power management and persistent monitoring. The activity monitors are *read*-only in operation and their configuration is limited to the highest *Exception level* implemented.

The Cortex®-A76 *core* implements five counters, 0-4, and activity monitoring is only implemented in AArch64.

Accessing the activity monitors

The activity monitors can be accessed by:

- The system register interface for both *AArch64* and *AArch32* states.
- Read-only memory-mapped access using the debug *APB* interface.
-
-
-

Access enable bit

The access enable bit for traps on accesses to activity monitor registers is required at EL2 and EL3.

In the Cortex®-A76 *core*, the AMEN[4] bit in registers ACTLR_EL2 and ACTLR_EL3 controls the activity monitor registers enable.



Note: In the Cortex®-A76, the AMEN[4] bit is *RES0* in ACTLR and HACTLR. Activity monitors are not implemented in *AArch32*.

System register access

The core implements activity monitoring in AArch64 and the activity monitors can be accessed using the MRS and MSR instructions.

External memory-mapped access

Activity monitors can also be memory-mapped accessed from the APB debug interface.

In this case, the AMU registers just provide debug information and are read-only.

AMU counters

The Cortex®-A76 core implements five counters, 0-4. The activity monitor counters, AMEVCNTR0-4, have the following characteristics:

- All events are counted in 64-bit wrapping counters that overflow when they wrap. There is no support for overflow status indication or interrupts.
- Any change in clock frequency, including when a WFI and WFE instruction stops the clock, can affect any counter.
- Events 0, 1, 2, 3, and 4 are fixed, and the AMEVTYPER<n> evtCount bits are *read*-only.

AMU events

The following table describes the counters that are implemented in the Cortex[®]-A76 core and the mapping to fixed and programmable events.

Table 70: Mapping of counters to fixed events

Activity monitor counter <n>	<i>Event</i> type		number	Description
0	Fixed	Cycles at <i>core</i> frequency	11	Cycles count.
1	Fixed	Cycles at constant frequency	EF	This counter is used to replicate the generic system counter that is incremented on a constant basis, and not incremented depending on the <i>PE</i> frequency .
2	Fixed	Instructions retired	08	Instruction architecturally executed. This counter increments for every instruction that is executed architecturally, including instructions that fail their <i>condition code check</i> .
3	Fixed	First miss	F0	The first miss event tracks whether any external load miss is outstanding and starts counting only from a first-miss until data returns for that miss. The counter does not count for any remaining part of overlapping accesses, only counting again when the first-miss condition is re-detected.
4	Fixed	High activity	F1	Instructions executing through the design which act as a hint for potential high power activity.



Note: To program AMU counter 4, you need to program the AMEVTYR4_EL0 register. For more information, see [AMEVTYRn_EL0, Activity Monitor Type Register, EL0](#).

Chapter

19

Embedded Trace Macrocell

Topics:

- [About the ETM](#)
- [ETM trace unit generation options and resources](#)
- [ETM trace unit functional description](#)
- [Resetting the ETM](#)
- [Programming and reading ETM trace unit registers](#)
- [ETM trace unit register interfaces](#)

This chapter describes the ETM for the Cortex[®]-A76 core.

About the ETM

The ETM trace unit is a module that performs real-time instruction flow tracing based on the ETMv4 architecture. The ETM is a CoreSight component, and is an integral part of the Arm Real-time Debug solution, DS-5 Development Studio.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for more information.

ETM trace unit generation options and resources

The following table shows the trace generation options implemented in the Cortex®-A76 ETM trace unit.

Table 71: ETM trace unit generation options implemented

Description	Configuration
Instruction address size in bytes	8
Data address size in bytes	0
Data value size in bytes	0
Virtual Machine ID size in bytes	4
Context ID size in bytes	4
Support for conditional instruction tracing	Not implemented
Support for tracing of data	Not implemented
Support for tracing of load and store instructions as P0 elements	Not implemented
Support for cycle counting in the instruction trace	Implemented
Support for branch broadcast tracing	Implemented
Number of events supported in the trace	4
Return stack support	Implemented
Tracing of SError <i>exception</i> support	Implemented
Instruction trace cycle counting minimum threshold	1
Size of Trace ID	7 bits
Synchronization period support	Read-write
Global timestamp size	64 bits
Number of <i>cores</i> available for tracing	1
ATB trigger support	Implemented
Low power behavior override	Not implemented
Stall control support	Implemented
Support for overflow avoidance	Not implemented
Support for using CONTEXTIDR_EL2 in VMID comparator	Implemented

The following table shows the resources implemented in the Cortex[®]-A76 trace unit.

Table 72: trace unit resources implemented

Description	Configuration
Number of resource selection pairs implemented	8
Number of external input selectors implemented	4
Number of external inputs implemented	165, 4 <i>CTI</i> + 161 PMU
Number of counters implemented	2
Reduced function counter implemented	Not implemented
Number of sequencer states implemented	4
Number of Virtual Machine ID comparators implemented	1
Number of Context ID comparators implemented	1
Number of address comparator pairs implemented	4
Number of single-shot comparator controls	1
Number of comparator inputs implemented	0
Data address comparisons implemented	Not implemented
Number of data value comparators implemented	0

ETM trace unit functional description

This section describes the functionality of the ETM trace unit.

The following figure shows the main functional blocks of the *ETM* trace unit.

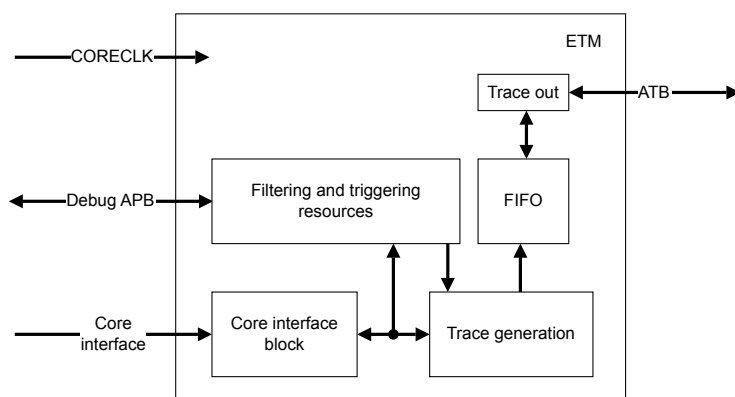


Figure 115: functional blocks

Core interface

This block monitors the behavior of the *core* and generates P0 elements that are essentially executed branches and *exceptions* traced in program order.

Trace generation

The trace generation block generates various trace packets based on P0 elements.

Filtering and triggering resources

You can limit the amount of trace data generated by the through the process of filtering.

For example, generating trace only in a certain address range. More complicated logic analyzer style filtering options are also available.

The trace unit can also generate a trigger that is a signal to the trace capture *device* to stop capturing trace.

FIFO

The trace generated by the trace unit is in a highly-compressed form.

The FIFO enables trace bursts to be flattened out. When the FIFO becomes full, the FIFO signals an overflow. The trace generation logic does not generate any new trace until the FIFO is emptied. This causes a gap in the trace when viewed in the debugger.

Trace out

Trace from FIFO is output on the AMBA *ATB* interface.

Resetting the ETM

The reset for the ETM trace unit is the same as a Cold reset for the core.

The *ETM* trace unit is not reset when *Warm reset* is applied to the *core* so that tracing through Warm reset is possible.

If the trace unit is reset, tracing stops until the trace unit is reprogrammed and re-enabled. However, if the is reset using , the last few instructions provided by the before the reset might not be traced.

Programming and reading ETM trace unit registers

You program and read the ETM trace unit registers using the Debug APB interface.

The *core* does not have to be in debug state when you program the *ETM* trace unit registers.

When you are programming the trace unit registers, you must enable all the changes at the same time. Otherwise, if you program the counter, it might start to count based on incorrect events before the correct setup is in place for the trigger condition.

To disable the trace unit, use the TRCPRGCTLR.EN bit.

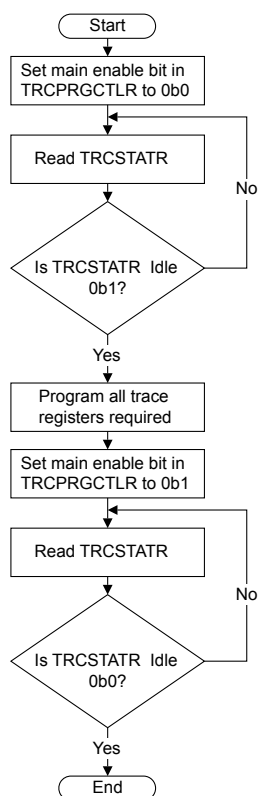


Figure 116: Programming trace unit registers

ETM trace unit register interfaces

The Cortex[®]-A76 core supports only memory-mapped interface to trace registers.

See the *Arm[®] Embedded Trace Macrocell Architecture Specification ETMv4* for information on the behaviors on register accesses for different trace unit states and the different access mechanisms.

Related reference

[External debug interface](#) on page 309

For information about external debug interface, including debug memory map and debug signals, see the *Arm[®] DynamIQ™ Shared Unit Technical Reference Manual*.

Part IV

Debug registers

Topics:

- [AArch32 debug registers](#)
- [AArch64 debug registers](#)
- [Memory-mapped debug registers](#)
- [AArch32 PMU registers](#)
- [AArch64 PMU registers](#)
- [Memory-mapped PMU registers](#)
- [PMU snapshot registers](#)
- [AArch64 AMU registers](#)
- [ETM registers](#)

This part describes the debug registers of the Cortex[®]-A76 core.

Chapter

20

AArch32 debug registers

Topics:

- [AArch32 debug register summary](#)

This chapter describes the debug registers in the AArch32 Execution state and shows examples of how to use them.

AArch32 debug register summary

The following table summarizes the 32-bit and 64-bit debug control registers that are accessible in the AArch32 Execution state from the internal CP14 interface. These registers are accessed by the MCR and MRC instructions in the order of CRn, op2, CRm, Op1 or MCRR and MRRC instructions in the order of CRm, Op1.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*.

Table 73: AArch32 debug register summary

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c0	0	c1	0	DBGDSCRint	RO	000x0000	Debug Status and Control Register, Internal View
c0	0	c5	0	DBGDTRTXin	WO	-	Debug Data Transfer Register, Transmit, Internal View
c0	0	c5	0	DBGDTRRXin	RO	00000000	Debug Data Transfer Register, Receive, Internal View

Chapter 21

AArch64 debug registers

Topics:

- [AArch64 debug register summary](#)
- [DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1](#)
- [DBGCLAIMSET_EL1, Debug Claim Tag Set Register, EL1](#)

This chapter describes the debug registers in the AArch64 Execution state and shows examples of how to use them.

AArch64 debug register summary

These registers, listed in the following table, are accessed by the MRS and MSR instructions in the order of Op0, CRn, Op1, CRm, Op2.

See [Memory-mapped debug register summary](#) for a complete list of registers accessible from the external . The 64-bit registers cover two addresses on the external memory interface. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table 74: AArch64 debug register summary

Name	Type	Reset	Width	Description
OSDTRRX_EL1	RW	00000000	32	Debug Data Transfer Register, Receive, External View
DBGBVR0_EL1	RW	-	64	Debug Breakpoint Value Register 0
DBGBCR0_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
DBGWVR0_EL1	RW	-	64	Debug Watchpoint Value Register 0
DBGWCR0_EL1	RW	UNK	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
DBGBVR1_EL1	RW	-	64	Debug Breakpoint Value Register 1
DBGBCR1_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
DBGWVR1_EL1	RW	-	64	Debug Watchpoint Value Register 1
DBGWCR1_EL1	RW	UNK	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
MDCCINT_EL1	RW	00000000	32	Monitor Debug Comms Channel Interrupt Enable Register
MDSCR_EL1	RW	-	32	
DBGBVR2_EL1	RW	-	64	Debug Breakpoint Value Register 2

Name	Type	Reset	Width	Description
DBGBCR2_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
DBGWVR2_EL1	RW	-	64	Debug Watchpoint Value Register 2
DBGWCR2_EL1	RW	UNK	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
OSDTRTX_EL1	RW	-	32	Debug Data Transfer Register, Transmit, External View
DBGBVR3_EL1	RW	-	64	Debug Breakpoint Value Register 3
DBGBCR3_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
DBGWVR3_EL1	RW	-	64	Debug Watchpoint Value Register 3
DBGWCR3_EL1	RW	UNK	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
DBGBVR4_EL1	RW	-	64	Debug Breakpoint Value Register 4
DBGBCR4_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
DBGBVR5_EL1	RW	-	64	Debug Breakpoint Value Register 5
DBGBCR5_EL1	RW	UNK	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
OSECCR_EL1	RW	00000000	32	Debug OS Lock Exception Catch Register
MDCCSR_EL0	RO	00000000	32	Monitor Debug Comms Channel Status Register
DBGDTR_EL0	RW	00000000	64	Debug Data Transfer Register, half-duplex

Name	Type	Reset	Width	Description
DBGDTRTX_EL0	WO	00000000	32	Debug Data Transfer Register, Transmit, Internal View
DBGDTRRX_EL0	RO	00000000	32	Debug Data Transfer Register, Receive, Internal View
MDRAR_EL1	RO	-	64	Debug ROM Address Register. This register is reserved, <i>RES0</i>
OSLAR_EL1	WO	-	32	Debug OS Lock Access Register
OSLSR_EL1	RO	0000000A	32	Debug OS Lock Status Register
OSDLR_EL1	RW	00000000	32	Debug OS Double Lock Register
DBGPRCR_EL1	RW	-	32	Debug Power/Reset Control Register
DBGCLAIMSET_EL1	RW	000000FF	32	DBGCLAIMSET_EL1 , Debug Claim Tag Set Register, EL1
DBGCLAIMCLR_EL1	RW	00000000	32	Debug Claim Tag Clear Register
DBGAUTHSTATUS_EL1	RO	000000AA	32	Debug Authentication Status Register

DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1

The DBGBCRn_EL1 holds control information for a breakpoint. Each DBGBCRn_EL1 is associated with a DBGBCRn_EL1 to form a *Breakpoint Register Pair* (BRP). DBGBCRn_EL1 is associated with DBGBCRn_EL1 to form BRPn. The range of *n* for DBGBCRn_EL1 is 0 to 5.

Bit field descriptions

The DBGBCRn_EL1 registers are 32-bit registers.

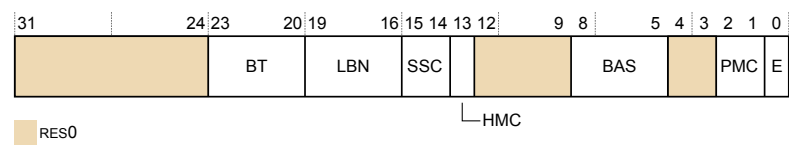


Figure 117: DBGBCRn_EL1 bit assignments

RES0, [31:24]	<i>res0</i>	Reserved.
BT, [23:20]	Breakpoint Type. This field controls the behavior of Breakpoint debug event generation. This includes	

the meaning of the value held in the associated $DBGBVR_n_EL1$, indicating whether it is an instruction address match or mismatch, or a Context match. It also controls whether the *breakpoint* is linked to another . The possible values are:

0b0000	Unlinked instruction address match.
0b0001	Linked instruction address match.
0b0010	Unlinked Context ID match.
0b0011	Linked Context ID match.
0b0100	Unlinked instruction address mismatch.
0b0101	Linked instruction address mismatch.
0b0110	Unlinked CONTEXTIDR_EL1 match.
0b0111	Linked CONTEXTIDR_EL1 match.
0b1000	Unlinked VMID match.
0b1001	Linked VMID match.
0b1010	Unlinked VMID + Context ID match.
0b1011	Linked VMID + Context ID match.
0b1100	Unlinked CONTEXTIDR_EL2 match.
0b1101	Linked CONTEXTIDR_EL2 match.
0b1110	Unlinked Full Context ID match.
0b1111	Linked Full Context ID match.

The field break down is:

- BT[3:1]: Base type. If the is not context-aware, these bits are *res0*. Otherwise, the possible values are:

0b000	Match address. $DBGBVR_n_EL1$ is the address of an instruction.
--------------	---

	0b001	Match context ID. DBGBVR n _EL1[31:0] is a context ID.
	0b010	Match VMID. DBGBVR n _EL1[47:32] is a VMID.
	0b011	Match VMID and CONTEXTIDR_EL1. DBGBVR n _EL1[31:0] is a context ID, and DBGBVR n _EL1[47:32] is a VMID.
	<ul style="list-style-type: none">BT[2]: Mismatch. <i>res0</i>.BT[0]: Enable linking.	
LBN, [19:16]	Linked number. For Linked address matching s, this specifies the index of the Context-matching linked to.	
SSC, [15:14]	<p>Security State Control. Determines the Security states under which a Breakpoint debug event for n is generated.</p> <p>This field must be interpreted with the <i>Higher Mode Control</i> (HMC), and <i>Privileged Mode Control</i> (PMC), fields to determine the mode and security states that can be tested.</p> <p>See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> for possible values of the HMC and PMC fields.</p>	
HMC, [13]	<p>Hyp Mode Control bit. Determines the debug perspective for deciding when a debug event for n is generated.</p> <p>This bit must be interpreted with the SSC and PMC fields to determine the mode and security states that can be tested.</p> <p>See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> for possible values of the SSC and PMC fields.</p>	
RES0, [12:9]	<i>res0</i>	Reserved.
BAS, [8:5]	Byte Address Select. Defines which half-words a regular matches, regardless of the instruction set and execution state. A debugger must program this field as follows:	
	3	Match the <i>T32</i> instruction at DBGBVR n _EL1.
	C	Match the instruction at DBGBVR $n+2$ _EL1.
	F	Match the <i>A64</i> or <i>A32</i> instruction at DBGBVR n _EL1, or context match.

All other values are reserved.

The Arm®v8-A architecture does not support direct execution of Java bytecodes. BAS[3] and BAS[1] ignore writes and on *reads* return the values of BAS[2] and BAS[0] respectively.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on how the BAS field is interpreted by hardware.

RES0, [4:3]

res0 Reserved.

PMC, [2:1]

Privileged Mode Control. Determines the *Exception level* or levels that a debug event for *n* is generated.

This field must be interpreted with the SSC and HMC fields to determine the mode and security states that can be tested.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for possible values of the SSC and HMC fields.

Bits[2:1] have no effect for accesses made in Hyp mode.

E, [0]

Enable . This bit enables the BRP:

0	BRP disabled.
1	BRP enabled.

A BRP never generates a debug event when it is disabled.

The value of DBGBCR*n*_EL1.E is *unknown* on reset. A debugger must ensure that DBGBCR*n*_EL1.E has a defined value before it enables debug.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

DBGCLAIMSET_EL1, Debug Claim Tag Set Register, EL1

The DBGCLAIMSET_EL1 is used by software to set CLAIM bits to 1.

Bit field descriptions

The DBGCLAIMSET_EL1 is a 32-bit register.

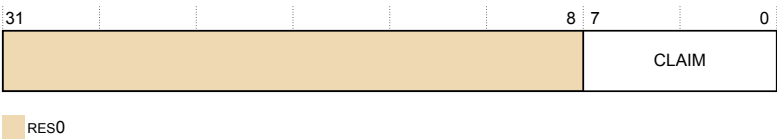


Figure 118: DBGCLAIMSET_EL1 bit assignments

RES0, [31:8]

res0

Reserved.

CLAIM, [7:0]

Claim set bits.

Writing a 1 to one of these bits sets the corresponding CLAIM bit to 1. This is an indirect write to the CLAIM bits.

A single write operation can set multiple bits to 1.

Writing 0 to one of these bits has no effect.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter

22

Memory-mapped debug registers

Topics:

- [Memory-mapped debug register summary](#)
- [EDCIDR0, External Debug Component Identification Register 0](#)
- [EDCIDR1, External Debug Component Identification Register 1](#)
- [EDCIDR2, External Debug Component Identification Register 2](#)
- [EDCIDR3, External Debug Component Identification Register 3](#)
- [EDDEVID, External Debug Device ID Register 0](#)
- [EDDEVID1, External Debug Device ID Register 1](#)
- [EDPIDR0, External Debug Peripheral Identification Register 0](#)
- [EDPIDR1, External Debug Peripheral Identification Register 1](#)
- [EDPIDR2, External Debug Peripheral Identification Register 2](#)
- [EDPIDR3, External Debug Peripheral Identification Register 3](#)
- [EDPIDR4, External Debug Peripheral Identification Register 4](#)
- [EDPIDRn, External Debug Peripheral Identification Registers 5-7](#)
- [EDRCR, External Debug Reserve Control Register](#)

This chapter describes the memory-mapped debug registers and shows examples of how to use them.

Memory-mapped debug register summary

The following table shows the offset address for the registers that are accessible from the external debug interface.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table 75: Memory-mapped debug register summary

Offset	Name	Type	Width	Description
0x000-01C	-	-	-	Reserved
020	EDESR	RW	32	External Debug Event Status Register
024	EDECR	RW	32	External Debug Execution Control Register
0x028-02C	-	-	-	Reserved
030	EDWAR[31:0]	RO	64	External Debug Watchpoint Address Register
034	EDWAR[63:32]			
0x038-07C	-	-	-	Reserved
080	DBGDTRRX_EL0	RW	32	Debug Data Transfer Register, Receive
084	EDITR	WO	32	External Debug Instruction Transfer Register
088	EDSCR	RW	32	External Debug Status and Control Register
08C	DBGDTRTX_EL0	WO	32	Debug Data Transfer Register, Transmit
090	EDRCR	WO	32	EDRCR, External Debug Reserve Control Register
094	-	RW	32	Reserved
098	EDECCR	RW	32	External Debug Exception Catch Control Register
09C	-	-	-	Reserved
0A0	-	-	-	Reserved
0A4	-	-	-	Reserved
0A8	-	-	-	Reserved
0AC	-	-	-	Reserved

Offset	Name	Type	Width	Description
0x0B0–2FC	-	-	-	Reserved
300	OSLAR_EL1	WO	32	OS Lock Access Register
0x304–30C	-	-	-	Reserved
310	EDPRCR	RW	32	External Debug Power/Reset Control Register
314	EDPRSR	RO	32	External Debug Processor Status Register
0x318–3FC	-	-	-	Reserved
400	DBGBVR0_EL1[31:0]	RW	64	Debug Breakpoint Value Register 0
404	DBGBVR0_EL1[63:32]			
408	DBGBCR0_EL1	RW	32	DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1
40C	-	-	-	Reserved
410	DBGBVR1_EL1[31:0]	RW	64	Debug Breakpoint Value Register 1
414	DBGBVR1_EL1[63:32]			
418	DBGBCR1_EL1	RW	32	DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1
41C	-	-	-	Reserved
420	DBGBVR2_EL1[31:0]	RW	64	Debug Breakpoint Value Register 2
424	DBGBVR2_EL1[63:32]			
428	DBGBCR2_EL1	RW	32	DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1
42C	-	-	-	Reserved
430	DBGBVR3_EL1[31:0]	RW	64	Debug Breakpoint Value Register 3
434	DBGBVR3_EL1[63:32]			
438	DBGBCR3_EL1	RW	32	DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1
43C	-	-	-	Reserved

Offset	Name	Type	Width	Description
440	DBGBVR4_EL1[31:0]	RW	64	Debug Breakpoint Value Register 4
444	DBGBVR4_EL1[63:32]			
448	DBGBCR4_EL1	RW	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
44C	-	-	-	Reserved
450	DBGBVR5_EL1[31:0]	RW	64	Debug Breakpoint Value Register 5
454	DBGBVR5_EL1[63:32]			
458	DBGBCR5_EL1	RW	32	DBGBCRn_EL1 , Debug Breakpoint Control Registers, EL1
0x45C-7FC	-	-	-	Reserved
800	DBGWVR0_EL1[31:0]	RW	64	Debug Watchpoint Value Register 0
804	DBGWVR0_EL1[63:32]			
808	DBGWCR0_EL1	RW	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
80C	-	-	-	Reserved
810	DBGWVR1_EL1[31:0]	RW	64	Debug Watchpoint Value Register 1
814	DBGWVR1_EL1[63:32]			
818	DBGWCR1_EL1	RW	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
81C	-	-	-	Reserved
820	DBGWVR2_EL1[31:0]	RW	64	Debug Watchpoint Value Register 2
824	DBGWVR2_EL1[63:32]			
828	DBGWCR2_EL1	RW	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1
82C	-	-	-	Reserved
830	DBGWVR3_EL1[31:0]	RW	64	Debug Watchpoint Value Register 0,
834	DBGWVR3_EL1[63:32]			
838	DBGWCR3_EL1	RW	32	DBGWCRn_EL1 , Debug Watchpoint Control Registers, EL1

Offset	Name	Type	Width	Description
0x83C–CFC	-	-	-	Reserved
D00	MIDR	RO	32	MIDR_EL1 , Main ID Register, EL1
0xD04–D1C	-	-	-	Reserved
D20	EDPFR[31:0]	RO	64	ID_AA64PFR0_EL1 , <i>AArch64</i> Processor Feature Register 0, EL1
D24	EDPFR[63:32]			
D28	EDDFR[31:0]	RO	64	ID_AA64PFR0_EL1 , Processor Feature Register 0, EL1
D2C	EDDFR[63:32]			
0xD60–EFC	-	-	-	Reserved
F00	-	-	-	Reserved
0xF04–F9C	-	-	-	Reserved
FA0	DBGCLAIMSET_EL1	RW	32	DBGCLAIMSET_EL1 , Debug Claim Tag Set Register, EL1
FA4	DBGCLAIMCLR_EL1	RW	32	Debug Claim Tag Clear Register
FA8	EDDEVAFF0	RO	32	External Debug Device Affinity Register 0
FAC	EDDEVAFF1	RO	32	External Debug Device Affinity Register 1
FB0	-	-	-	Reserved
FB4	-	-	-	Reserved
FB8	DBGAUTHSTATUS_EL0	RO	32	Debug Authentication Status Register
FBC	EDDEVARCH	RO	32	External Debug Device Architecture Register
FC0	EDDEVID2	RO	32	External Debug Device ID Register 2, <i>res0</i>
FC4	EDDEVID1	RO	32	EDDEVID1 , External Debug Device ID Register 1
FC8	EDDEVID	RO	32	EDDEVID , External Debug Device ID Register 0

Offset	Name	Type	Width	Description
FCC	EDDEVTYPE	RO	32	External Debug Device Type Register
FD0	EDPIDR4	RO	32	EDPIDR4, External Debug Peripheral Identification Register 4
0xFD4–FDC	EDPIDR5-7	RO	32	EDPIDRn, External Debug Peripheral Identification Registers 5-7
FE0	EDPIDR0	RO	32	EDPIDR0, External Debug Peripheral Identification Register 0
FE4	EDPIDR1	RO	32	EDPIDR1, External Debug Peripheral Identification Register 1
FE8	EDPIDR2	RO	32	EDPIDR2, External Debug Peripheral Identification Register 2
FEC	EDPIDR3	RO	32	EDPIDR3, External Debug Peripheral Identification Register 3
FF0	EDCIDR0	RO	32	EDCIDR0, External Debug Component Identification Register 0
FF4	EDCIDR1	RO	32	EDCIDR1, External Debug Component Identification Register 1
FF8	EDCIDR2	RO	32	EDCIDR2, External Debug Component Identification Register 2
FFC	EDCIDR3	RO	32	EDCIDR3, External Debug Component Identification Register 3

EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 provides information to identify an external debug component.

Bit field descriptions

The EDCIDR0 is a 32-bit register.

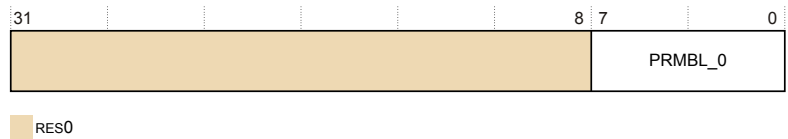


Figure 119: EDCIDR0 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_0, [7:0]	0D	Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR0 can be accessed through the external [debug interface](#), offset FF0.

EDCIDR1, External Debug Component Identification Register 1

The EDCIDR1 provides information to identify an external debug component.

Bit field descriptions

The EDCIDR1 is a 32-bit register.

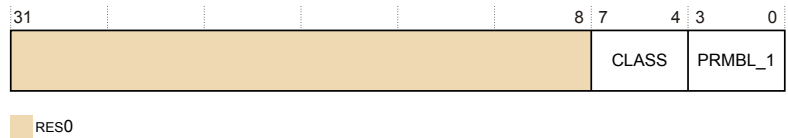


Figure 120: EDCIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
CLASS, [7:4]	9	Debug component.
PRMBL_1, [3:0]	0	Preamble.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR1 can be accessed through the external [debug interface](#), offset FF4.

EDCIDR2, External Debug Component Identification Register 2

The EDCIDR2 provides information to identify an external debug component.

Bit field descriptions

The EDCIDR2 is a 32-bit register.

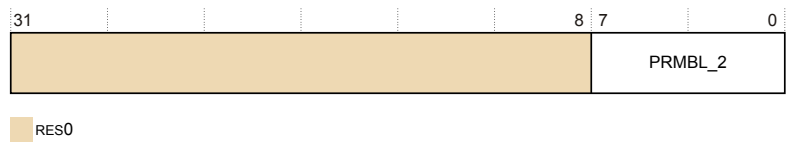


Figure 121: EDCIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_2, [7:0]	05	Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR2 can be accessed through the external [debug interface](#), offset FF8.

EDCIDR3, External Debug Component Identification Register 3

The EDCIDR3 provides information to identify an external debug component.

Bit field descriptions

The EDCIDR3 is a 32-bit register.

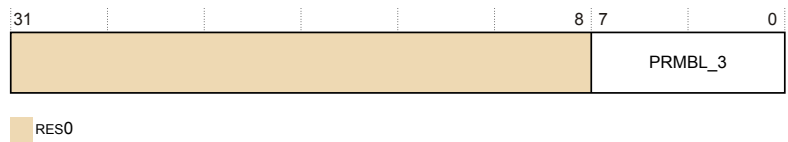


Figure 122: EDCIDR3 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_3, [7:0]	B1	Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR3 can be accessed through the external [debug interface](#), offset FFC.

EDDEVID, External Debug Device ID Register 0

The EDDEVID provides extra information for external debuggers about features of the debug implementation.

Bit field descriptions

The EDDEVID is a 32-bit register.

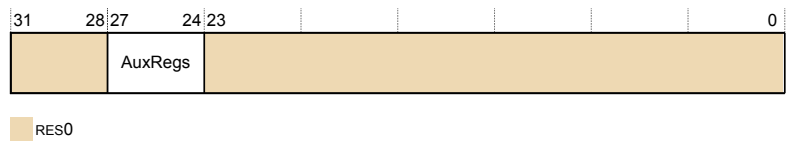


Figure 123: EDDEVID bit assignments

RES0, [31:28]	<i>res0</i>	Reserved.
AuxRegs, [27:24]		Indicates support for Auxiliary registers:
	0	None supported.
RES0, [23:0]	<i>res0</i>	Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDDEVID can be accessed through the external [debug interface](#), offset FC8.

EDDEVID1, External Debug Device ID Register 1

The EDDEVID1 provides extra information for external debuggers about features of the debug implementation.

Bit field descriptions

The EDDEVID1 is a 32-bit register.

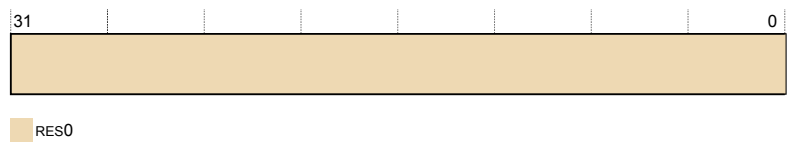


Figure 124: EDDEVID1 bit assignments

RES0, [31:0]	<i>res0</i>	Reserved.
--------------	-------------	-----------

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDDEVID1 can be accessed through the external [debug interface](#), offset FC4.

EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 provides information to identify an external debug component.

Bit field descriptions

The EDPIDR0 is a 32-bit register.

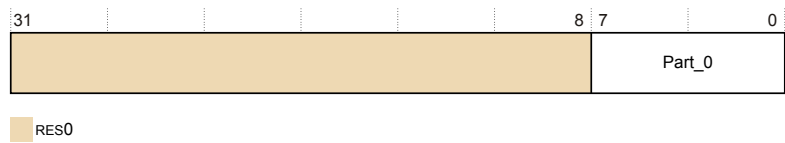


Figure 125: EDPIDR0 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Part_0, [7:0]	0B	Least significant byte of the debug part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR0 can be accessed through the external [debug interface](#), offset FE0.

EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 provides information to identify an external debug component.

Bit field descriptions

The EDPIDR1 is a 32-bit register.

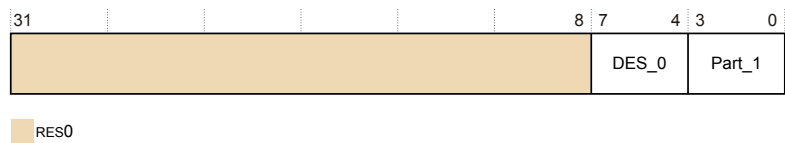


Figure 126: EDPIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
DES_0, [7:4]	B	Arm Limited. This is the least significant nibble of JEP106 ID code.
Part_1, [3:0]	D	Most significant nibble of the debug part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR1 can be accessed through the external [debug interface](#), offset FE4.

EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 provides information to identify an external debug component.

Bit field descriptions

The EDPIDR2 is a 32-bit register.

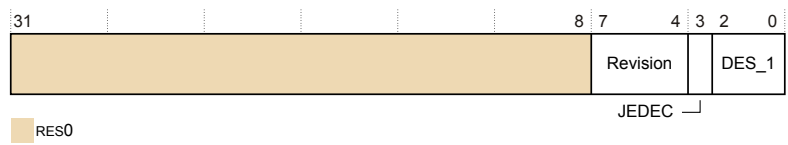


Figure 127: EDPIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Revision, [7:4]	3	r0p1.
JEDEC, [3]	1	RAO. Indicates a JEP106 identity code is used.
DES_1, [2:0]	011	Arm Limited. This is the most significant nibble of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR2 can be accessed through the external [debug interface](#), offset FE8.

EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 provides information to identify an external debug component.

Bit field descriptions

The EDPIDR3 is a 32-bit register.

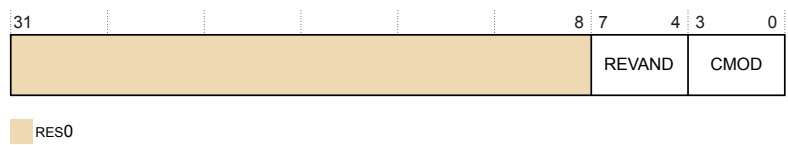


Figure 128: EDPIDR3 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
--------------	-------------	-----------

REVAND, [7:4]	0	Part minor revision.
CMOD, [3:0]	0	Customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR3 can be accessed through the external [debug interface](#), offset FEC.

EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 provides information to identify an external debug component.

Bit field descriptions

The EDPIDR4 is a 32-bit register.

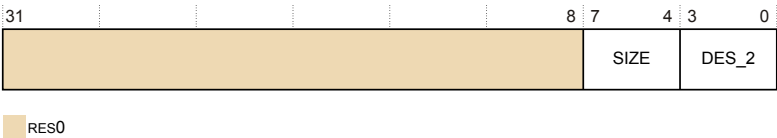


Figure 129: EDPIDR4 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
SIZE, [7:4]	0	Size of the component. Log ₂ the number of 4KB pages from the start of the component to the end of the component ID registers.
DES_2, [3:0]	4	Arm Limited This is the least significant nibble JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR4 can be accessed through the external [debug interface](#), offset FD0.

EDPIDRn, External Debug Peripheral Identification Registers 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are *res0*.

EDRCR, External Debug Reserve Control Register

The EDRCR is part of the Debug registers functional group.

Bit field descriptions

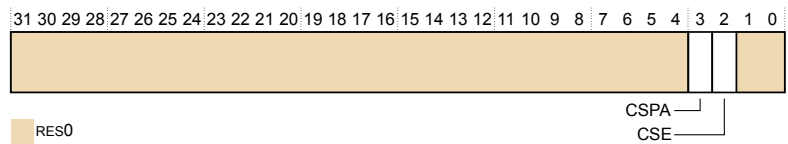


Figure 130: EDRCR bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
CSPA, [3]	Clear Sticky Pipeline Advance. This bit is used to clear the EDSCR.PipeAdv bit to 0. The actions on writing to this bit are:	
	0	No action.
	1	Clear the EDSCR.PipeAdv bit to 0.
CSE, [2]	Clear Sticky Error. Used to clear the EDSCR cumulative error bits to 0. The actions on writing to this bit are:	
	0	No action
	1	Clear the EDSCR.{TXU, RXO, ERR} bits, and, if the <i>core</i> is in Debug state, the EDSCR.ITO bit, to 0.
RES0, [1:0]	<i>res0</i>	Reserved.

The EDRCR can be accessed through the internal memory-mapped interface and the external [debug interface](#), offset 090.

Usage constraints

This register is accessible as follows:

Off	DLK	OSLK	SLK	Default
Error	Error	Error	WI	WO

Configurations

EDRCR is in the Core power domain.

Chapter

23

AArch32 PMU registers

Topics:

- [AArch32 PMU register summary](#)
- [PMCEID0, Performance Monitors Common Event Identification Register 0](#)
- [PMCEID1, Performance Monitors Common Event Identification Register 1](#)
- [PMCR, Performance Monitors Control Register](#)

This chapter describes the AArch32 PMU registers and shows examples of how to use them.

AArch32 PMU register summary

The PMU counters and their associated control registers are accessible in the AArch32 Execution state from the internal CP15 system register interface with MCR and MRC instructions for 32-bit registers and MCRR and MRRC for 64-bit registers.

The following table gives a summary of the Cortex[®]-A76 PMU registers in the [AArch32](#) Execution state. For those registers not described in this chapter, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table 76: PMU register summary in the Execution state

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c9	0	c12	0	PMCR	RW	32	0x410B30XX	PMCR, Performance Monitors Control Register
c9	0	c12	1	PMCNTENSEL	RW	32	00000000	Performance Monitors Count Enable Set Register
c9	0	c12	2	PMCNTENCCLR	RW	32	00000000	Performance Monitors Count Enable Clear Register
c9	0	c12	3	PMOVS	RW	32	00000000	Performance Monitors Overflow Flag Status Register
c9	0	c12	4	PMSWINC	WO	32	UNK	Performance Monitors Software Increment Register
c9	0	c12	5	PMSEL	RW	32	UNK	Performance Monitors <i>Event</i> Counter Selection Register

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c9	0	c12	6	PMCEID0	RO	32	7FFF0F3F	PMCEID0_EL0, Performance Monitors Common Identification Register 0, EL0 (Ares/Enyo specific)
c9	0	c12	7	PMCEID1	RO	32	FEF2AE7F	PMCEID1_EL0, Performance Monitors Common Identification Register 1, EL0 (Ares/Enyo specific)
c9	0	c14	4	PMCEID2	RO	32	00000000	Reserved
c9	0	c14	5	PMCEID3	RO	32	00000000	Reserved
c9	0	c13	0	PMCCNTR[31:0]	RW	32	UNK	Performance Monitors Cycle Count Register
c9	3	c13	0	PMCCNTR[63:0]	RW	64	UNK	
c9	0	c13	1	PMXEVTYP[31:0]	RW	32	UNK	Performance Monitors Selected Type Register
c9	0	c13	2	PMXEVCNT[31:0]	RW	32	UNK	Performance Monitors Selected Count Register
c9	0	c14	0	PMUSEREN[31:0]	RW	32	UNK	Performance Monitors User Enable Register
c9	0	c14	3	PMOVSSET[31:0]	RW	32	00000000	Performance Monitor Overflow Flag Status Set Register

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c14	0	c8	0	PMEVCNTR0	RW	32	UNK	Performance Monitor Count Registers
c14	0	c8	1	PMEVCNTR1	RW	32	UNK	
c14	0	c8	2	PMEVCNTR2	RW	32	UNK	
c14	0	c8	3	PMEVCNTR3	RW	32	UNK	
c14	0	c8	4	PMEVCNTR4	RW	32	UNK	
c14	0	c8	5	PMEVCNTR5	RW	32	UNK	
c14	0	c12	0	PMEVTYPER0	RW	32	UNK	Performance Monitors Type Registers
c14	0	c12	1	PMEVTYPER1	RW	32	UNK	
c14	0	c12	2	PMEVTYPER2	RW	32	UNK	
c14	0	c12	3	PMEVTYPER3	RW	32	UNK	
c14	0	c12	4	PMEVTYPER4	RW	32	UNK	
c14	0	c12	5	PMEVTYPER5	RW	32	UNK	
c14	0	c15	7	PMCCFILTR	RW	32	UNK	Performance Monitors Cycle Count Filter Register

PMCEID0, Performance Monitors Common Event Identification Register 0

The PMCEID0 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

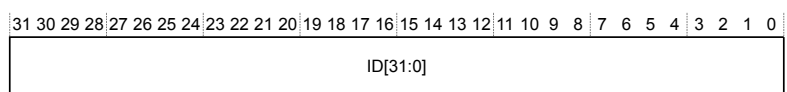


Figure 131: PMCEID0 bit assignments

ID[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

The following table shows the PMCEID0 bit assignments with event implemented or not implemented when the associated bit is set to 1 or 0. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information about these events.


Table 77: PMU events

Bit	Event mnemonic	Description
[31]	L1D_CACHE_ALLOCATE	L1 Data cache allocate: 0 This event is not implemented.
[30]	CHAIN	Chain. For odd-numbered counters, counts once for each overflow of the preceding even-numbered counter. For even-numbered counters, does not count: 1 This event is implemented.
[29]	BUS_CYCLES	Bus cycle: 1 This event is implemented.
[28]	TTBR_WRITE_RETIRED	TTBR write, architecturally executed, condition check pass - write to <i>translation table</i> base: 1 This event is implemented.
[27]	INST_SPEC	Instruction speculatively executed: 1 This event is implemented.
[26]	MEMORY_ERROR	Local memory error: 1 This event is implemented.
[25]	BUS_ACCESS	Bus access: 1 This event is implemented.
[24]	L2D_CACHE_WB	L2 Data cache <i>Write</i> -Back: 1 This event is implemented.
[23]	L2D_CACHE_REFILL	L2 Data cache refill: 1 This event is implemented.

Bit	Event mnemonic	Description
[22]	L2D_CACHE	L2 Data cache access: 1 This event is implemented.
[21]	L1D_CACHE_WB	L1 Data cache -Back: 1 This event is implemented.
[20]	L1I_CACHE	L1 Instruction cache access: 1 This event is implemented.
[19]	MEM_ACCESS	Data memory access: 1 This event is implemented.
[18]	BR_PRED	Predictable branch speculatively executed: 1 This event is implemented.
[17]	CPU_CYCLES	Cycle: 1 This event is implemented.
[16]	BR_MIS_PRED	Mispredicted or not predicted branch speculatively executed: 1 This event is implemented.
[15]	UNALIGNED_LDST_RETIRE	Instruction architecturally executed, condition check pass - unaligned load or store: 0 This event is not implemented.
[14]	BR_RETURN_RETIRE	Instruction architecturally executed, condition check pass - procedure return: 0 This event is not implemented.

Bit	Event mnemonic	Description
[13]	BR_IMMED_RETIRED	Instruction architecturally executed - immediate branch: 0 This event is not implemented.
[12]	PC_WRITE_RETIRED	Instruction architecturally executed, condition check pass - software change of the PC: 0 This event is not implemented.
[11]	CID_WRITE_RETIRED	Instruction architecturally executed, condition check pass - write to CONTEXTIDR: 1 This event is implemented.
[10]	EXC_RETURN	Instruction architecturally executed, condition check pass - <i>exception</i> return: 1 This event is implemented.
[9]	EXC_TAKEN	Exception taken: 1 This event is implemented.
[8]	INST_RETIRED	Instruction architecturally executed: 1 This event is implemented.
[7]	ST_RETIRED	Instruction architecturally executed, condition check pass - store: 0 This event is not implemented.
[6]	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 0 This event is not implemented.
[5]	L1D_ <i>TLB</i> _REFILL	L1 Data refill: 1 This event is implemented.

Bit	Event mnemonic	Description
[4]	L1D_CACHE	L1 Data cache access: 1 This event is implemented.
[3]	L1D_CACHE_REFILL	L1 Data cache refill: 1 This event is implemented.
[2]	L1I_REFILL	L1 Instruction refill: 1 This event is implemented.
[1]	L1I_CACHE_REFILL	L1 Instruction cache refill: 1 This event is implemented.
[0]	SW_INCR	Instruction architecturally executed, condition check pass - software increment: 1 This event is implemented.

 **Note:** The PMU events implemented in the above table can be found in [Table 1](#).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

PMCEID1, Performance Monitors Common Event Identification Register 1

The PMCEID1 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

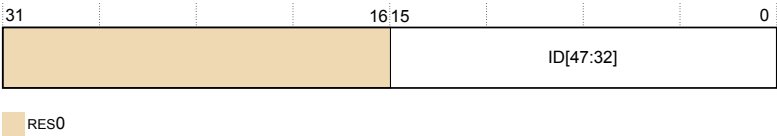



Figure 132: PMCEID1 bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
ID[47:32], [15:0]		Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table 78: PMU common events

Bit	Event mnemonic	Description
[15]	L2D_ <i>TLB</i>	
[13]	L2D__REFILL	
[6]	L1I_	
[5]	L1D_	
[4]	STALL_BACKEND	
[3]	STALL_FRONTEND	
[2]	BR_MIS_PRED_RETIRE	
[1]	BR_RETIRE	
[0]	L2D_CACHE_ALLOCATE	

 **Note:** The PMU events implemented in the above table can be found in [Table 1](#).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

PMCR, Performance Monitors Control Register

The PMCR provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Bit field descriptions

PMCR is a 32-bit register, and is part of the Performance Monitors registers functional group.

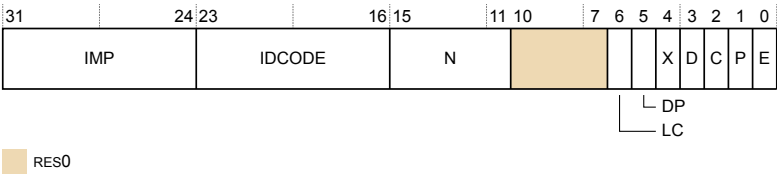


Figure 133: PMCR bit assignments

IMP, [31:24]	Indicates the implementer code. The value is:
	41 ASCII character 'A' - implementer is Arm® Limited.
IDCODE, [23:16]	Identification code. The value is:
	0B Cortex®-A76 <i>core</i> .

N, [15:11]

Identifies the number of event counters implemented.

00110

The implements six event counters.

RES0, [10:7]

res0

Reserved.

LC, [6]

Long cycle count enable. Determines which PMCCNTR bit generates an overflow recorded in PMOVSR[31]. The overflow event is generated on a 32-bit or 64-bit boundary. The possible values are:

0

Overflow event is generated on a 32-bit boundary, when an increment changes PMCCNTR[31] from 1 to 0. This is the reset value.

1

Overflow event is generated on a 64-bit boundary, when an increment changes PMCCNTR[63] from 1 to 0.

Arm deprecates use of PMCR.LC = 0.

DP, [5]

Disable cycle counter CCNT when event counting is prohibited. The possible values are:

0

Cycle counter operates regardless of the non-invasive debug authentication settings. This is the reset value.

1

Cycle counter is disabled if non-invasive debug is not permitted and enabled.

X, [4]

Export enable. This bit permits events to be exported to another debug *device*, such as a trace macrocell, over an event bus. The possible values are:

0

Export of events is disabled. This is the reset value.

1

Export of events is enabled.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the *PE*.

D, [3]

When this register has an architecturally defined reset value, if this field is implemented as an RW field, it resets to 0.

Clock divider. The possible values are:

- | | |
|----------|---|
| 0 | When enabled, counter CCNT counts every clock cycle. This is the reset value. |
| 1 | When enabled, counter CCNT counts once every 64 clock cycles. |

Arm deprecates use of PMCR.D = 1.

C, [2]

Cycle counter reset. This bit is WO. The effects of writing to this bit are:

- | | |
|----------|-------------------------------------|
| 0 | No action. This is the reset value. |
| 1 | Reset PMCCNTR to zero. |

This bit is always *RAZ*.

Resetting PMCCNTR does not clear the PMCCNTR overflow bit to 0. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

P, [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

- | | |
|----------|--|
| 0 | No action. This is the reset value. |
| 1 | Reset all event counters accessible in the current EL, not including PMCCNTR, to zero. |

This bit is always .

In Non-secure EL0 and EL1, a write of 1 to this bit does not reset event counters that HDCR.HPMN or MDCR_EL2.HPMN reserves for EL2 use.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Resetting the event counters does not clear any overflow bits to 0.

E, [0]

Enable. The possible values are:

- | | |
|----------|---|
| 0 | All counters that are accessible at Non-secure EL1, including |
|----------|---|

PMCCNTR, are disabled.
This is the reset value.

1

When this register has an architecturally defined reset value, this field resets to 0.

This bit is RW.

This bit does not affect the operation of event counters that HDCR.HPMN or MDCR_EL2.HPMN reserves for EL2 use.

When this register has an architecturally defined reset value, this field resets to 0.

Configurations

[AArch32](#) System register PMCR is architecturally mapped to [AArch64](#) System register PMCR_EL0. See [PMCR_EL0, Performance Monitors Control Register, EL0](#).

System register PMCR bits [6:0] are architecturally mapped to External register PMCR_EL0[6:0].

There is one instance of this register that is used in both Secure and Non-secure states.

This register is in the *Warm reset* domain. Some or all RW fields of this register have defined reset values. On a Warm or *Cold reset* these apply only if the resets into an *Exception level* that is using . Otherwise, on a Warm or RW fields in this register reset to architecturally *UNKNOWN* values.

Chapter

24

AArch64 PMU registers

Topics:

- [AArch64 PMU register summary](#)
- [PMCEID0_EL0, Performance Monitors Common Event Identification Register 0, EL0](#)
- [PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0](#)
- [PMCR_EL0, Performance Monitors Control Register, EL0](#)

This chapter describes the AArch64 PMU registers and shows examples of how to use them.

AArch64 PMU register summary

The PMU counters and their associated control registers are accessible in the AArch64 Execution state with MRS and MSR instructions.

The following table gives a summary of the Cortex[®]-A76 PMU registers in the *AArch64* Execution state. For those registers not described in this chapter, see the *Arm[®] Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table 79: PMU register summary in the Execution state

Name	Type	Width	Reset	Description
PMCR_EL0	RW	32	0x410B30XX	PMCR_EL0 , Performance Monitors Control Register, EL0
PMCNTENSET_EL0	RW	32	UNK	Performance Monitors Count Enable Set Register
PMCNTENCLR_EL0	RW	32	UNK	Performance Monitors Count Enable Clear Register
PMOVSLR_EL0	RW	32	UNK	Performance Monitors Overflow Flag Status Register
PMSWINC_EL0	WO	32	UNK	Performance Monitors Software Increment Register
PMSELR_EL0	RW	32	UNK	Performance Monitors <i>Event</i> Counter Selection Register
PMCEID0_EL0	RO	64	F7FF0F3F	PMCEID0_EL0 , Performance Monitors Common Identification Register 0, EL0 (Ares/Enyo specific)
PMCEID1_EL0	RO	64	0000BE7F	PMCEID1_EL0 , Performance Monitors Common Identification Register 1, EL0 (Ares/Enyo Specific)
PMCCNTR_EL0	RW	64	UNK	Performance Monitors Cycle Count Register

Name	Type	Width	Reset	Description
PMXEVTYPER_EL0	RW	32	UNK	Performance Monitors Selected Type and Filter Register
PMCCFILTR_EL0	RW	32	UNK	Performance Monitors Cycle Count Filter Register
PMXEVCNTR_EL0	RW	32	UNK	Performance Monitors Selected Count Register
PMUSERENR_EL0	RW	32	UNK	Performance Monitors User Enable Register
PMINTENSET_EL1	RW	32	UNK	Performance Monitors Interrupt Enable Set Register
PMINTENCLR_EL1	RW	32	UNK	Performance Monitors Interrupt Enable Clear Register
PMOVSSET_EL0	RW	32	UNK	Performance Monitors Overflow Flag Status Set Register
PMEVCNTR0_EL0	RW	32	UNK	Performance Monitors Count Registers
PMEVCNTR1_EL0	RW	32	UNK	
PMEVCNTR2_EL0	RW	32	UNK	
PMEVCNTR3_EL0	RW	32	UNK	
PMEVCNTR4_EL0	RW	32	UNK	
PMEVCNTR5_EL0	RW	32	UNK	
PMEVTYPER0_EL0	RW	32	UNK	Performance Monitors Type Registers
PMEVTYPER1_EL0	RW	32	UNK	
PMEVTYPER2_EL0	RW	32	UNK	
PMEVTYPER3_EL0	RW	32	UNK	
PMEVTYPER4_EL0	RW	32	UNK	
PMEVTYPER5_EL0	RW	32	UNK	
PMCCFILTR_EL0	RW	32	UNK	Performance Monitors Cycle Count Filter Register

Related reference

[PMU events \(Ares/Enyo/Deimos Specific\)](#) on page 313

The following table shows the events that are generated and the numbers that the PMU uses to reference the events. The table also shows the bit position of each event on the event bus. Event reference numbers that are not listed are reserved.

PMCEID0_EL0, Performance Monitors Common Event Identification Register 0, EL0

The PMCEID0_EL0 defines which common architectural and common microarchitectural feature events are implemented.

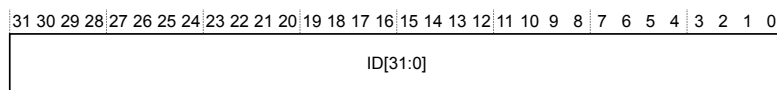
Bit field descriptions

Figure 134: PMCEID0_EL0 bit assignments

ID[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table 80: PMU common events


Bit	Event mnemonic	Description
[31]	L1D_CACHE_ALLOCATE	L1 Data cache allocate: 0 This event is not implemented.
[30]	CHAIN	Chain. For odd-numbered counters, counts once for each overflow of the preceding even-numbered counter. For even-numbered counters, does not count: 1 This event is implemented.
[29]	BUS_CYCLES	Bus cycle: 1 This event is implemented.

Bit	Event mnemonic	Description
[28]	TTBR_WRITE_RETIRED	TTBR write, architecturally executed, condition check pass - write to <i>translation table</i> base: 1 This event is implemented.
[27]	INST_SPEC	Instruction speculatively executed: 1 This event is implemented.
[26]	MEMORY_ERROR	Local memory error: 1 This event is implemented.
[25]	BUS_ACCESS	Bus access: 1 This event is implemented.
[24]	L2D_CACHE_WB	L2 Data cache <i>Write</i> -Back: 1 This event is implemented.
[23]	L2D_CACHE_REFILL	L2 Data cache refill: 1 This event is implemented.
[22]	L2D_CACHE	L2 Data cache access: 1 This event is implemented.
[21]	L1D_CACHE_WB	L1 Data cache -Back: 1 This event is implemented.
[20]	L1I_CACHE	L1 Instruction cache access: 1 This event is implemented.
[19]	MEM_ACCESS	Data memory access: 1 This event is implemented.

Bit	Event mnemonic	Description
[18]	BR_PRED	Predictable branch speculatively executed: 1 This event is implemented.
[17]	CPU_CYCLES	Cycle: 1 This event is implemented.
[16]	BR_MIS_PRED	Mispredicted or not predicted branch speculatively executed: 1 This event is implemented.
[15]	UNALIGNED_LDST_RETIRED	Instruction architecturally executed, condition check pass - unaligned load or store: 0 This event is not implemented.
[14]	BR_RETURN_RETIRED	Instruction architecturally executed, condition check pass - procedure return: 0 This event is not implemented.
[13]	BR_IMMED_RETIRED	Instruction architecturally executed - immediate branch: 0 This event is not implemented.
[12]	PC_WRITE_RETIRED	Instruction architecturally executed, condition check pass - software change of the PC: 0 This event is not implemented.
[11]	CID_WRITE_RETIRED	Instruction architecturally executed, condition check pass - write to CONTEXTIDR: 1 This event is implemented.

Bit	Event mnemonic	Description
[10]	EXC_RETURN	Instruction architecturally executed, condition check pass - <i>exception</i> return: 1 This event is implemented.
[9]	EXC_TAKEN	Exception taken: 1 This event is implemented.
[8]	INST_RETIRED	Instruction architecturally executed: 1 This event is implemented.
[7]	ST_RETIRED	Instruction architecturally executed, condition check pass - store: 0 This event is not implemented.
[6]	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 0 This event is not implemented.
[5]	L1D_ <i>TLB</i> _REFILL	L1 Data refill: 1 This event is implemented.
[4]	L1D_CACHE	L1 Data cache access: 1 This event is implemented.
[3]	L1D_CACHE_REFILL	L1 Data cache refill: 1 This event is implemented.
[2]	L1I__REFILL	L1 Instruction refill: 1 This event is implemented.
[1]	L1I_CACHE_REFILL	L1 Instruction cache refill: 1 This event is implemented.

Bit	Event mnemonic	Description
[0]	SW_INCR	Instruction architecturally executed, condition check pass - software increment: 1 This event is implemented.

 **Note:** The PMU events implemented in the above table can be found in [Table 1](#).

PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0

The PMCEID1_EL0 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

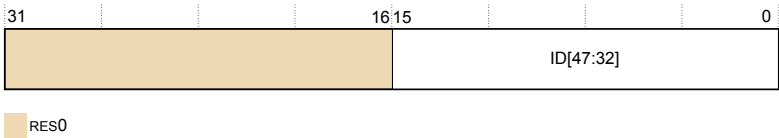


Figure 135: PMCEID1_EL0 bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
ID[47:32], [15:0]		Common architectural and microarchitectural feature events that can be counted by the PMU event counters. For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table 81: PMU common events

Bit	Event mnemonic	Description
[15]	L2D_ <i>TLB</i>	
[13]	L2D__REFILL	
[6]	L1I_	
[5]	L1D_	
[4]	STALL_BACKEND	
[3]	STALL_FRONTEND	
[2]	BR_MIS_PRED_RETIRE	
[1]	BR_RETIRE	
[0]	L2D_CACHE_ALLOCATE	



Note: The PMU events implemented in the above table can be found in [Table 1](#).

PMCR_EL0, Performance Monitors Control Register, EL0

The PMCR_EL0 provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Bit field descriptions

31	24	23	16	15	11	10	7	6	5	4	3	2	1	0	
IMP		IDCODE			N				LC	DP	X	D	C	P	E

 RES0

Figure 136: PMCR_EL0 bit assignments

IMP, [31:24]

Implementer code:

41

Arm.

This is a *read*-only field.

IDCODE, [23:16]

Identification code:

0B

Cortex[®]-A76.

This is a -only field.

N, [15:11]

Number of event counters.

0b00110

Six counters.

RES0, [10:7]

RES0

Reserved.

LC, [6]

Long cycle count enable. Determines which PMCCNTR_EL0 bit generates an overflow recorded in PMOVSr[31]. The possible values are:

0

Overflow on increment that changes PMCCNTR_EL0[31] from 1 to 0.

1

Overflow on increment that changes PMCCNTR_EL0[63] from 1 to 0.

DP, [5]

Disable cycle counter, PMCCNTR_EL0 when event counting is prohibited:

0

Cycle counter operates regardless of the non-invasive debug

		authentication settings. This is the reset value.
	1	Cycle counter is disabled if non-invasive debug is not permitted and enabled.
	This bit is /write.	
X, [4]	Export enable. This bit permits events to be exported to another debug <i>device</i> , such as a trace macrocell, over an event bus:	
	0	Export of events is disabled. This is the reset value.
	1	Export of events is enabled.
	This bit is /write and does not affect the generation of Performance Monitors interrupts on the nPMUIRQ pin.	
D, [3]	Clock divider:	
	0	When enabled, PMCCNTR_EL0 counts every clock cycle. This is the reset value.
	1	When enabled, PMCCNTR_EL0 counts every 64 clock cycles.
	This bit is /write.	
C, [2]	Clock counter reset. This bit is WO. The effects of writing to this bit are:	
	0	No action. This is the reset value.
	1	Reset PMCCNTR_EL0 to 0.
	This bit is always <i>RAZ</i> .	
	Resetting PMCCNTR_EL0 does not clear the PMCCNTR_EL0 overflow bit to 0. See the <i>Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile</i> for more information.	
P, [1]	Event counter reset. This bit is WO. The effects of writing to this bit are:	
	0	No action. This is the reset value.
	1	Reset all event counters, not including PMCCNTR_EL0, to zero.

This bit is always .

In Non-secure EL0 and EL1, a write of 1 to this bit does not reset event counters that MDCR_EL2.HPMN reserves for EL2 use.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Resetting the event counters does not clear any overflow bits to 0.

E, [0]

Enable. The possible values of this bit are:

- | | |
|----------|---|
| 0 | All counters, including PMCCNTR_EL0, are disabled. This is the reset value. |
| 1 | All counters are enabled. |

This bit is RW.

In Non-secure EL0 and EL1, this bit does not affect the operation of event counters that MDCR_EL2.HPMN reserves for EL2 use.

On *Warm reset*, the field resets to 0.

Configurations

AArch64 System register PMCR_EL0 is architecturally mapped to [AArch32](#) System register PMCR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter

25

Memory-mapped PMU registers

Topics:

- [Memory-mapped PMU register summary](#)
- [PMCFGR, Performance Monitors Configuration Register](#)
- [PMCIDR0, Performance Monitors Component Identification Register 0](#)
- [PMCIDR1, Performance Monitors Component Identification Register 1](#)
- [PMCIDR2, Performance Monitors Component Identification Register 2](#)
- [PMCIDR3, Performance Monitors Component Identification Register 3](#)
- [PMPIDR0, Performance Monitors Peripheral Identification Register 0](#)
- [PMPIDR1, Performance Monitors Peripheral Identification Register 1](#)
- [PMPIDR2, Performance Monitors Peripheral Identification Register 2](#)
- [PMPIDR3, Performance Monitors Peripheral Identification Register 3](#)
- [PMPIDR4, Performance Monitors Peripheral Identification Register 4](#)
- [PMPIDRn, Performance Monitors Peripheral Identification Register 5-7](#)

This chapter describes the memory-mapped PMU registers and shows examples of how to use them.

Memory-mapped PMU register summary

There are PMU registers that are accessible through the external debug interface.

These registers are listed in the following table. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table 82: Memory-mapped PMU register summary

Offset	Name	Type	Description
000	PMEVCNTR0_EL0	RW	Performance Monitor <i>Event</i> Count Register 0
004	-	-	Reserved
008	PMEVCNTR1_EL0	RW	Performance Monitor Count Register 1
00C	-	-	Reserved
010	PMEVCNTR2_EL0	RW	Performance Monitor Count Register 2
014	-	-	Reserved
018	PMEVCNTR3_EL0	RW	Performance Monitor Count Register 3
01C	-	-	Reserved
020	PMEVCNTR4_EL0	RW	Performance Monitor Count Register 4
024	-	-	Reserved
028	PMEVCNTR5_EL0	RW	Performance Monitor Count Register 5
0x02C-F4	-	-	Reserved
0F8	PMCCNTR_EL0[31:0]	RW	Performance Monitor Cycle Count Register
0FC	PMCCNTR_EL0[63:32]	RW	
200	PMPCSR[31:0]	RO	<i>Program Counter</i> Sample Register
204	PMPCSR[63:32]		
208	PMCID1SR	RO	CONTEXTIDR_EL1 Sample Register
20C	PMVIDSR	RO	VMID Sample Register
220	PMPCSR[31:0]	RO	Sample Register (alias)
224	PMPCSR[63:32]		
228	PMCID1SR	RO	CONTEXTIDR_EL1 Sample Register (alias)

Offset	Name	Type	Description
22C	PMCID2SR	RO	CONTEXTIDR_EL2 Sample Register
0x100–3FC	-	-	Reserved
0x418–478	-	-	Reserved
47C	PMCCFILTR_EL0	RW	Performance Monitor Cycle Count Filter Register
600	PMPCSSR_LO	RO	PMPCSSR, Snapshot Sample Register
604	PMPCSSR_HI	RO	
608	PMCIDSSR	RO	PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register
60C	PMCID2SSR	RO	PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register
610	PMSSSR	RO	PMSSSR, PMU Snapshot Status Register
614	PMOVSSR	RO	PMOVSSR, PMU Overflow Status Snapshot Register
618	PMCCNTR_LO	RO	PMCCNTR, PMU Cycle Counter Snapshot Register
61C	PMCCNTR_HI	RO	
620+ 4×n	PMEVCNTRn	RO	PMEVCNTRn, PMU Cycle Counter Snapshot Registers 0-5
6F0	PMSSCR	WO	PMSSCR, PMU Snapshot Capture Register
C00	PMCNTENSET_EL0	RW	Performance Monitor Count Enable Set Register
0xC04–C1C	-	-	Reserved
C20	PMCNTENCLR_EL0	RW	Performance Monitor Count Enable Clear Register
0xC24–C3C	-	-	Reserved
C40	PMINTENSET_EL1	RW	Performance Monitor Interrupt Enable Set Register
0xC44–C5C	-	-	Reserved
C60	PMINTENCLR_EL1	RW	Performance Monitor Interrupt Enable Clear Register
0xC64–C7C	-	-	Reserved
C80	PMOVSLR_EL0	RW	Performance Monitor Overflow Flag Status Register

Offset	Name	Type	Description
0xC84-C9C	-	-	Reserved
CA0	PMSWINC_EL0	WO	Performance Monitor Software Increment Register
0xCA4-CBC	-	-	Reserved
CC0	PMOVSSET_EL0	RW	Performance Monitor Overflow Flag Status Set Register
0xCC4-DFC	-	-	Reserved
E00	PMCFGR	RO	PMCFGR , Performance Monitors Configuration Register
E04	PMCR_EL0	RW	Performance Monitors Control Register. This register is distinct from the PMCR_EL0 system register. It does not have the same value.
0xE08-E1C	-	-	Reserved
E20	PMCEID0	RO	PMCEID0_EL0 , Performance Monitors Common Identification Register 0, EL0 (Ares/Enyo specific)
E24	PMCEID1	RO	PMCEID1_EL0 , Performance Monitors Common Identification Register 1, EL0 (Ares/Enyo Specific)
E28	PMCEID2	RO	
E2C	PMCEID3	RO	
FA4	-	-	Reserved
FA8	PMDEVAFF0	RO	MPIDR_EL1 , Multiprocessor Affinity Register, EL1
FAC	PMDEVAFF1	RO	MPIDR_EL1 , Multiprocessor Affinity Register, EL1
FB8	PMAUTHSTATUS	RO	Performance Monitor Authentication Status Register
FBC	PMDEVARCH	RO	Performance Monitor Device Architecture Register
0xFC0-FC8	-	-	Reserved
FCC	PMDEVTYPE	RO	Performance Monitor Device Type Register
FD0	PMPIDR4	RO	PMPIDR4 , Performance Monitors Peripheral Identification Register 4
FD4	PMPIDR5	RO	PMPIDRn , Performance Monitors Peripheral Identification Register 5-7
FD8	PMPIDR6	RO	

Offset	Name	Type	Description
FDC	PMPIDR7	RO	
FE0	PMPIDR0	RO	PMPIDR0, Performance Monitors Peripheral Identification Register 0
FE4	PMPIDR1	RO	PMPIDR1, Performance Monitors Peripheral Identification Register 1
FE8	PMPIDR2	RO	PMPIDR2, Performance Monitors Peripheral Identification Register 2
FEC	PMPIDR3	RO	PMPIDR3, Performance Monitors Peripheral Identification Register 3
FF0	PMCIDR0	RO	PMCIDR0, Performance Monitors Component Identification Register 0
FF4	PMCIDR1	RO	PMCIDR1, Performance Monitors Component Identification Register 1
FF8	PMCIDR2	RO	PMCIDR2, Performance Monitors Component Identification Register 2
FFC	PMCIDR3	RO	PMCIDR3, Performance Monitors Component Identification Register 3

PMCFGR, Performance Monitors Configuration Register

The PMCFGR contains PMU specific configuration data.

Bit field descriptions

The PMCFGR is a 32-bit register.

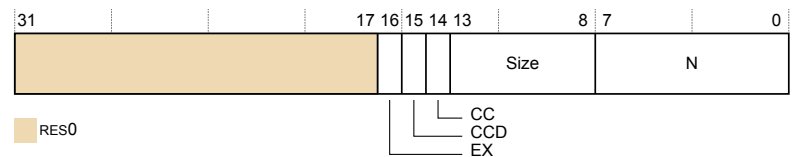


Figure 137: PMCFGR bit assignments

RES0, [31:17]	<i>res0</i>	Reserved.
EX, [16]	Export supported. The value is:	
	1	Export is supported. PMCR_EL0.EX is read/write .
CCD, [15]	Cycle counter has pre-scale. The value is:	
	1	PMCR_EL0.D is write .
CC, [14]	Dedicated cycle counter supported. The value is:	

	1	Dedicated cycle counter is supported.
Size, [13:8]	Counter size. The value is:	
	0b1111111	64-bit counters.
N, [7:0]	Number of event counters. The value is:	
	06	Six counters.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCFGR can be accessed through the external [debug interface](#), offset E00.

PMCIDR0, Performance Monitors Component Identification Register 0

The PMCIDR0 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMCIDR0 is a 32-bit register.

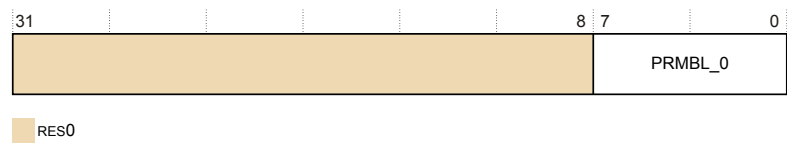


Figure 138: PMCIDR0 bit assignments

RES0, [31:8]	res0	Reserved.
PRMBL_0, [7:0]	0D	Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR0 can be accessed through the external [debug interface](#), offset FF0.

PMCIDR1, Performance Monitors Component Identification Register 1

The PMCIDR1 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMCIDR1 is a 32-bit register.

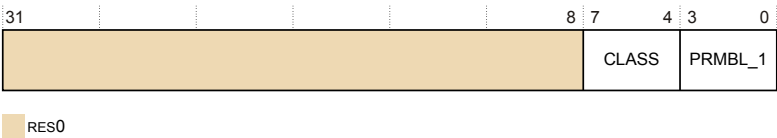


Figure 139: PMCIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
CLASS, [7:4]	9	Debug component.
PRMBL_1, [3:0]	0	Preamble byte 1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR1 can be accessed through the external [debug interface](#), offset FF4.

PMCIDR2, Performance Monitors Component Identification Register 2

The PMCIDR2 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMCIDR2 is a 32-bit register.

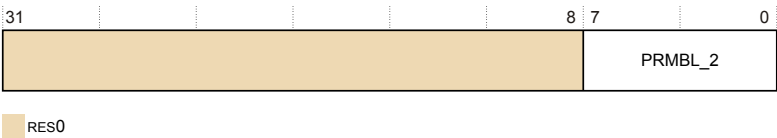


Figure 140: PMCIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_2, [7:0]	05	Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR2 can be accessed through the external [debug interface](#), offset FF8.

PMCIDR3, Performance Monitors Component Identification Register 3

The PMCIDR3 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMCIDR3 is a 32-bit register.

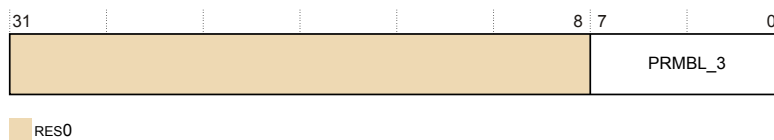


Figure 141: PMCIDR3 bit assignments

RES0, [31:8]	res0	Reserved.
PRMBL_3, [7:0]	B1	Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR3 can be accessed through the external [debug interface](#), offset FFC.

PMPIDR0, Performance Monitors Peripheral Identification Register 0

The PMPIDR0 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMPIDR0 is a 32-bit register.

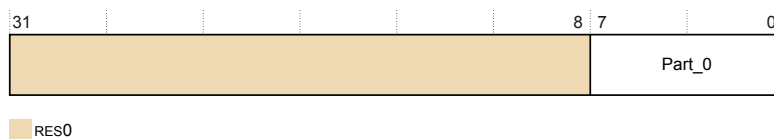


Figure 142: PMPIDR0 bit assignments

RES0, [31:8]	res0	Reserved.
Part_0, [7:0]	0B	Least significant byte of the performance monitor part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR0 can be accessed through the external [debug interface](#), offset FE0.

PMPIDR1, Performance Monitors Peripheral Identification Register 1

The PMPIDR1 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMPIDR1 is a 32-bit register.

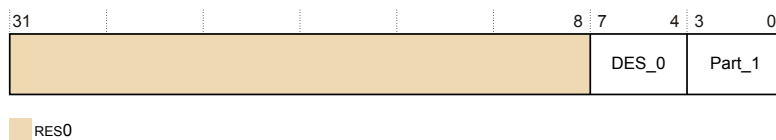


Figure 143: PMPIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
DES_0, [7:4]	B	Arm Limited. This is the least significant nibble of JEP106 ID code.
Part_1, [3:0]	D	Most significant nibble of the performance monitor part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR1 can be accessed through the external [debug interface](#), offset FE4.

PMPIDR2, Performance Monitors Peripheral Identification Register 2

The PMPIDR2 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMPIDR2 is a 32-bit register.

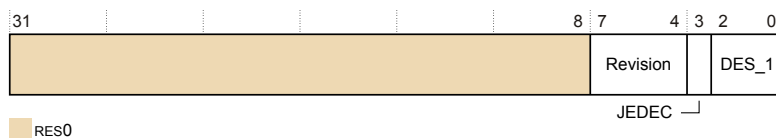


Figure 144: PMPIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Revision, [7:4]	0	r0p0.
JEDEC, [3]	0b1	RAO. Indicates a JEP106 identity code is used.
DES_1, [2:0]	0b011	Arm Limited. This is the most significant nibble of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR2 can be accessed through the external [debug interface](#), offset FE8.

PMPIDR3, Performance Monitors Peripheral Identification Register 3

The PMPIDR3 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMPIDR3 is a 32-bit register.

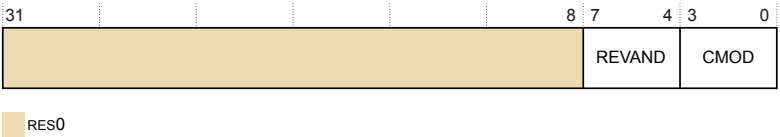


Figure 145: PMPIDR3 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
REVAND, [7:4]	0	Part minor revision.
CMOD, [3:0]	0	Customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR3 can be accessed through the external [debug interface](#), offset FEC.

PMPIDR4, Performance Monitors Peripheral Identification Register 4

The PMPIDR4 provides information to identify a Performance Monitor component.

Bit field descriptions

The PMPIDR4 is a 32-bit register.

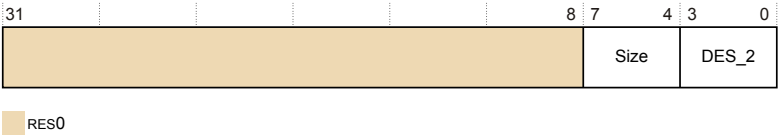


Figure 146: PMPIDR4 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Size, [7:4]	0	Size of the component. Log ₂ the number of 4KB pages from the start of the component to the

end of the component ID registers.

DES_2, [3:0]

4

Arm Limited. This is the least significant nibble JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR4 can be accessed through the external [debug interface](#), offset FD0.

PMPIDRn, Performance Monitors Peripheral Identification Register 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are *res0*.

Chapter

26

PMU snapshot registers

Topics:

- [PMU snapshot register summary](#)
- [PMPCSSR, Snapshot Program Counter Sample Register](#)
- [PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register](#)
- [PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register](#)
- [PMSSSR, PMU Snapshot Status Register](#)
- [PMOVSSR, PMU Overflow Status Snapshot Register](#)
- [PMCCNTSR, PMU Cycle Counter Snapshot Register](#)
- [PMEVCNTRn, PMU Cycle Counter Snapshot Registers 0-5](#)
- [PMSSCR, PMU Snapshot Capture Register](#)

PMU snapshot registers are an *IMPLEMENTATION DEFINED* extension to an Arm®v8-A compliant PMU to support an external core monitor that connects to a system profiler.

PMU snapshot register summary

The snapshot registers are visible in an *implementation defined* region of the PMU external debug interface. Each time the debugger sends a snapshot request, information is collected to see how the code is executed in the different cores.

The following table describes the PMU snapshot registers implemented in the [core](#).

Table 83: PMU snapshot register summary

Offset	Name	Type	Width	Description
600	PMPCSSR_LO	RO	32	PMPCSSR, Snapshot Program Counter Sample Register
604	PMPCSSR_HI	RO	32	
608	PMPCIDSSR	RO	32	PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register
60C	PMPCID2SSR	RO	32	PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register
610	PMSSSR	RO	32	PMSSSR, PMU Snapshot Status Register
614	PMOVSSR	RO	32	PMOVSSR, PMU Overflow Status Snapshot Register
618	PMCCNTSR_LO	RO	32	PMCCNTSR, PMU Cycle Counter Snapshot Register
61C	PMCCNTSR_HI	RO	32	
620 + 4×n	PMEVCNTSRn	RO	32	PMEVCNTSRn, PMU Cycle Counter Snapshot Registers 0-5
6F0	PMSSCR	WO	32	PMSSCR, PMU Snapshot Capture Register

PMPCSSR, Snapshot Program Counter Sample Register

The PMPCSSR is an alias for the PCSR register.

However, unlike the other view of PCSR, it is not sensitive to [reads](#). That is, s of PMPCSSR through the PMU snapshot view do not cause a new sample capture and do not change CIDSR, CID2SR, or VIDSR.

Bit field descriptions

The PMPCSSR is a 64-bit -only register.

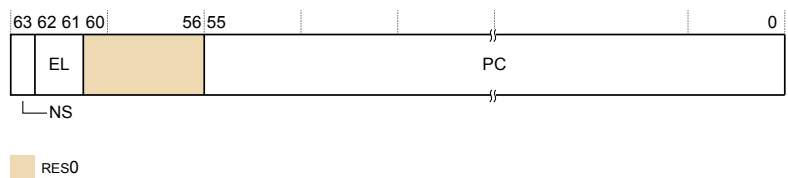


Figure 147: PMPCSSR bit assignments

NS, [63]	Non-secure sample.
EL, [62:61]	Exception level sample.
RES0, [60:56]	Reserved, <i>RES0</i> .
PC, [55:0]	Sampled PC.
Configurations	There are no configuration notes.

Usage constraints

Any access to PMPCSSR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register

The PMCIDSSR is an alias for the CIDSR register.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCIDSSR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register

The PMCID2SSR is an alias for the CID2SR register.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCID2SSR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMSSSR, PMU Snapshot Status Register

The PMSSSR holds status information about the captured counters.

Bit field descriptions

The PMSSSR is a 32-bit *read*-only register.

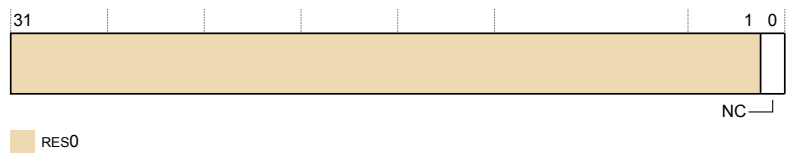


Figure 148: PMSSSR bit assignments

RES0, [31:1]	Reserved, <i>RES0</i> .				
NC, [0]	No capture. This bit indicates whether the PMU counters have been captured. The possible values are: <table><tr><td>0</td><td>PMU counters are captured.</td></tr><tr><td>1</td><td>PMU counters are not captured.</td></tr></table> If there is a security violation, the <i>core</i> does not capture the event counters. The external monitor is responsible for keeping track of whether it managed to capture the snapshot registers from the . This bit does not reflect the status of the captured <i>Program Counter</i> Sample registers. The resets this bit to 1 by a <i>Warm reset</i> but MPSSSR.NC is overwritten at the first capture.	0	PMU counters are captured.	1	PMU counters are not captured.
0	PMU counters are captured.				
1	PMU counters are not captured.				
Configurations	There are no configuration notes.				

Usage constraints

Any access to PMSSSR returns an error if any of the following occurs:

- The power domain is off.
- DoubleLockStatus() == TRUE.

PMOVSSR, PMU Overflow Status Snapshot Register

The PMOVSSR is a captured copy of PMOVSR.

Once it is captured, the value in PMOVSSR is unaffected by writes to PMOVSSET_EL0 and PMOVSCLR_EL0.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMOVSSR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMCCNTSR, PMU Cycle Counter Snapshot Register

The PMCCNTSR is a captured copy of PMCCNTR_EL0.

Once it is captured, the value in PMCCNTSR is unaffected by writes to PMCCNTR_EL0 and PMCR_EL0.C.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCCNTSR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMEVCNTRn, PMU Cycle Counter Snapshot Registers 0-5

The PMEVCNTRn, are captured copies of PMEVCNTRn_EL0, n is 0-5.

When they are captured, the value in PMSSEVCNTRn is unaffected by writes to PMSSEVCNTRn_EL0 and PMCR_EL0.P.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMSSEVCNTRn returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

PMSSCR, PMU Snapshot Capture Register

The PMSSCR provides a mechanism for software to initiate a sample.

Bit field descriptions

The PMSSCR is a 32-bit write-only register.

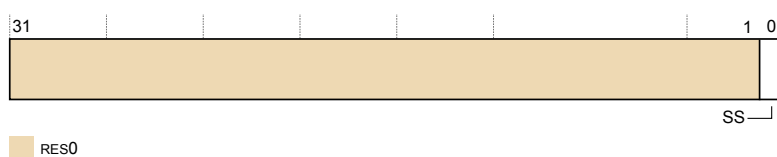


Figure 149: PMSSCR bit assignments

RES0, [31:1]

Reserved, *RES0*.

SS, [0]

Capture now. The possible values are:

0

ignored.

1

Initiate a capture immediately.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMSSCR returns an error if any of the following occurs:

- The *core* power domain is off.
- DoubleLockStatus() == TRUE.

Chapter

27

AArch64 AMU registers

Topics:

- [AArch64 AMU register summary](#)
- [AMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0](#)
- [AMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0](#)
- [AMCFGFR_EL0, Activity Monitors Configuration Register, EL0](#)
- [AMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0](#)
- [AMEVCNTRn_EL0, Activity Monitor Event Counter Register, EL0](#)
- [AMEVTYPERn_EL0, Activity Monitor Event Type Register, EL0](#)

This chapter describes the AArch64 AMU registers and shows examples of how to use them.

AArch64 AMU register summary

The following table gives a summary of the Cortex[®]-A76 AMU registers in the AArch64 Execution state.

Table 84: AArch64 AMU registers

Name	Width	Reset	Description
AMCNTENCLR_EL0	32	00000000	AMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0
AMCNTENSET_EL0	32	00000000	AMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0
AMCFGR_EL0	32	00003F04	AMCFGR_EL0, Activity Monitors Configuration Register, EL0
AMUSERENR_EL0	32	00000000	AMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0
AMEVCNTRn_EL0	64	0000000000000000	AMEVCNTRn_EL0, Activity Monitor <i>Event</i> Counter Register, EL0
AMEVTYPERn_EL0	32	<p>The reset value depends on the register:</p> <ul style="list-style-type: none"> • AMEVTYPER0_EL0 = 00000011. • AMEVTYPER1_EL0 = 000000EF. • AMEVTYPER2_EL0 = 00000008. • AMEVTYPER3_EL0 = 000000F0. • AMEVTYPER4_EL0 = 000000F1. 	AMEVTYPERn_EL0, Activity Monitor Type Register, EL0

AMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0

The AMCNTENCLR_EL0 disables the activity monitor counters implemented, AMEVCNTR0-4.

Bit field descriptions

The AMCNTENCLR_EL0 is a 32-bit register.

P<n>, bit[n]

AMEVCNTRn disable bit. The possible values are:

0

When this bit is *read*, the activity counter n is disabled. When it is written, it has no effect.

1

When this bit is , the activity counter n is enabled. When it is written, it disables the activity counter n.

Configurations

There are no configuration notes.

Usage constraints

Accessing the AMCNTENCLR_EL0

To access the AMCNTENCLR_EL0:

```
MRS <Xt>, AMCNTENCLR_EL0 ; Read
AMCNTENCLR_EL0 into Xt
MSR AMCNTENCLR_EL0, <Xt> ; Write
<Xt> to AMCNTENCLR_EL0
```

Register access is encoded as follows:

Table 85: AMCNTENCLR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	111

The AMCNTENCLR_EL0 can be accessed through the external *debug interface*, offset C20. In this case, it is -only.

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

AMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0

The AMCNTENSET_EL0 enables the activity monitor counters implemented, AMEVCNTRn (n is 0-4).

Bit field descriptions

The AMCNTENSET_EL0 is a 32-bit register.

P<n>, bit[n]

AMEVCNTRn enable bit. The possible values are:

0

When this bit is *read*, the activity counter n

is disabled. When it is written, it has no effect.

1

When this bit is , the activity counter n is enabled. When it is written, it enables the activity counter n.

Configurations

There are no configuration notes.

Usage constraints

Accessing the AMCNTENSET_EL0

To access the AMCNTENSET_EL0:

```
MRS <Xt>, AMCNTENSET_EL0 ; Read
AMCNTENSET_EL0 into Xt
MSR AMCNTENSET_EL0, <Xt> ; Write
<Xt> to AMCNTENSET_EL0
```

Register access is encoded as follows:

Table 86: AMCNTENSET_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	110

The AMCNTENSET_EL0 can be accessed through the external *debug interface*, offset C00. In this case, it is - only.

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

AMCFGR_EL0, Activity Monitors Configuration Register, EL0

The AMCFGR_EL0 provides information on the number of activity counters implemented and their size.

Bit field descriptions

The AMCFGR_EL0 is a 32-bit register.

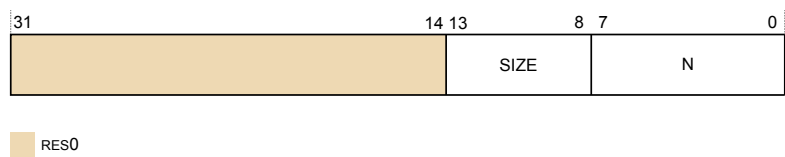


Figure 150: AMCFGR_EL0 bit assignments

RES0, [31:14]

Reserved, *RES0*.

SIZE, [13:8]

Size of counters, minus one.

This field defines the size of the largest counter implemented by the activity monitors. In the Armv8-A architecture, the largest counter has 64 bits, therefore the value of this field is 11111.

N, [7:0]

Number of activity counters implemented, where the number of counters is N+1. The Cortex®-A76 *core* implements five counters, therefore the value is 04.

Configurations

There are no configuration notes.

Usage constraints

Accessing the AMCFGR_EL0

To access the AMCFGR_EL0:

```
MRS <Xt>, AMCFGR_EL0 ; Read
AMCFGR_EL0 into Xt
```

Register access is encoded as follows:

Table 87: AMCFGR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	110

The AMCFGR_EL0 can be accessed through the external *debug interface*, offset E00. In this case, it is *read-only*.

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RO

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

AMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0

The AMUSERENR_EL0 enables or disables EL0 access to the activity monitors.

Bit field descriptions

The AMUSERENR_EL0 is a 32-bit register.

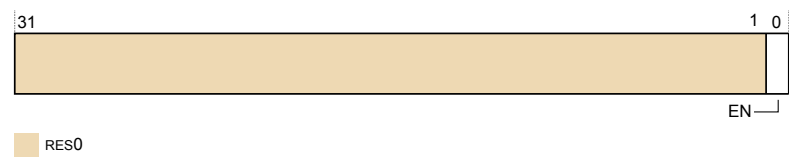


Figure 151: MUSERENR_EL0 bit assignments

RES0, [31:1]	Reserved, <i>RES0</i> .				
EN, [0]	Traps EL0 accesses to the activity monitor registers to EL1. The possible values are: <table><tr><td>0</td><td>EL0 accesses to the activity monitor registers are trapped to EL1.</td></tr><tr><td>1</td><td>EL0 accesses to the activity monitor registers are not trapped to EL1. Software can access all activity monitor registers at EL0.</td></tr></table>	0	EL0 accesses to the activity monitor registers are trapped to EL1.	1	EL0 accesses to the activity monitor registers are not trapped to EL1. Software can access all activity monitor registers at EL0.
0	EL0 accesses to the activity monitor registers are trapped to EL1.				
1	EL0 accesses to the activity monitor registers are not trapped to EL1. Software can access all activity monitor registers at EL0.				

Configurations There are no configuration notes.

Usage constraints

Accessing the AMUSERENR_EL0	To access the AMUSERENR_EL0: <pre>MRS <Xt>, AMUSERENR_EL0 ; Read AMUSERENR_EL0 into Xt MSR AMUSERENR_EL0, <Xt> ; Write Xt to AMUSERENR_EL0</pre>
-----------------------------	---

Register access is encoded as follows:

Table 88: AMUSERENR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	111

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RW	RW	RW



Note: AMUSERENR_EL0 is always RO at EL0 and not trapped by the EN bit.

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

AMEVCNTRn_EL0, Activity Monitor Event Counter Register, EL0

The activity counters AMEVCNTRn_EL0 are directly accessible in the memory mapped-view. n is 0-4.

Bit field descriptions

The AMEVCNTRn_EL0 is a 64-bit register.

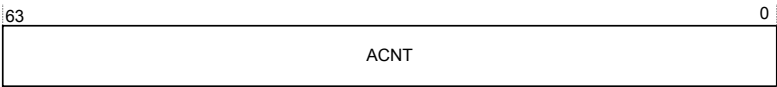


Figure 152: AMEVCNTRn_EL0 bit assignments

ACNT, [63:0]

Value of the activity counter AMEVCNTRn_EL0.

This bit field resets to zero and the counters monitoring cycle events do not increment when the *core* is in WFI or WFE.

Configurations

Counters might have fixed event allocation.

Usage constraints

Accessing the AMEVCNTRn_EL0

To access the AMEVCNTRn_EL0:

```
MRS <Xt>, AMEVCNTRn_EL0 ; Read
AMEVCNTRn_EL0 into Xt
MSR AMEVCNTRn_EL0, <Xt> ; Write Xt
to AMEVCNTRn_EL0
```

Register access is encoded as follows:

Table 89: AMEVCNTRn_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	<0-4>

The AMEVCNTRn_EL0[63:32] can also be accessed through the external memory-mapped interface, offset 004+8n. In this case, it is read-only.

The AMEVCNTRn_EL0[31:0] can also be accessed through the external memory-mapped interface, offset 000+8n. In this case, it is read-only.

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

AMEVTYPERN_EL0, Activity Monitor Event Type Register, EL0

The activity counters AMEVTYPERN_EL0n are directly accessible in the memory mapped view, where n is 0-4.

Bit field descriptions

The AMEVTYPERN_EL0 is a 32-bit register.

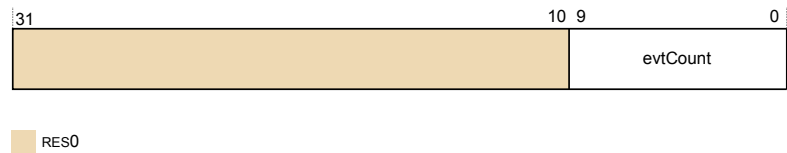


Figure 153: AMEVTYPERN_EL0 bit assignments

RES0, [31:10]	Reserved, <i>RES0</i> .
evtCount, bits[9:0]	The event the counter monitors might be fixed at implementation. In this case, the field is <i>read</i> -only. See AMU events .
Configurations	Counters might have fixed event allocation.

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Usage constraints

Accessing the AMEVTYPERN_EL0

To access the AMEVTYPERN_EL0:

```
MRS <Xt>, AMEVTYPERN_EL0 ; Read
AMEVTYPERN_EL0 into Xt
MSR AMEVTYPERN_EL0, <Xt> ; Write Xt
to AMEVTYPERN_EL0
```

Register access is encoded as follows:

Table 90: AMEVTYPER_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	<0-4>

This register can also be accessed through the external memory-mapped interface, offset 400+4n. In this case, it is -only.

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RO

Traps and enables

If ACTLR_EL2.AMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.AMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Chapter

28

ETM registers

Topics:

- [TRCACATRn, Address Comparator Access Type Registers 0-7](#)
- [TRCACVRn, Address Comparator Value Registers 0-7](#)
- [TRCAUTHSTATUS, Authentication Status Register](#)
- [TRCAUXCTLR, Auxiliary Control Register](#)
- [TRCBBCTLR, Branch Broadcast Control Register](#)
- [TRCCCCTLR, Cycle Count Control Register](#)
- [TRCCIDCCTLR0, Context ID Comparator Control Register 0](#)
- [TRCCIDCVR0, Context ID Comparator Value Register 0](#)
- [TRCCIDR0, ETM Component Identification Register 0](#)
- [TRCCIDR1, ETM Component Identification Register 1](#)
- [TRCCIDR2, ETM Component Identification Register 2](#)
- [TRCCIDR3, ETM Component Identification Register 3](#)
- [TRCCLAIMCLR, Claim Tag Clear Register](#)
- [TRCCLAIMSET, Claim Tag Set Register](#)
- [TRCCNTCTLR0, Counter Control Register 0](#)
- [TRCCNTCTLR1, Counter Control Register 1](#)
- [TRCCNTRLDVRn, Counter Reload Value Registers 0-1](#)
- [TRCCNTVRn, Counter Value Registers 0-1](#)

This chapter describes the ETM registers.

- TRCCONFIGR, Trace Configuration Register
- TRCDEVAFF0, Device Affinity Register 0
- TRCDEVAFF1, Device Affinity Register 1
- TRCDEVARCH, Device Architecture Register
- TRCDEVID, Device ID Register
- TRCDEVTYPE, Device Type Register
- TRCEVENTCTL0R, Event Control 0 Register
- TRCEVENTCTL1R, Event Control 1 Register
- TRCEXTINSEL, External Input Select Register
- TRCIDR0, ID Register 0
- TRCIDR1, ID Register 1
- TRCIDR2, ID Register 2
- TRCIDR3, ID Register 3
- TRCIDR4, ID Register 4
- TRCIDR5, ID Register 5
- TRCIDR8, ID Register 8
- TRCIDR9, ID Register 9
- TRCIDR10, ID Register 10
- TRCIDR11, ID Register 11
- TRCIDR12, ID Register 12
- TRCIDR13, ID Register 13
- TRCIMSPEC0, Implementation Specific Register 0
- TRCITATBIDR, Integration ATB Identification Register
- TRCITCTRL, Integration Mode Control Register
- TRCITIATBINR, Integration Instruction ATB In Register
- TRCITIATBOUTR, Integration Instruction ATB Out Register
- TRCITIDATAR, Integration Instruction ATB Data Register
- TRCLAR, Software Lock Access Register
- TRCLSR, Software Lock Status Register
- TRCCNTVRn, Counter Value Registers 0-1
- TRCOSLAR, OS Lock Access Register

- TRCOSLSR, OS Lock Status Register
- TRCPDCR, Power Down Control Register
- TRCPDSR, Power Down Status Register
- TRCPIDR0, ETM Peripheral Identification Register 0
- TRCPIDR1, ETM Peripheral Identification Register 1
- TRCPIDR2, ETM Peripheral Identification Register 2
- TRCPIDR3, ETM Peripheral Identification Register 3
- TRCPIDR4, ETM Peripheral Identification Register 4
- TRCPIDRn, ETM Peripheral Identification Registers 5-7
- TRCPRGCTLR, Programming Control Register
- TRCRSCTLRn, Resource Selection Control Registers 2-16
- TRCSEQEVRn, Sequencer State Transition Control Registers 0-2
- TRCSEQRSTEV, Sequencer Reset Control Register
- TRCSEQSTR, Sequencer State Register
- TRCSSCCR0, Single-Shot Comparator Control Register 0
- TRCSSCSR0, Single-Shot Comparator Status Register 0
- TRCSTALLCTLR, Stall Control Register
- TRCSTATR, Status Register
- TRCSYNCP, Synchronization Period Register
- TRCTRACEIDR, Trace ID Register
- TRCTSCTLR, Global Timestamp Control Register
- TRCVICTLR, ViewInst Main Control Register
- TRCVIIECTLR, ViewInst Include-Exclude Control Register
- TRCVISSCTLR, ViewInst Start-Stop Control Register

- TRCVMIDCVR0, VMID Comparator Value Register 0
- TRCVMIDCCTLR0, Virtual context identifier Comparator Control Register 0

TRCACATRn, Address Comparator Access Type Registers 0-7

The TRCACATRn control the access for the corresponding address comparators.

Bit field descriptions

The TRCACATRn is a 64-bit register.

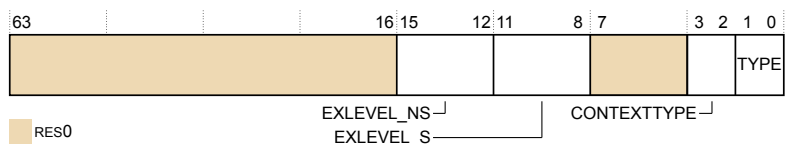


Figure 154: TRCACATRn bit assignments

RES0, [63:16]

res0

Reserved.

EXLEVEL_NS, [15:12]

Each bit controls whether a comparison can occur in Non-secure state for the corresponding *Exception level*. The possible values are:

0

The trace unit can perform a comparison, in Non-secure state, for *n*.

1

The trace unit does not perform a comparison, in Non-secure state, for *n*.

The s are:

Bit[12]

0.

Bit[13]

1.

Bit[14]

2.

Bit[15]

Always *res0*.

EXLEVEL_S, [11:8]

Each bit controls whether a comparison can occur in Secure state for the corresponding . The possible values are:

0

The trace unit can perform a comparison, in Secure state, for *n*.

1

The trace unit does not perform a comparison, in Secure state, for *n*.

The s are:

Bit[8]

0.

Bit[9]

1.

Bit[10]

Always *res0*.

	Bit[11]	3.
RES0, [7:4]	res0	Reserved.
CONTEXT TYPE, [3:2]	Controls whether the trace unit performs a Context ID comparison, a VMID comparison, or both comparisons:	
	0b00	The trace unit does not perform a Context ID comparison.
	0b01	The trace unit performs a Context ID comparison using the Context ID comparator that the CONTEXT field specifies, and signals a match if both the Context ID comparator matches and the address comparator match.
	0b10	The trace unit performs a VMID comparison using the VMID comparator that the CONTEXT field specifies, and signals a match if both the VMID comparator and the address comparator match.
	0b11	The trace unit performs a Context ID comparison and a VMID comparison using the comparators that the CONTEXT field specifies, and signals a match if the Context ID comparator matches, the VMID comparator matches, and the address comparator matches.
TYPE, [1:0]	Type of comparison:	
	0b00	Instruction address, res0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCACATR*n* can be accessed through the external [debug interface](#), offset 0x480–4B8.

TRCACVRn, Address Comparator Value Registers 0-7

The TRCACVRn indicate the address for the address comparators.

Bit field descriptions

The TRCACVRn is a 64-bit register.

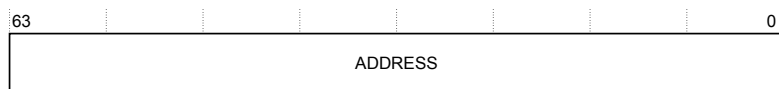


Figure 155: TRCACVRn bit assignments

ADDRESS, [63:0]

The address value to compare against.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCACVRn can be accessed through the external [debug interface](#), offset 400-43C.

TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS indicates the current level of tracing permitted by the system.

Bit field descriptions

The TRCAUTHSTATUS is a 64-bit register.

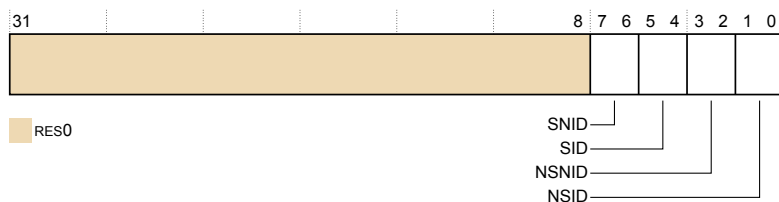


Figure 156: TRCAUTHSTATUS bit assignments

RES0, [31:8]

res0

Reserved.

SNID, [7:6]

Secure Non-invasive Debug:

0b10

Secure Non-invasive Debug implemented but disabled.

0b11

Secure Non-invasive Debug implemented and enabled.

SID, [5:4]

Secure Invasive Debug:

0b00

Secure Invasive Debug is not implemented.

NSNID, [3:2]

Non-secure Non-invasive Debug:

0b10

Non-secure Non-invasive Debug implemented but disabled, NIDEN=0.

0b11

Non-secure Non-invasive Debug implemented and enabled, NIDEN=1.

NSID, [1:0]

Non-secure Invasive Debug:

0b00

Non-secure Invasive Debug is not implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCAUTHSTATUS can be accessed through the external [debug interface](#), offset FB8.

TRCAUXCTLR, Auxiliary Control Register

The TRCAUXCTLR provides *implementation defined* configuration and control options.

Bit field descriptions

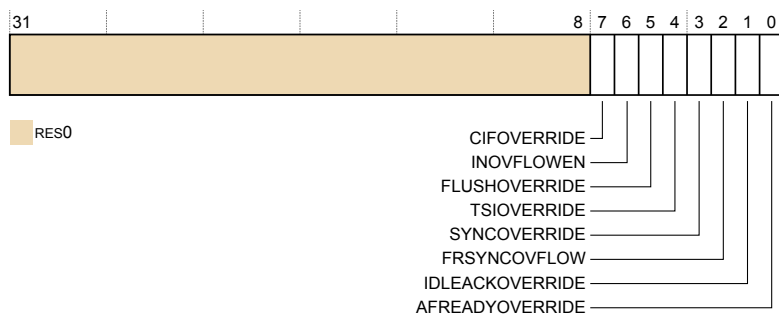


Figure 157: TRCAUXCTLR bit assignments

RES0, [31:8]

res0

Reserved.

CIFOVERRIDE, [7]

Override [core](#) interface register repeater clock enable. The possible values are:

0

Core interface clock gate is enabled.

1

Core interface clock gate is disabled.

INOVFLOWEN, [6]

Allow overflows of the interface buffer, removing any rare impact that the trace unit might have on the 's speculation when enabled. The possible values are:

0

Core interface buffer overflows are disabled.

FLUSHOVERRIDE, [5]

1 Core interface buffer overflows are enabled.

When this bit is set to 1, the trace start/stop logic might deviate from architecturally-specified behavior.

Override *ETM* flush behavior. The possible values are:

0 trace unit FIFO is flushed and trace unit enters idle state when DBGEN or NIDEN is LOW.

1 trace unit FIFO is not flushed and trace unit does not enter idle state when DBGEN or NIDEN is LOW.

When this bit is set to 1, the trace unit behavior deviates from architecturally-specified behavior.

TSIOVERRIDE, [4]

Override TS packet insertion behavior. The possible values are:

0 Timestamp packets are inserted into FIFO only when trace activity is LOW.

1 Timestamp packets are inserted into FIFO irrespective of trace activity.

SYNCOVERRIDE, [3]

Override SYNC packet insertion behavior. The possible values are:

0 SYNC packets are inserted into FIFO only when trace activity is low.

1 SYNC packets are inserted into FIFO irrespective of trace activity.

FRSYNCOVFLOW, [2]

Force overflows to output synchronization packets. The possible values are:

0 No FIFO overflow when SYNC packets are delayed.

1 Forces FIFO overflow when SYNC packets are delayed.

When this bit is set to 1, the trace unit behavior deviates from architecturally-specified behavior.

IDLEACKOVERRIDE, [1]

- Force idle acknowledge. The possible values are:
- 0** trace unit idle acknowledge is asserted only when the trace unit is in idle state.
 - 1** trace unit idle acknowledge is asserted irrespective of the trace unit idle state.

When this bit is set to 1, trace unit behavior deviates from architecturally-specified behavior.

AFREADYOVERRIDE, [0]

- Force assertion of AFREADYM output. The possible values are:
- 0** trace unit AFREADYM output is asserted only when the trace unit is in idle state or when all the trace bytes in FIFO before a flush request are output.
 - 1** trace unit AFREADYM output is always asserted HIGH.

When this bit is set to 1, trace unit behavior deviates from architecturally-specified behavior.

The TRCAUXCTLR can be accessed through the internal memory-mapped interface and the external *debug interface*, offset 018.

Configurations Available in all configurations.

TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR controls how branch broadcasting behaves, and allows branch broadcasting to be enabled for certain memory regions.

Bit field descriptions

The TRCAUXCTLR is a 32-bit register.

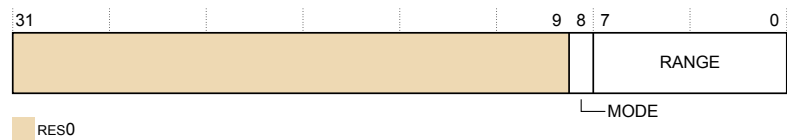


Figure 158: TRCBBCTLR bit assignments

RES0, [31:9]	<i>res0</i>	Reserved.
--------------	-------------	-----------

MODE, [8]

Mode bit:

0	<p>Exclude mode. Branch broadcasting is not enabled in the address range that RANGE defines.</p> <p>If RANGE==0 then branch broadcasting is enabled for the entire memory map.</p>
1	<p>Include mode. Branch broadcasting is enabled in the address range that RANGE defines.</p> <p>If RANGE==0 then the behavior of the trace unit is <i>constrained unpredictable</i>. That is, the trace unit might or might not consider any instructions to be in a branch broadcast <i>region</i>.</p>

RANGE, [7:0]

Address range field.

Selects which address range comparator pairs are in use with branch broadcasting. Each bit represents an address range comparator pair, so bit[*n*] controls the selection of address range comparator pair *n*. If bit[*n*] is:

0	The address range that address range comparator pair <i>n</i> defines, is not selected.
1	The address range that address range comparator pair <i>n</i> defines, is selected.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCBBCTLR can be accessed through the external *debug interface*, offset 03C.

TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR sets the threshold value for cycle counting.

Bit field descriptions

The TRCCCCTLR is a 32-bit register.

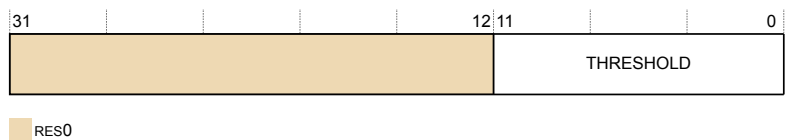


Figure 159: TRCCCCTLR bit assignments

RES0, [31:12] *res0* Reserved.

THRESHOLD, [11:0] Instruction trace cycle count threshold.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCCCTLR can be accessed through the external [debug interface](#), offset 038.

TRCCIDCCTLR0, Context ID Comparator Control Register 0

The TRCCIDCCTLR0 controls the mask value for the context ID comparators.

Bit field descriptions

The TRCCIDCCTLR0 is a 32-bit register.

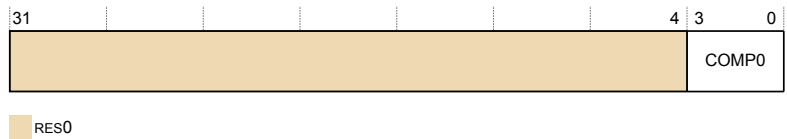


Figure 160: TRCCIDCCTLR0 bit assignments

RES0, [31:4] *res0* Reserved.

COMP0, [3:0] Controls the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0. When a bit is:

0 The trace unit includes the relevant byte in TRCCIDCVR0 when it performs the Context ID comparison.

1 The trace unit ignores the relevant byte in TRCCIDCVR0 when it performs the Context ID comparison.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDCCTLR0 can be accessed through the external [debug interface](#), offset 680.

TRCCIDCVR0, Context ID Comparator Value Register 0

The TRCCIDCVR0 contains a Context ID value.

Bit field descriptions

The TRCCIDCVR0 is a 64-bit register.



Figure 161: TRCCIDCVR0 bit assignments

RES0, [63:32]	<i>res0</i>	Reserved.
VALUE, [31:0]		The data value to compare against.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDCVR0 can be accessed through the external [debug interface](#), offset 600.

TRCCIDR0, ETM Component Identification Register 0

The TRCCIDR0 provides information to identify a trace component.

Bit field descriptions

The TRCCIDR0 is a 32-bit register.

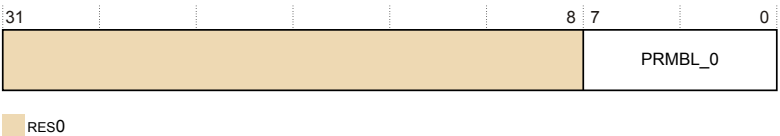


Figure 162: TRCCIDR0 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_0, [7:0]	0D	Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR0 can be accessed through the external [debug interface](#), offset FF0.

TRCCIDR1, ETM Component Identification Register 1

The TRCCIDR1 provides information to identify a trace component.

Bit field descriptions

The TRCCIDR1 is a 32-bit register.

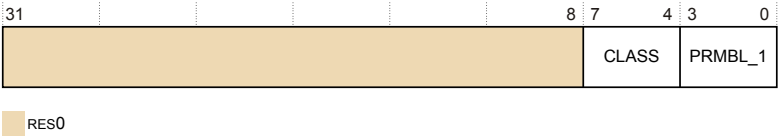


Figure 163: TRCCIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
CLASS, [7:4]	9	Debug component.
PRMBL_1, [3:0]	0	Preamble byte 1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR1 can be accessed through the external [debug interface](#), offset FF4.

TRCCIDR2, ETM Component Identification Register 2

The TRCCIDR2 provides information to identify a CTI component.

Bit field descriptions

The TRCCIDR2 is a 32-bit register.

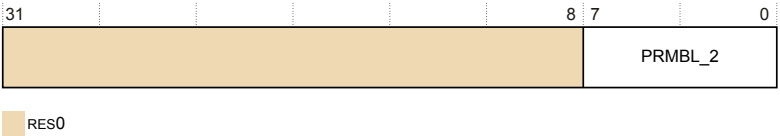


Figure 164: TRCCIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_2, [7:0]	05	Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR2 can be accessed through the external [debug interface](#), offset FF8.

TRCCIDR3, ETM Component Identification Register 3

The TRCCIDR3 provides information to identify a trace component.

Bit field descriptions

The TRCCIDR3 is a 32-bit register.

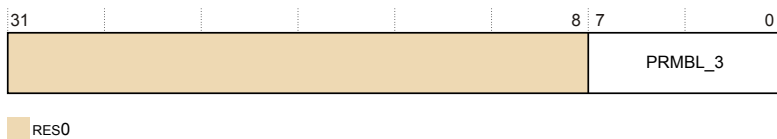


Figure 165: TRCCIDR3 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
PRMBL_3, [7:0]	B1	Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR3 can be accessed through the external [debug interface](#), offset FFC.

TRCCLAIMCLR, Claim Tag Clear Register

The TRCCLAIMCLR clears bits in the claim tag and determines the current value of the claim tag.

Bit field descriptions

The TRCCLAIMCLR is a 32-bit register.

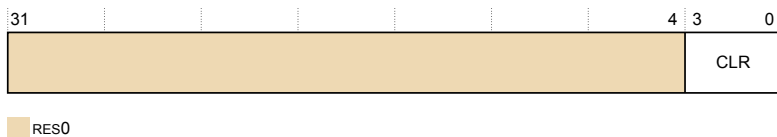


Figure 166: TRCCLAIMCLR bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
CLR, [3:0]	On <i>reads</i> , for each bit:	
	0	Claim tag bit is not set.
	1	Claim tag bit is set.
	On <i>writes</i> , for each bit:	
	0	Has no effect.
	1	Clears the relevant bit of the claim tag.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCLAIMCLR can be accessed through the external [debug interface](#), offset FA4.

TRCCLAIMSET, Claim Tag Set Register

The TRCCLAIMSET sets bits in the claim tag and determines the number of claim tag bits implemented.

Bit field descriptions

The TRCCLAIMSET is a 32-bit register.

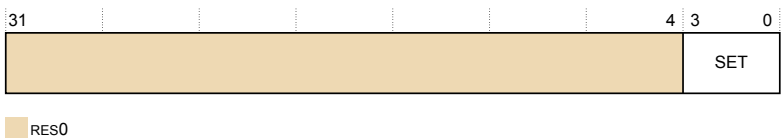


Figure 167: TRCCLAIMSET bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
SET, [3:0]	On <i>reads</i> , for each bit:	
	0	Claim tag bit is not implemented.
	1	Claim tag bit is implemented.
	On <i>writes</i> , for each bit:	
	0	Has no effect.
	1	Sets the relevant bit of the claim tag.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

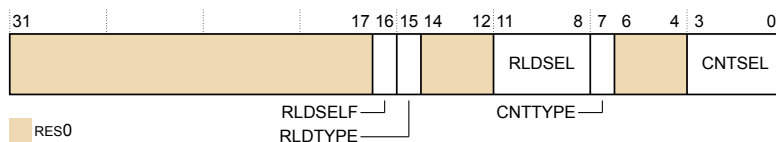
The TRCCLAIMSET can be accessed through the external [debug interface](#), offset FA0.

TRCCNTCTLR0, Counter Control Register 0

The TRCCNTCTLR0 controls the counter.

Bit field descriptions

The TRCCNTCTLR0 is a 32-bit register.

**Figure 168: TRCCNTCTLR0 bit assignments****RES0, [31:17]***res0*

Reserved.

RLDSELF, [16]

Defines whether the counter reloads when it reaches zero:

0

The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL.

1

The counter reloads when it reaches zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.

RLDTYPE, [15]

Selects the resource type for the reload:

0

Single selected resource.

1

Boolean combined resource pair.

RES0, [14:12]*res0*

Reserved.

RLDSEL, [11:8]

Selects the resource number, based on the value of RLDTYPE:

When RLDTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When RLDTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

CNTTYPE, [7]

Selects the resource type for the counter:

0

Single selected resource.

1

Boolean combined resource pair.

RES0, [6:4]*res0*

Reserved.

CNTSEL, [3:0]

Selects the resource number, based on the value of CNTTYPE:

When CNTTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When CNTTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTCTLR0 can be accessed through the external [debug interface](#), offset 150.

TRCCNTCTLR1, Counter Control Register 1

The TRCCNTCTLR1 controls the counter.

Bit field descriptions

The TRCCNTCTLR1 is a 32-bit register.

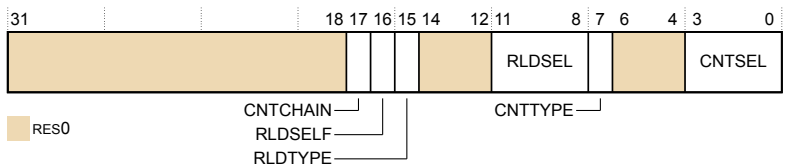


Figure 169: TRCCNTCTLR1 bit assignments

RES0, [31:18]	<i>res0</i>	Reserved.
CNTCHAIN, [17]		
	0	The counter operates independently from the counter. The counter only decrements based on CNTTYPE and CNTSEL.
	1	The counter decrements when the counter reloads. The counter also decrements when the resource selected by CNTTYPE and CNTSEL is active.
RLDSELF, [16]		
	0	Defines whether the counter reloads when it reaches zero: The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL.

	1	The counter reloads when it is zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.
RLDTYPE, [15]	Selects the resource type for the reload:	
	0	Single selected resource.
	1	Boolean combined resource pair.
RES0, [14:12]	<i>res0</i>	Reserved.
RLDSEL, [11:8]	Selects the resource number, based on the value of RLDTYPE:	
	When RLDTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].	
	When RLDTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].	
CNTTYPE, [7]	Selects the resource type for the counter:	
	0	Single selected resource.
	1	Boolean combined resource pair.
RES0, [6:4]	<i>res0</i>	Reserved.
CNTSEL, [3:0]	Selects the resource number, based on the value of CNTTYPE:	
	When CNTTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].	
	When CNTTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTCTLR1 can be accessed through the external [debug interface](#), offset 154.

TRCCNTRLDVRn, Counter Reload Value Registers 0-1

The TRCCNTRLDVRn define the reload value for the counter.

Bit field descriptions

The TRCCNTRLDVRn is a 32-bit register.

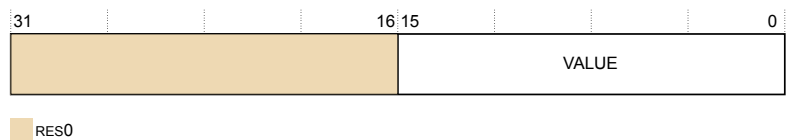


Figure 170: TRCCNTRLDVRn bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
VALUE, [15:0]	Defines the reload value for the counter. This value is loaded into the counter each time the reload event occurs.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTRLDVRn registers can be accessed through the external [debug interface](#), offsets:

TRCCNTRLDVR0	140.
TRCCNTRLDVR1	144.

TRCCNTRn, Counter Value Registers 0-1

The TRCCNTRn contain the current counter value.

Bit field descriptions

The TRCCNTRn is a 32-bit register.

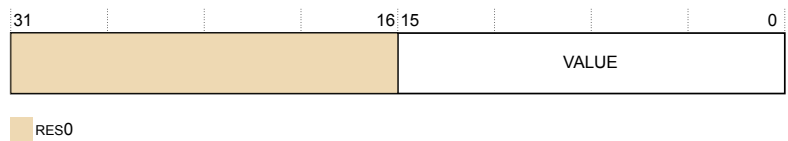


Figure 171: TRCCNTRn bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
VALUE, [15:0]	Contains the current counter value.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTRLDVRn registers can be accessed through the external [debug interface](#), offsets:

TRCCNTR0	160.
TRCCNTR1	164.

TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR controls the tracing options.

Bit field descriptions

The TRCCONFIGR is a 32-bit register.

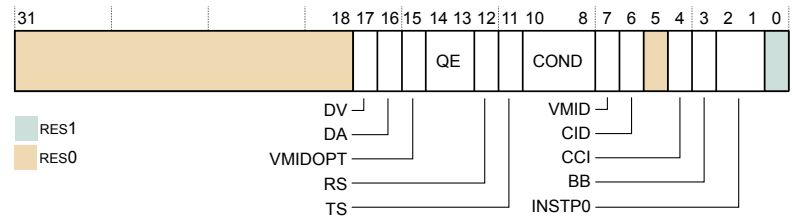


Figure 172: TRCCONFIGR bit assignments

RES0, [31:18]

res0

Reserved.

DV, [17]

Enables data value tracing. The possible values are:

0

Disables data value tracing.

1

Enables data value tracing.

DA, [16]

Enables data address tracing. The possible values are:

0

Disables data address tracing.

1

Enables data address tracing.

VMIDOPT, [15]

Configures the Virtual context identifier value used by the trace unit, both for trace generation and in the Virtual context identifier comparators. The possible values are:

0b0

VTTBR_EL2.VMID is used. If the trace unit supports a Virtual context identifier larger than the VTTBR_EL2.VMID, the upper unused bits are always zero. If the trace unit supports a Virtual context identifier larger than 8 bits and if the VTCR_EL2.VS bit forces use of an 8-bit Virtual context identifier, bits [15:8] of the trace unit Virtual context identifier are always zero.

	0b1	CONTEXTIDR_EL2 is used. TRCIDR2.VMIDOPT indicates whether this field is implemented.
QE, [14:13]	Enables Q element. The possible values are:	
	0b00	Q elements are disabled.
	0b01	Q elements with instruction counts are disabled. Q elements without instruction counts are disabled.
	0b10	Reserved.
	0b11	Q elements with and without instruction counts are enabled.
RS, [12]	Enables the return stack. The possible values are:	
	0	Disables the return stack.
	1	Enables the return stack.
TS, [11]	Enables global timestamp tracing. The possible values are:	
	0	Disables global timestamp tracing.
	1	Enables global timestamp tracing.
COND, [10:8]	Enables conditional instruction tracing. The possible values are:	
	0b000	Conditional instruction tracing is disabled.
	0b001	Conditional load instructions are traced.
	0b010	Conditional store instructions are traced.
	0b011	Conditional load and store instructions are traced.
	0b111	All conditional instructions are traced.
VMID, [7]	Enables VMID tracing. The possible values are:	
	0	Disables VMID tracing.
	1	Enables VMID tracing.

CID, [6]	Enables context ID tracing. The possible values are:	
	0	Disables context ID tracing.
	1	Enables context ID tracing.
RES0, [5]	<i>res0</i>	Reserved.
CCI, [4]	Enables cycle counting instruction trace. The possible values are:	
	0	Disables cycle counting instruction trace.
	1	Enables cycle counting instruction trace.
BB, [3]	Enables branch broadcast mode. The possible values are:	
	0	Disables branch broadcast mode.
	1	Enables branch broadcast mode.
INSTRP0, [2:1]	Controls whether load and store instructions are traced as P0 instructions. The possible values are:	
	0b00	Load and store instructions are not traced as P0 instructions.
	0b01	Load instructions are traced as P0 instructions.
	0b10	Store instructions are traced as P0 instructions.
	0b11	Load and store instructions are traced as P0 instructions.
RES1, [0]	<i>res1</i>	Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCONFIGR can be accessed through the external [debug interface](#), offset 010.

TRCDEVAFF0, Device Affinity Register 0

The TRCDEVAFF0 provides an additional core identification mechanism for scheduling purposes in a cluster. TRCDEVAFF0 is a read-only copy of MPIDR accessible from the external debug interface.

Bit field descriptions

The TRCDEVAFF0 is a 32-bit register and is a copy of the MPIDR register. See [MPIDR_EL1, Multiprocessor Affinity Register, EL1](#) for full bit field descriptions.

TRCDEVAFF1, Device Affinity Register 1

The TRCDEVAFF1 is a read-only copy of MPIDR_EL1[63:32] as seen from EL3, unaffected by VMPIDR_EL2.

TRCDEVARCH, Device Architecture Register

The TRCDEVARCH identifies the ETM trace unit as an ETMv4 component.

Bit field descriptions

The TRCDEVARCH is a 32-bit register.

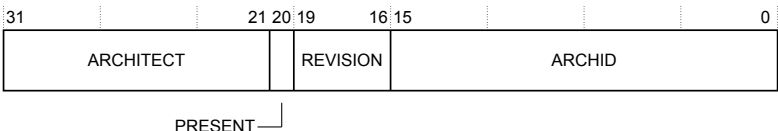


Figure 173: TRCDEVARCH bit assignments

ARCHITECT, [31:21]	Defines the architect of the component:
4	Arm JEP continuation.
3B	Arm JEP 106 code.
PRESENT, [20]	Indicates the presence of this register:
0b1	Register is present.
REVISION, [19:16]	Architecture revision:
02	Architecture revision 2.
ARCHID, [15:0]	Architecture ID:
4A13	ETMv4 component.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVARCH can be accessed through the external [debug interface](#), offset FBC.

TRCDEVID, Device ID Register

The TRCDEVID indicates the capabilities of the ETM trace unit.

Bit field descriptions

The TRCDEVID is a 32-bit register.

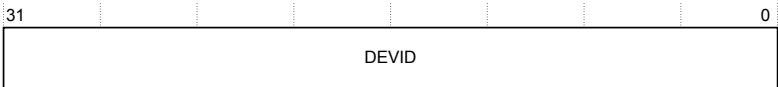


Figure 174: TRCDEVID bit assignments

DEVID, [31:0] RAZ. There are no component-defined capabilities.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVID can be accessed through the external [debug interface](#), offset FC8.

TRCDEVTYPE, Device Type Register

The TRCDEVTYPE indicates the type of the component.

Bit field descriptions

The TRCDEVTYPE is a 32-bit register.

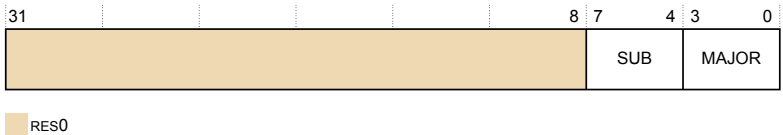


Figure 175: TRCDEVTYPE bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
SUB, [7:4]	The sub-type of the component:	
	0b0001	Core trace.
MAJOR, [3:0]	The main type of the component:	
	0b0011	Trace source.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVTYPE can be accessed through the external [debug interface](#), offset FCC.

TRCEVENTCTL0R, Event Control 0 Register

The TRCEVENTCTL0R controls the tracing of events in the trace stream. The events also drive the external outputs from the ETM trace unit. The events are selected from the Resource Selectors.

Bit field descriptions

The TRCEVENTCTL0R is a 32-bit register.

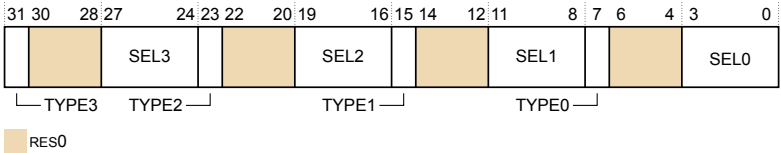


Figure 176: TRCEVENTCTL0R bit assignments

TYPE3, [31]	<p>Selects the resource type for trace event 3:</p> <p>0 Single selected resource.</p> <p>1 Boolean combined resource pair.</p>
RES0, [30:28]	<p><i>res0</i> Reserved.</p>
SEL3, [27:24]	<p>Selects the resource number, based on the value of TYPE3:</p> <p>When TYPE3 is 0, selects a single selected resource from 0-15 defined by bits[3:0].</p> <p>When TYPE3 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].</p>
TYPE2, [23]	<p>Selects the resource type for trace event 2:</p> <p>0 Single selected resource.</p> <p>1 Boolean combined resource pair.</p>
RES0, [22:20]	<p><i>res0</i> Reserved.</p>
SEL2, [19:16]	<p>Selects the resource number, based on the value of TYPE2:</p> <p>When TYPE2 is 0, selects a single selected resource from 0-15 defined by bits[3:0].</p> <p>When TYPE2 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].</p>
TYPE1, [15]	<p>Selects the resource type for trace event 1:</p> <p>0 Single selected resource.</p>

	1	Boolean combined resource pair.
RES0, [14:12]	<i>res0</i>	Reserved.
SEL1, [11:8]	Selects the resource number, based on the value of TYPE1: When TYPE1 is 0, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].	
TYPE0, [7]	Selects the resource type for trace event 0: 0 Single selected resource. 1 Boolean combined resource pair.	
RES0, [6:4]	<i>res0</i>	Reserved.
SEL0, [3:0]	Selects the resource number, based on the value of TYPE0: When TYPE0 is 0, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE0 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEVENTCTL0R can be accessed through the external [debug interface](#), offset 020.

TRCEVENTCTL1R, Event Control 1 Register

The TRCEVENTCTL1R controls the behavior of the events that TRCEVENTCTL0R selects.

Bit field descriptions

The TRCEVENTCTL1R is a 32-bit register.

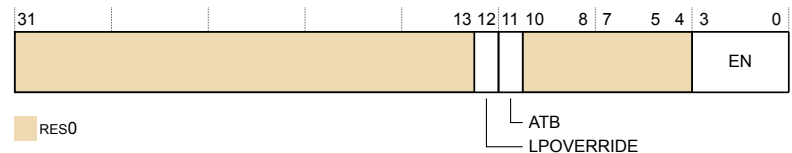


Figure 177: TRCEVENTCTL1R bit assignments

RES0, [31:13]	<i>res0</i>	Reserved.
LPOVERRIDE, [12]	Low-power state behavior override:	

	0	Low-power state behavior unaffected.
	1	Low-power state behavior overridden. The resources and <i>Event</i> trace generation are unaffected by entry to a low-power state.
ATB, [11]	ATB trigger enable:	
	0	ATB trigger disabled.
	1	ATB trigger enabled.
RES0, [10:4]	<i>res0</i>	Reserved.
EN, [3:0]	One bit per event, to enable generation of an event element in the instruction trace stream when the selected event occurs:	
	0	does not cause an event element.
	1	causes an event element.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEVENTCTL1R can be accessed through the external [debug interface](#), offset 024.

TRCEXTINSELR, External Input Select Register

The TRCEXTINSELR controls the selectors that choose an external input as a resource in the ETM trace unit. You can use the Resource Selectors to access these external input resources.

Bit field descriptions

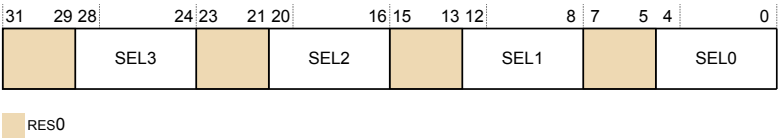


Figure 178: TRCEXTINSELR bit assignments

RES0, [31:29]	<i>res0</i>	Reserved.
SEL3, [28:24]	Selects an event from the external input bus for External Input Resource 3.	
RES0, [23:21]	<i>res0</i>	Reserved.
SEL2, [20:16]	Selects an event from the external input bus for External Input Resource 2.	

RES0, [15:13]	<i>res0</i>	Reserved.
SEL1, [12:8]	Selects an event from the external input bus for External Input Resource 1.	
RES0, [7:5]	<i>res0</i>	Reserved.
SEL0, [4:0]	Selects an event from the external input bus for External Input Resource 0.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEXTINSELR can be accessed through the external [debug interface](#), offset 120.

TRCIDR0, ID Register 0

The TRCIDR0 returns the tracing capabilities of the ETM trace unit.

Bit field descriptions

The TRCIDR0 is a 32-bit register.

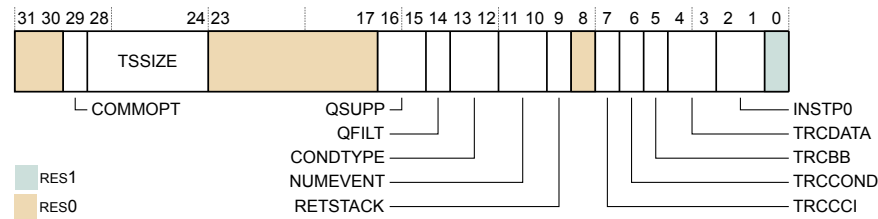


Figure 179: TRCIDR0 bit assignments

RES0, [31:30]	<i>res0</i>	Reserved.
COMMOPT, [29]	Indicates the meaning of the commit field in some packets:	
	1	Commit mode 1.
TSSIZE, [28:24]	Global timestamp size field:	
	0b01000	Implementation supports a maximum global timestamp of 64 bits.
RES0, [23:17]	<i>res0</i>	Reserved.
QSUPP, [16:15]	Indicates Q element support:	
	0b00	Q elements not supported.
QFILT, [14]	Indicates Q element filtering support:	

	0b0	Q element filtering not supported.
CONDTYPE, [13:12]	Indicates how conditional results are traced:	
	0b00	Conditional trace not supported.
NUMEVENT, [11:10]	Number of events supported in the trace, minus 1:	
	0b11	Four events supported.
RETSTACK, [9]	Return stack support:	
	1	Return stack implemented.
RES0, [8]	<i>res0</i>	Reserved.
TRCCCI, [7]	Support for cycle counting in the instruction trace:	
	1	Cycle counting in the instruction trace is implemented.
TRCCOND, [6]	Support for conditional instruction tracing:	
	0	Conditional instruction tracing is not supported.
TRCBB, [5]	Support for branch broadcast tracing:	
	1	Branch broadcast tracing is implemented.
TRCDATA, [4:3]	Conditional tracing field:	
	0b00	Tracing of data addresses and data values is not implemented.
INSTP0, [2:1]	P0 tracing support field:	
	0b00	Tracing of load and store instructions as P0 elements is not supported.
RES1, [0]	<i>res1</i>	Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR0 can be accessed through the external [debug interface](#), offset 1E0.

TRCIDR1, ID Register 1

The TRCIDR1 returns the base architecture of the trace unit.

Bit field descriptions

The TRCIDR1 is a 32-bit register.

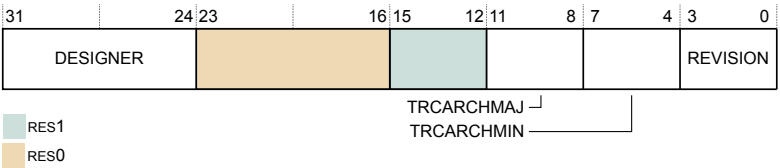


Figure 180: TRCIDR1 bit assignments

DESIGNER, [31:24]	Indicates which company designed the trace unit:	
	41	Arm.
RES0, [23:16]	<i>res0</i>	Reserved.
RES1, [15:12]	<i>res1</i>	Reserved.
TRCARCHMAJ, [11:8]	Major trace unit architecture version number:	
	0b0100	ETMv4.
TRCARCHMIN, [7:4]	Minor trace unit architecture version number:	
	2	ETMv4.2
REVISION, [3:0]	Trace unit implementation revision number:	
	3	ETM revision.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR1 can be accessed through the external [debug interface](#), offset 1E4.

TRCIDR2, ID Register 2

The TRCIDR2 returns the maximum size of six parameters in the trace unit.

Bit field descriptions

The parameters are:

- Cycle counter.
- Data value.
- Data address.
- VMID.
- Context ID.

- Instruction address.

The TRCIDR2 is a 32-bit register.

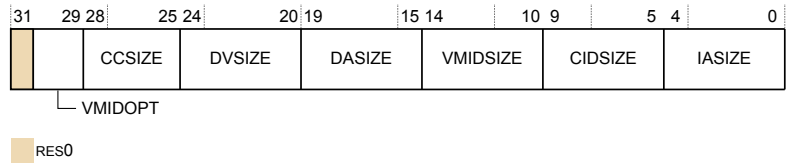


Figure 181: TRCIDR2 bit assignments

RES0, [31]	<i>res0</i>	Reserved.
VMIDOPT, [30:29]		Indicates the options for observing the Virtual context identifier:
	1	VMIDOPT is implemented.
CCSIZE, [28:25]		Size of the cycle counter in bits minus 12:
	0	The cycle counter is 12 bits in length.
DVSIZE, [24:20]		Data value size in bytes:
	00	Data value tracing is not implemented.
DASIZE, [19:15]		Data address size in bytes:
	00	Data address tracing is not implemented.
VMIDSIZE, [14:10]		Virtual Machine ID size:
	4	Maximum of 32-bit Virtual Machine ID size.
CIDSIZE, [9:5]		Context ID size in bytes:
	4	Maximum of 32-bit Context ID size.
IASIZE, [4:0]		Instruction address size in bytes:
	8	Maximum of 64-bit address size.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR2 can be accessed through the external [debug interface](#), offset 1E8.

TRCIDR3, ID Register 3

The TRCIDR3 indicates:

Bit field descriptions

- Whether TRCVICTLR is supported.
- The number of *cores* available for tracing.
- If an *Exception level* supports instruction tracing.
- The minimum threshold value for instruction trace cycle counting.
- Whether the synchronization period is fixed.
- Whether TRCSTALLCTLR is supported and if so whether it supports trace overflow prevention and supports stall control of the .

The TRCIDR3 is a 32-bit register.

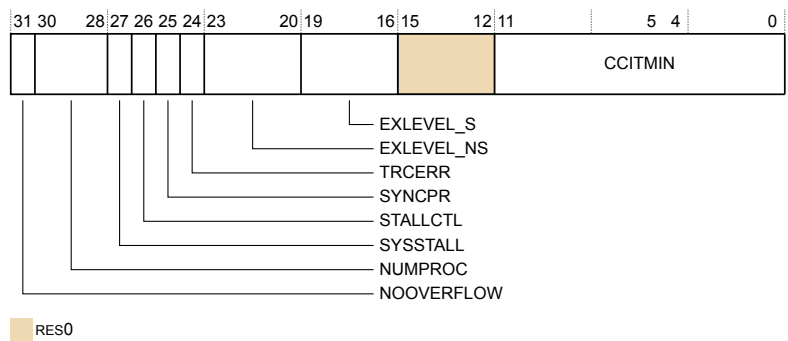


Figure 182: TRCIDR3 bit assignments

NOOVERFLOW, [31]	Indicates whether TRCSTALLCTLR.NOOVERFLOW is implemented:
	0 TRCSTALLCTLR.NOOVERFLOW is not implemented.
NUMPROC, [30:28]	Indicates the number of s available for tracing:
	0b000 The trace unit can trace one , <i>ETM</i> trace unit sharing not supported.
SYSSTALL, [27]	Indicates whether stall control is implemented:
	1 The system supports stall control.
STALLCTL, [26]	Indicates whether TRCSTALLCTLR is implemented:
	1 TRCSTALLCTLR is implemented.
	This field is used in conjunction with SYSSTALL.
SYNCPR, [25]	Indicates whether there is a fixed synchronization period:

	0	TRCSYNCPR is <i>read</i> -write so software can change the synchronization period.
TRCERR, [24]	Indicates whether TRCVICTLR.TRCERR is implemented:	
	1	TRCVICTLR.TRCERR is implemented.
EXLEVEL_NS, [23:20]	Each bit controls whether instruction tracing in Non-secure state is implemented for the corresponding :	
	0b0111	Instruction tracing is implemented for Non-secure EL0, EL1, and EL2 s.
EXLEVEL_S, [19:16]	Each bit controls whether instruction tracing in Secure state is implemented for the corresponding :	
	0b1011	Instruction tracing is implemented for Secure EL0, EL1, and EL3 s.
RES0, [15:12]	<i>res0</i>	Reserved.
CCITMIN, [11:0]	The minimum value that can be programmed in TRCCCCTLR.THRESHOLD:	
	004	Instruction trace cycle counting minimum threshold is 4.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR3 can be accessed through the external *debug interface*, offset 1EC.

TRCIDR4, ID Register 4

The TRCIDR4 indicates the resources available in the ETM trace unit.

Bit field descriptions

The TRCIDR4 is a 32-bit register.

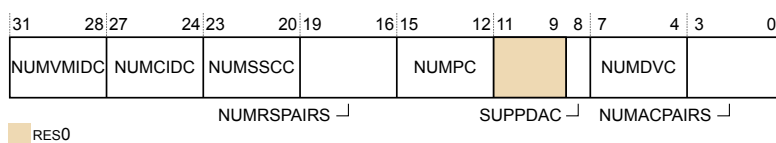


Figure 183: TRCIDR4 bit assignments

NUMVMIDC, [31:28]	Indicates the number of VMID comparators available for tracing:
1	One VMID comparator is available.
NUMCIDC, [27:24]	Indicates the number of CID comparators available for tracing:
1	One Context ID comparator is available.
NUMSSCC, [23:20]	Indicates the number of single-shot comparator controls available for tracing:
1	One single-shot comparator control is available.
NUMRSPAIRS, [19:16]	Indicates the number of resource selection pairs available for tracing:
7	Eight resource selection pairs are available.
NUMPC, [15:12]	Indicates the number of <i>core</i> comparator inputs available for tracing:
0	Core comparator inputs are not implemented.
RES0, [11:9]	<i>res0</i> Reserved.
SUPPDAC, [8]	Indicates whether the implementation supports data address comparisons: This value is:
0	Data address comparisons are not implemented.
NUMDVC, [7:4]	Indicates the number of data value comparators available for tracing:
0	Data value comparators not implemented.
NUMACPAIRS, [3:0]	Indicates the number of address comparator pairs available for tracing:
4	Four address comparator pairs are implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR4 can be accessed through the external *debug interface*, offset 1F0.

TRCIDR5, ID Register 5

The TRCIDR5 returns how many resources the trace unit supports.

Bit field descriptions

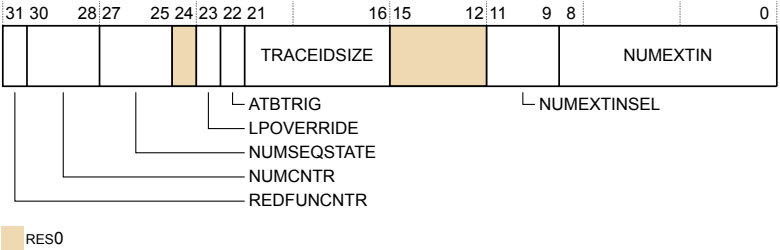


Figure 184: TRCIDR5 bit assignments

REDFUNCNTR, [31]

Reduced Function Counter implemented:

0 Reduced Function Counter not implemented.

NUMCNTR, [30:28]

Number of counters implemented:

0b010 Two counters implemented.

NUMSEQSTATE, [27:25]

Number of sequencer states implemented:

0b100 Four sequencer states implemented.

RES0, [24]

res0 Reserved.

LPOVERRIDE, [23]

Low-power state override support:

1 Low-power state override support implemented.

ATBTRIG, [22]

ATB trigger support:

1 ATB trigger support implemented.

TRACEIDSIZE, [21:16]

Number of bits of trace ID:

07 Seven-bit trace ID implemented.

RES0, [15:12]

res0 Reserved.

NUMEXTINSEL, [11:9]

Number of external input selectors implemented:

	0b100	Four external input selectors implemented.
NUMEXTIN, [8:0]	Number of external inputs implemented:	
	AD	32 external inputs implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR5 can be accessed through the external [debug interface](#), offset 1F4.

TRCIDR8, ID Register 8

The TRCIDR8 returns the maximum speculation depth of the instruction trace stream.

Bit field descriptions

The TRCIDR8 is a 32-bit register.

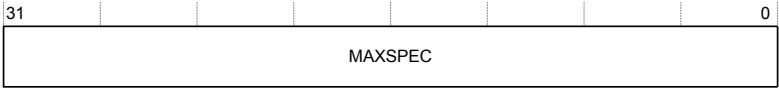


Figure 185: TRCIDR8 bit assignments

MAXSPEC, [31:0]	The maximum number of P0 elements in the trace stream that can be speculative at any time.	
	0	Maximum speculation depth of the instruction trace stream.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR8 can be accessed through the external [debug interface](#), offset 180.

TRCIDR9, ID Register 9

The TRCIDR9 returns the number of P0 right-hand keys that the trace unit can use.

Bit field descriptions

The TRCIDR9 is a 32-bit register.

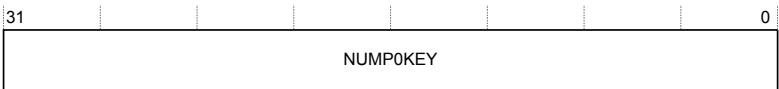


Figure 186: TRCIDR9 bit assignments

NUMPOKEY, [31:0]	The number of P0 right-hand keys that the trace unit can use.	
------------------	---	--

0 Number of P0 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR9 can be accessed through the external [debug interface](#), offset 184.

TRCIDR10, ID Register 10

The TRCIDR10 returns the number of P1 right-hand keys that the trace unit can use.

Bit field descriptions

The TRCIDR10 is a 32-bit register.

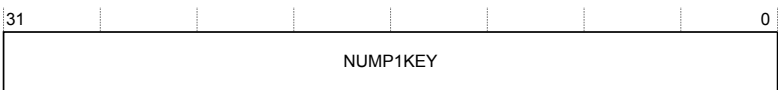


Figure 187: TRCIDR10 bit assignments

NUMP1KEY, [31:0] The number of P1 right-hand keys that the trace unit can use.

0 Number of P1 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR10 can be accessed through the external [debug interface](#), offset 188.

TRCIDR11, ID Register 11

The TRCIDR11 returns the number of special P1 right-hand keys that the trace unit can use.

Bit field descriptions

The TRCIDR11 is a 32-bit register.

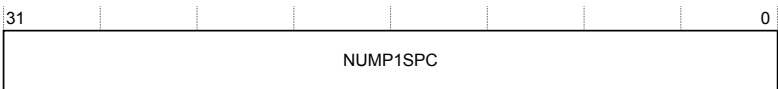


Figure 188: TRCIDR11 bit assignments

NUMP1SPC, [31:0] The number of special P1 right-hand keys that the trace unit can use.

0 Number of special P1 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR11 can be accessed through the external [debug interface](#), offset 18C.

TRCIDR12, ID Register 12

The TRCIDR12 returns the number of conditional instruction right-hand keys that the trace unit can use.

Bit field descriptions

The TRCIDR10 is a 32-bit register.

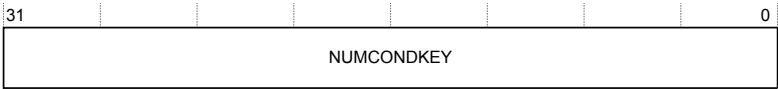


Figure 189: TRCIDR12 bit assignments

NUMCONDKEY, [31:0]

The number of conditional instruction right-hand keys that the trace unit can use, including normal and special keys.

0

Number of conditional instruction right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR12 can be accessed through the external [debug interface](#), offset 190.

TRCIDR13, ID Register 13

The TRCIDR13 returns the number of special conditional instruction right-hand keys that the trace unit can use.

Bit field descriptions

The TRCIDR11 is a 32-bit register.

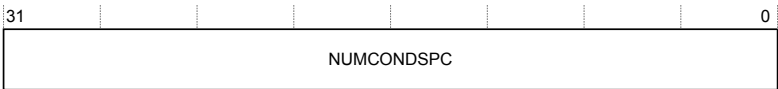


Figure 190: TRCIDR13 bit assignments

NUMCONDSPC, [31:0]

The number of special conditional instruction right-hand keys that the trace unit can use, including normal and special keys.

0

Number of special conditional instruction right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR13 can be accessed through the external [debug interface](#), offset 194.

TRCIMSPEC0, *Implementation Specific Register 0*

The TRCIMSPEC0 shows the presence of any *implementation specific* features, and enables any features that are provided.

Bit field descriptions

The TRCIMSPEC0 is a 32-bit register.

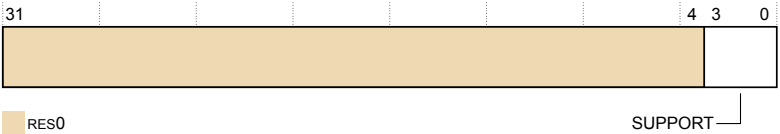


Figure 191: TRCIMSPEC0 bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
SUPPORT, [3:0]	0	No <i>implementation specific</i> extensions are supported.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIMSPEC0 can be accessed through the external [debug interface](#), offset 1C0.

TRCITATBIDR, *Integration ATB Identification Register*

The TRCITATBIDR sets the state of output pins, mentioned in the bit descriptions in this section.

Bit field descriptions

The TRCITATBIDR is a 32-bit register.

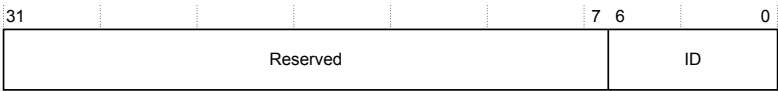


Figure 192: TRCITATBIDR bit assignments

[31:7]	Reserved. Read undefined.
ID, [6:0]	<p>Drives the ATIDMn[6:0] output pins.</p> <p>When a bit is set to 0, the corresponding output pin is LOW.</p> <p>When a bit is set to 1, the corresponding output pin is HIGH.</p> <p>The TRCITATBIDR bit values correspond to the physical state of the output pins.</p>

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITATBIDR can be accessed through the external [debug interface](#), offset EE4.

TRCITCTRL, Integration Mode Control Register

The TRCITCTRL enables topology detection or integration testing, by putting the ETM trace unit into integration mode.

Bit field descriptions

The TRCITCTRL is a 32-bit register.

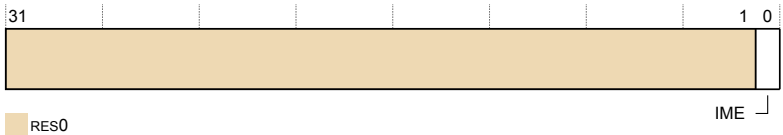


Figure 193: TRCITCTRL bit assignments

RES0, [31:1]	<i>res0</i>	Reserved.
IME, [0]	Integration mode enable bit. The possible values are:	
	0	The trace unit is not in integration mode.
	1	The trace unit is in integration mode. This mode enables: <ul style="list-style-type: none"> A debug agent to perform topology detection. SoC test software to perform integration testing.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITCTRL can be accessed through the external [debug interface](#), offset F00.

TRCITIATBINR, Integration Instruction ATB In Register

The TRCITIATBINR reads the state of the input pins described in this section.

Bit field descriptions

The TRCITIATBINR is a 32-bit register.

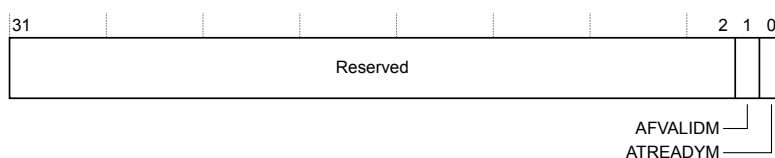


Figure 194: TRCITIATBINR bit assignments

For all non-reserved bits:

- When an input pin is LOW, the corresponding register bit is 0.
- When an input pin is HIGH, the corresponding register bit is 1.
- The TRCITIATBINR bit values always correspond to the physical state of the input pins.

[31:2]	Reserved. Read undefined.
AFVALIDM, [1]	Returns the value of the AFVALIDMn input pin.
ATREADYM, [0]	Returns the value of the ATREADYMn input pin.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIATBINR can be accessed through the external [debug interface](#), offset EF4.

TRCITIATBOUTR, Integration Instruction ATB Out Register

The TRCITIATBOUTR sets the state of the output pins mentioned in the bit descriptions in this section.

Bit field descriptions

The TRCITIATBOUTR is a 32-bit register.

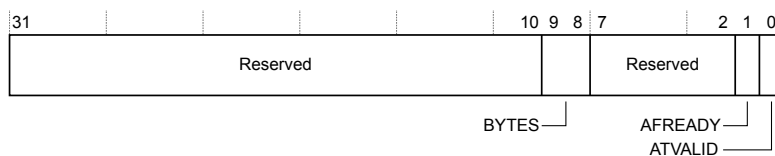


Figure 195: TRCITIATBOUTR bit assignments

For all non-reserved bits:

- When a bit is set to 0, the corresponding output pin is LOW.
- When a bit is set to 1, the corresponding output pin is HIGH.
- The TRCITIATBOUTR bit values always correspond to the physical state of the output pins.

[31:10]	Reserved. Read undefined.
BYTES, [9:8]	Drives the ATBYTESMn[1:0] output pins.
[7:2]	Reserved. Read undefined.
AFREADY, [1]	Drives the AFREADYMn output pin.
ATVALID, [0]	Drives the ATVALIDMn output pin.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIATBOUTR can be accessed through the external [debug interface](#), offset EFC.

TRCITIDATAR, Integration Instruction ATB Data Register

The TRCITIDATAR sets the state of the ATDATAMn output pins shown in the TRCITIDATAR bit assignments table.

Bit field descriptions

The TRCITIDATAR is a 32-bit register.

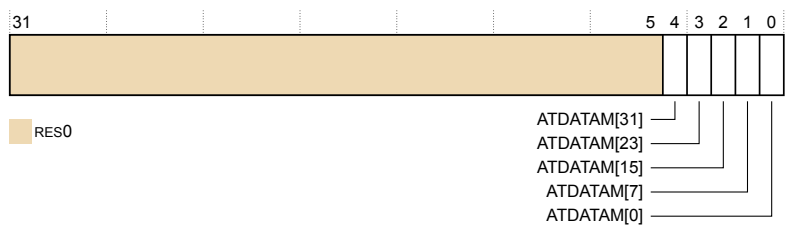


Figure 196: TRCITIDATAR bit assignments

RES0, [31:5]	<i>res0</i>	Reserved.
ATDATAM[31], [4]	Drives the ATDATAM[31] output. ⁴	
ATDATAM[23], [3]	Drives the ATDATAM[23] output. ⁴	
ATDATAM[15], [2]	Drives the ATDATAM[15] output. ⁴	
ATDATAM[7], [1]	Drives the ATDATAM[7] output. ⁴	
ATDATAM[0], [0]	Drives the ATDATAM[0] output. ⁴	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIDATAR can be accessed through the external [debug interface](#), offset EEC.

TRCLAR, Software Lock Access Register

The TRCLAR controls access to registers using the memory-mapped interface, when PADDRDBG31 is LOW.

Bit field descriptions

When the software lock is set, write accesses using the memory-mapped interface to all [ETM](#) trace unit registers are *ignored*.

When the software lock is set, [read](#) accesses of TRCPDSR do not change the TRCPDSR.STICKYPD bit. Read accesses of all other registers are not affected.

The TRCLAR is a 32-bit register.

⁴ When a bit is set to 0, the corresponding output pin is LOW. When a bit is set to 1, the corresponding output pin is HIGH. The TRCITIDATAR bit values correspond to the physical state of the output pins.

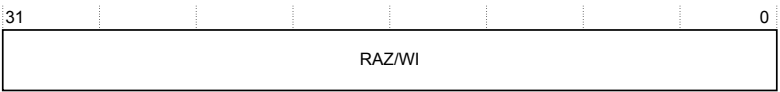


Figure 197: TRCLAR bit assignments

RAZ/WI, [31:0] Read-As-Zero, write ignore.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCLAR can be accessed through the external [debug interface](#), offset FB0.

TRCLSR, Software Lock Status Register

The TRCLSR determines whether the software lock is implemented, and indicates the current status of the software lock.

Bit field descriptions

The TRCLSR is a 32-bit register.



Figure 198: TRCLSR bit assignments

RAZ/WI, [31:0] Read-As-Zero, write ignore.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCLSR can be accessed through the external [debug interface](#), offset FB4.

TRCCNTVRn, Counter Value Registers 0-1

The TRCCNTVRn contains the current counter value.

Bit field descriptions

The TRCCNTVRn is a 32-bit register.

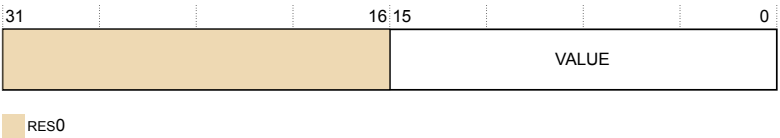


Figure 199: TRCCNTVRn bit assignments

RES0, [31:16] *res0* Reserved.

VALUE, [15:0] Contains the current counter value.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTVRn registers can be accessed through the external [debug interface](#), offsets:

TRCCNTVR0 160.

TRCCNTVR1 164.

TRCOSLAR, OS Lock Access Register

The TRCOSLAR sets and clears the OS Lock, to lock out external debugger accesses to the ETM trace unit registers.

Bit field descriptions

The TRCOSLAR is a 32-bit register.

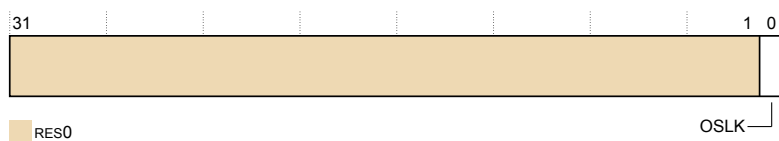


Figure 200: TRCOSLAR bit assignments

RES0, [31:1]	<i>res0</i>	Reserved.
OSLK, [0]	OS Lock key value:	
	0	Unlock the OS Lock.
	1	Lock the OS Lock.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCOSLAR can be accessed through the external [debug interface](#), offset 300.

TRCOSLSR, OS Lock Status Register

The TRCOSLSR returns the status of the OS Lock.

Bit field descriptions

The TRCOSLSR is a 32-bit register.

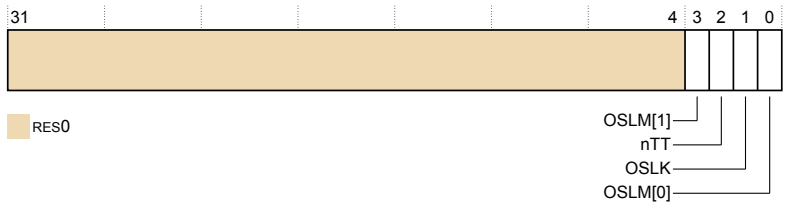


Figure 201: TRCOSLSR bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
OSLM[1], [3]		OS Lock model [1] bit. This bit is combined with OSLM[0] to form a two-bit field that indicates the OS Lock model is implemented. The value of this field is always 0b10, indicating that the OS Lock is implemented.
nTT, [2]		This bit is <i>RAZ</i> , that indicates that software must perform a 32-bit write to update the TRCOSLAR.
OSLK, [1]		OS Lock status bit: 0 OS Lock is unlocked. 1 OS Lock is locked.
OSLM[0], [0]		OS Lock model [0] bit. This bit is combined with OSLM[1] to form a two-bit field that indicates the OS Lock model is implemented. The value of this field is always 0b10, indicating that the OS Lock is implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCOSLSR can be accessed through the external *debug interface*, offset 304.

TRCPDCR, Power Down Control Register

The TRCPDCR request to the system power controller to keep the ETM trace unit powered up.

Bit field descriptions

The TRCPDCR is a 32-bit register.

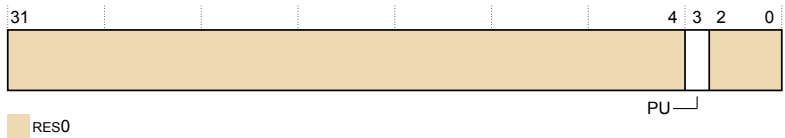


Figure 202: TRCPDCR bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
PU, [3]	Powerup request, to request that power to the <i>ETM</i> trace unit and access to the trace registers is maintained:	
	0	Power not requested.
	1	Power requested.
	This bit is reset to 0 on a trace unit reset.	
RES0, [2:0]	<i>res0</i>	Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPDCR can be accessed through the external *debug interface*, offset 310.

TRCPDSR, Power Down Status Register

The TRCPDSR indicates the power down status of the ETM trace unit.

Bit field descriptions

The TRCPDSR is a 32-bit register.

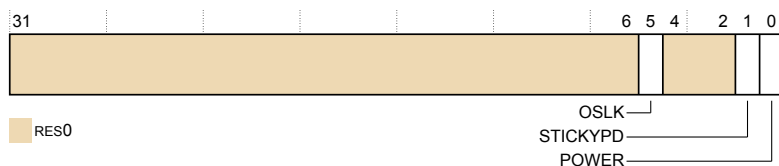


Figure 203: TRCPDSR bit assignments

RES0, [31:6]	<i>res0</i>	Reserved.
OSLK, [5]	OS lock status.	
	0	The OS Lock is unlocked.
	1	The OS Lock is locked.
RES0, [4:2]	<i>res0</i>	Reserved.
STICKYPD, [1]	Sticky power down state.	
	0	Trace register power has not been removed since the TRCPDSR was last <i>read</i> .
	1	Trace register power has been removed since the TRCPDSR was last .

This bit is set to 1 when power to the [ETM](#) trace unit registers is removed, to indicate that programming state has been lost. It is cleared after a of the TRCPDSR.

POWER, [0]

Indicates the trace unit is powered:

- 0

trace unit is not powered.
The trace registers are not accessible and they all return an error response.
- 1

trace unit is powered. All registers are accessible.

If a system implementation allows the trace unit to be powered off independently of the debug power domain, the system must handle accesses to the trace unit appropriately.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPDSR can be accessed through the external [debug interface](#), offset 314.

TRCPIDR0, ETM Peripheral Identification Register 0

The TRCPIDR0 provides information to identify a trace component.

Bit field descriptions

The TRCPIDR0 is a 32-bit register.

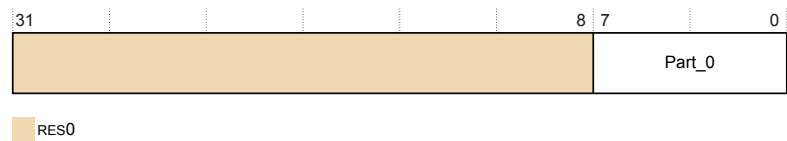


Figure 204: TRCPIDR0 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Part_0, [7:0]	0B	Least significant byte of the ETM trace unit part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR0 can be accessed through the external [debug interface](#), offset FE0.

TRCPIDR1, ETM Peripheral Identification Register 1

The TRCPIDR1 provides information to identify a trace component.

Bit field descriptions

The TRCPIDR1 is a 32-bit register.

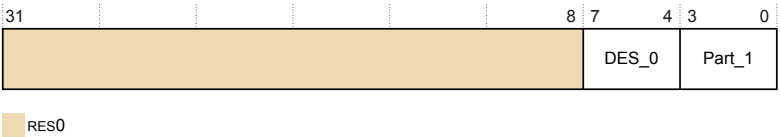


Figure 205: TRCPIDR1 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
DES_0, [7:4]	B	Arm Limited. This is bits[3:0] of JEP106 ID code.
Part_1, [3:0]	D	Most significant four bits of the <i>ETM</i> trace unit part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR1 can be accessed through the external [debug interface](#), offset FE4.

TRCPIDR2, ETM Peripheral Identification Register 2

The TRCPIDR2 provides information to identify a trace component.

Bit field descriptions

The TRCPIDR2 is a 32-bit register.

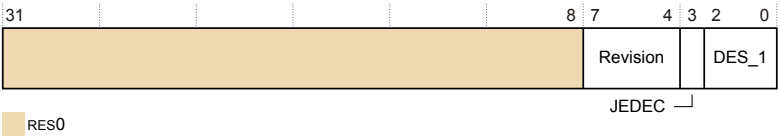


Figure 206: TRCPIDR2 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Revision, [7:4]	0	r0p0.

JEDEC, [3]	0b1	<i>res1</i> . Indicates a JEP106 identity code is used.
DES_1, [2:0]	0b011	Arm Limited. This is bits[6:4] of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR2 can be accessed through the external [debug interface](#), offset FE8.

TRCPIDR3, ETM Peripheral Identification Register 3

The TRCPIDR3 provides information to identify a trace component.

Bit field descriptions

The TRCPIDR3 is a 32-bit register.

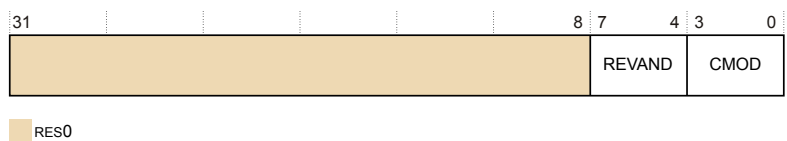


Figure 207: TRCPIDR3 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
REVAND, [7:4]	0	Part minor revision.
CMOD, [3:0]	0	Not customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR3 can be accessed through the external [debug interface](#), offset FEC.

TRCPIDR4, ETM Peripheral Identification Register 4

The TRCPIDR4 provides information to identify a trace component.

Bit field descriptions

The TRCPIDR4 is a 32-bit register.

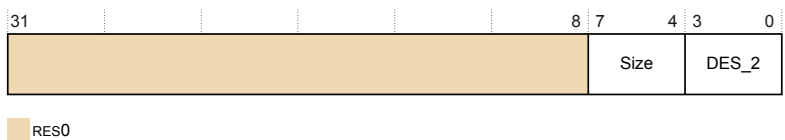


Figure 208: TRCPIDR4 bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
Size, [7:4]	0	Size of the component. Log2 the number of 4KB pages from the start of the component to the end of the component ID registers.
DES_2, [3:0]	4	Arm Limited. This is bits[3:0] of the JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR4 can be accessed through the external [debug interface](#), offset FD0.

TRCPIDRn, ETM Peripheral Identification Registers 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are *res0*.

TRCPRGCTLR, Programming Control Register

The TRCPRGCTLR enables the ETM trace unit.

Bit field descriptions

The TRCPRGCTLR is a 32-bit register.

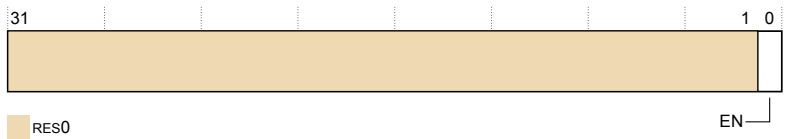


Figure 209: TRCPRGCTLR bit assignments

RES0, [31:1]	<i>res0</i>	Reserved.
EN, [0]	Trace program enable:	

0	The <i>ETM</i> trace unit interface in the <i>core</i> is disabled, and clocks are enabled only when necessary to process <i>APB</i> accesses, or drain any already generated trace. This is the reset value.
1	The trace unit interface in the is enabled, and clocks are enabled. <i>Writes</i> to most trace registers are <i>ignored</i> .

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8*, for *Armv8-A architecture profile*.

The TRCPRGCTLR can be accessed through the external *debug interface*, offset 004.

TRCRSCTLRn, Resource Selection Control Registers 2-16

The TRCRSCTLRn controls the trace resources. There are eight resource pairs, the first pair is predefined as {0,1,pair=0} and having reserved select registers. This leaves seven pairs to be implemented as programmable selectors.

Bit field descriptions

The TRCRSCTLRn is a 32-bit register.

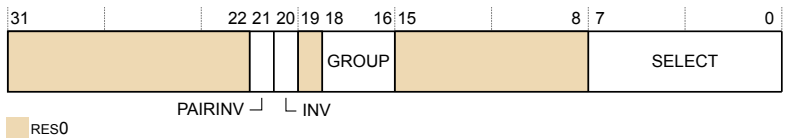


Figure 210: TRCRSCTLRn bit assignments

RES0, [31:22]	<i>res0</i>	Reserved.
PAIRINV, [21]		Inverts the result of a combined pair of resources. This bit is implemented only on the lower register for a pair of resource selectors.
INV, [20]		Inverts the selected resources:
	0	Resource is not inverted.
	1	Resource is inverted.
RES0, [19]	<i>res0</i>	Reserved.
GROUP, [18:16]		Selects a group of resources. See the <i>Arm® ETM Architecture Specification, v4</i> for more information.

RES0, [15:8]	<i>res0</i>	Reserved.
SELECT, [7:0]	Selects one or more resources from the required group. One bit is provided for each resource from the group.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCRSCTLRn can be accessed through the external [debug interface](#), offset 208–023C.

TRCSEQEVRn, Sequencer State Transition Control Registers 0-2

The TRCSEQEVRn defines the sequencer transitions that progress to the next state or backwards to the previous state. The ETM trace unit implements a sequencer state machine with up to four states.

Bit field descriptions

The TRCSEQEVRn is a 32-bit register.

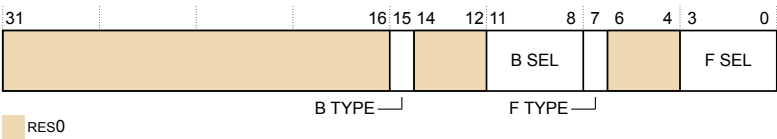


Figure 211: TRCSEQEVRn bit assignments

RES0, [31:16]	<i>res0</i>	Reserved.
B TYPE, [15]	Selects the resource type to move backwards to this state from the next state:	
	0	Single selected resource.
	1	Boolean combined resource pair.
RES0, [14:12]	<i>res0</i>	Reserved.
B SEL, [11:8]	Selects the resource number, based on the value of B TYPE:	
	When B TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].	
	When B TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].	
F TYPE, [7]	Selects the resource type to move forwards from this state to the next state:	
	0	Single selected resource.
	1	Boolean combined resource pair.

RES0, [6:4]	<i>res0</i>	Reserved.
F SEL, [3:0]	<p>Selects the resource number, based on the value of F TYPE:</p> <p>When F TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].</p> <p>When F TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].</p>	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQEVRn registers can be accessed through the external [debug interface](#), offsets:

TRCSEQEVR0	100.
TRCSEQEVR1	104.
TRCSEQEVR2	108.

TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR resets the sequencer to state 0.

Bit field descriptions

The TRCSEQRSTEVR is a 32-bit register

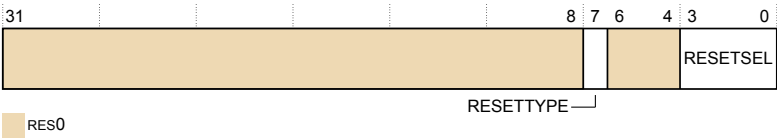


Figure 212: TRCSEQRSTEVR bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
RESETTYPE, [7]	<p>Selects the resource type to move back to state 0:</p> <p>0 Single selected resource.</p> <p>1 Boolean combined resource pair.</p>	
RES0, [6:4]	<i>res0</i>	Reserved.
RESETSSEL, [3:0]	<p>Selects the resource number, based on the value of RESETTYPE:</p> <p>When RESETTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].</p> <p>When RESETTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].</p>	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQRSTEV can be accessed through the external [debug interface](#), offset 118.

TRCSEQSTR, Sequencer State Register

The TRCSEQSTR holds the value of the current state of the sequencer.

Bit field descriptions

The TRCSEQSTR is a 32-bit register

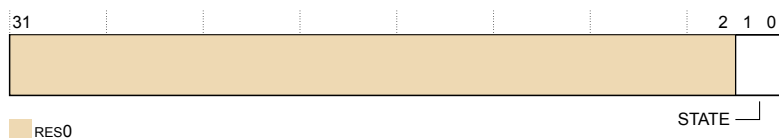


Figure 213: TRCSEQSTR bit assignments

RES0, [31:2]	<i>res0</i>	Reserved.
STATE, [1:0]	Current sequencer state:	
	0b00	State 0.
	0b01	State 1.
	0b10	State 2.
	0b11	State 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQSTR can be accessed through the external [debug interface](#), offset 11C.

TRCSSCCR0, Single-Shot Comparator Control Register 0

The TRCSSCCR0 controls the single-shot comparator.

Bit field descriptions

The TRCSSCSR0 is a 32-bit register

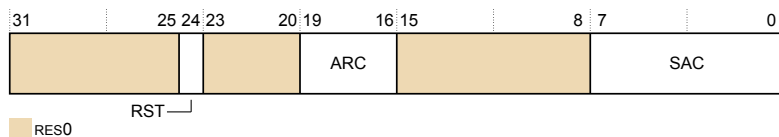


Figure 214: TRCSSCCR0 bit assignments

RES0, [31:25]	<i>res0</i>	Reserved.
---------------	-------------	-----------

RST, [24]	Enables the single-shot comparator resource to be reset when it occurs, to enable another comparator match to be detected:	
	1	Reset enabled. Multiple matches can occur.
RES0, [23:20]	<i>res0</i>	Reserved.
ARC, [19:16]	Selects one or more address range comparators for single-shot control. One bit is provided for each implemented address range comparator.	
RES0, [15:8]	<i>res0</i>	Reserved.
SAC, [7:0]	Selects one or more single address comparators for single-shot control. One bit is provided for each implemented single address comparator.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSSCCR0 can be accessed through the external [debug interface](#), offset 280.

TRCSSCSR0, Single-Shot Comparator Status Register 0

The TRCSSCSR0 indicates the status of the single-shot comparator. TRCSSCSR0 is sensitive to instruction addresses.

Bit field descriptions

The TRCSSCSR0 is a 32-bit register

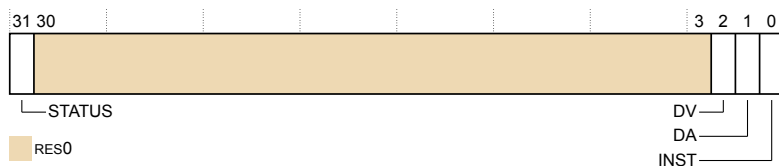


Figure 215: TRCSSCSR0 bit assignments

STATUS, [31]	Single-shot status. This indicates whether any of the selected comparators have matched:	
	0	Match has not occurred.
	1	Match has occurred at least once.
When programming the ETM trace unit, if TRCSSCCRn.RST is b0, the STATUS bit must		

be explicitly written to 0 to enable this single-shot comparator control.

RES0, [30:3]	res0	Reserved.
DV, [2]	Data value comparator support:	
	0	Single-shot data value comparisons not supported.
DA, [1]	Data address comparator support:	
	0	Single-shot data address comparisons not supported.
INST, [0]	Instruction address comparator support:	
	1	Single-shot instruction address comparisons supported.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSSCSR0 can be accessed through the external [debug interface](#), offset 2A0.

TRCSTALLCTLR, Stall Control Register

The TRCSTALLCTLR enables the ETM trace unit to stall the Cortex®-A76 core if the ETM trace unit FIFO overflows.

Bit field descriptions

The TRCSTALLCTLR is a 32-bit register.

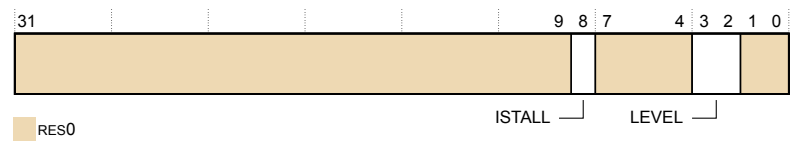


Figure 216: TRCSTALLCTLR bit assignments

RES0, [31:9]	res0	Reserved.
ISTALL, [8]	Instruction stall bit. Controls if the trace unit can stall the core when the instruction trace buffer space is less than LEVEL:	
	0	The trace unit does not stall the .

	1	The trace unit can stall the .
RES0, [7:4]	res0	Reserved.
LEVEL, [3:2]	Threshold level field. The field can support 4 monotonic levels from 0b00 to 0b11, where:	
	0b00	Zero invasion. This setting has a greater risk of an ETM trace unit FIFO overflow.
	0b11	Maximum invasion occurs but there is less risk of a FIFO overflow.
RES0, [1:0]	res0	Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSTALLCTLR can be accessed through the external [debug interface](#), offset 02C.

TRCSTATR, Status Register

The TRCSTATR indicates the ETM trace unit status.

Bit field descriptions

The TRCSTATR is a 32-bit register.

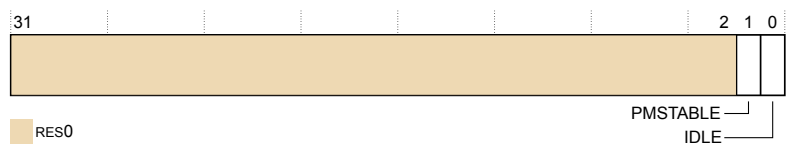


Figure 217: TRCSTATR bit assignments

RES0, [31:2]	res0	Reserved.
PMSTABLE, [1]	Indicates whether the ETM trace unit registers are stable and can be read :	
	0	The programmers model is not stable.
	1	The programmers model is stable.
IDLE, [0]	Idle status:	
	0	The trace unit is not idle.

1

The trace unit is idle.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSTATR can be accessed through the external [debug interface](#), offset 00C.

TRCSYNCP, Synchronization Period Register

The TRCSYNCP controls how often periodic trace synchronization requests occur.

Bit field descriptions

The TRCSYNCP is a 32-bit register.

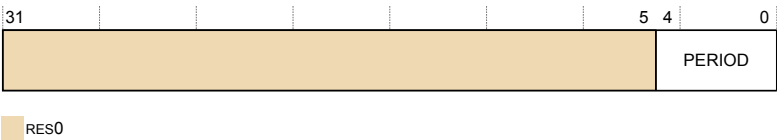


Figure 218: TRCSYNCP bit assignments

RES0, [31:5]

res0

Reserved.

PERIOD, [4:0]

Defines the number of bytes of trace between synchronization requests as a total of the number of bytes generated by both the instruction and data streams. The number of bytes is 2^N where N is the value of this field:

- A value of zero disables these periodic synchronization requests, but does not disable other synchronization requests.
- The minimum value that can be programmed, other than zero, is 8, providing a minimum synchronization period of 256 bytes.
- The maximum value is 20, providing a maximum synchronization period of 2^{20} bytes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSYNCP can be accessed through the external [debug interface](#), offset 034.

TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR sets the trace ID for instruction trace.

Bit field descriptions

The TRCTRACEIDR is a 32-bit register.

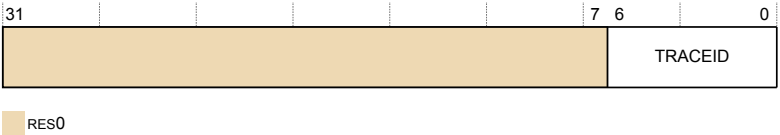


Figure 219: TRCTraceIDR bit Assignments

RES0, [31:7]	<i>res0</i>	Reserved.
TRACEID, [6:0]	Trace ID value. When only instruction tracing is enabled, this provides the trace ID.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCTraceIDR can be accessed through the external [debug interface](#), offset 040.

TRCTSCTLR, Global Timestamp Control Register

The TRCTSCTLR controls the insertion of global timestamps in the trace streams. When the selected event is triggered, the trace unit inserts a global timestamp into the trace streams. The event is selected from one of the Resource Selectors.

Bit field descriptions

The TRCTSCTLR is a 32-bit register.

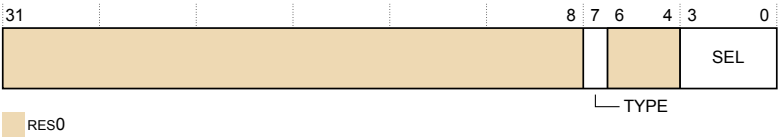


Figure 220: TRCTSCTLR bit assignments

RES0, [31:8]	<i>res0</i>	Reserved.
TYPE, [7]	Single or combined resource selector.	
RES0, [6:4]	<i>res0</i>	Reserved.
SEL, [3:1]	Identifies the resource selector to use.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCTSCTLR can be accessed through the external [debug interface](#), offset 030.

TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR controls instruction trace filtering.

Bit field descriptions

The TRCVICTLR is a 32-bit register.

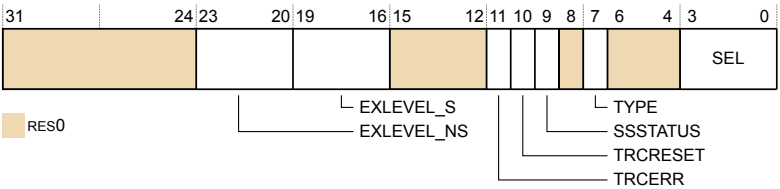


Figure 221: TRCVICTLR bit assignments

RES0, [31:24]

res0

Reserved.

EXLEVEL_NS, [23:20]

In Non-secure state, each bit controls whether instruction tracing is enabled for the corresponding *Exception level*:

0

Trace unit generates instruction trace, in Non-secure state, for *n*.

1

Trace unit does not generate instruction trace, in Non-secure state, for *n*.

The *s* are:

Bit[20]

0.

Bit[21]

1.

Bit[22]

2.

Bit[23]

RAZ/WI. Instruction tracing is not implemented for 3.

EXLEVEL_S, [19:16]

In Secure state, each bit controls whether instruction tracing is enabled for the corresponding :

0

Trace unit generates instruction trace, in Secure state, for *n*.

1

Trace unit does not generate instruction trace, in Secure state, for *n*.

The *s* are:

Bit[16]

0.

	Bit[17]	1.
	Bit[18]	RAZ/WI. Instruction tracing is not implemented for Exception level 2.
	Bit[19]	3.
RES0, [15:12]	<i>res0</i>	Reserved.
TRCERR, [11]	Selects whether a system error <i>exception</i> must always be traced:	
	0	System error is traced only if the instruction or immediately before the system error is traced.
	1	System error is always traced regardless of the value of ViewInst.
TRCRESET, [10]	Selects whether a reset must always be traced:	
	0	Reset is traced only if the instruction or immediately before the reset is traced.
	1	Reset is always traced regardless of the value of ViewInst.
SSSTATUS, [9]	Indicates the current status of the start/stop logic:	
	0	Start/stop logic is in the stopped state.
	1	Start/stop logic is in the started state.
RES0, [8]	<i>res0</i>	Reserved.
TYPE, [7]	Selects the resource type for the viewinst event:	
	0	Single selected resource.
	1	Boolean combined resource pair.
RES0, [6:4]	<i>res0</i>	Reserved.
SEL, [3:0]	Selects the resource number to use for the viewinst event, based on the value of TYPE: When TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].	

When TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVICTLR can be accessed through the external [debug interface](#), offset 080.

TRCVIIECTLR, ViewInst Include-Exclude Control Register

The TRCVIIECTLR defines the address range comparators that control the ViewInst Include/Exclude control.

Bit field descriptions

The TRCVIIECTLR is a 32-bit register.

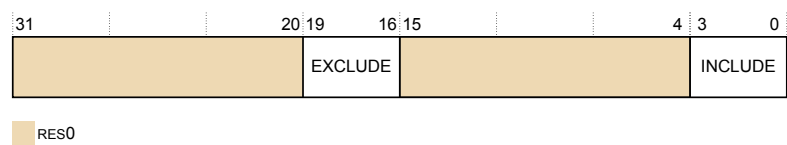


Figure 222: TRCVIIECTLR bit assignments

RES0, [31:20]	<i>res0</i>	Reserved.
EXCLUDE, [19:16]	Defines the address range comparators for ViewInst exclude control. One bit is provided for each implemented Address Range Comparator.	
RES0, [15:4]	<i>res0</i>	Reserved.
INCLUDE, [3:0]	Defines the address range comparators for ViewInst include control. Selecting no include comparators indicates that all instructions must be included. The exclude control indicates which ranges must be excluded. One bit is provided for each implemented Address Range Comparator.	

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVIIECTLR can be accessed through the external [debug interface](#), offset 084.

TRCVISSCTLR, ViewInst Start-Stop Control Register

The TRCVISSCTLR defines the single address comparators that control the ViewInst Start/Stop logic.

Bit field descriptions

The TRCVISSCTLR is a 32-bit register.

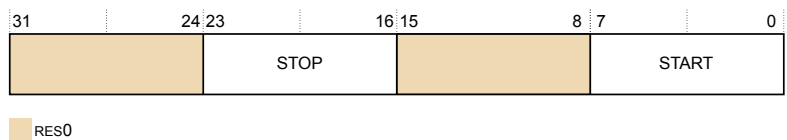


Figure 223: TRCVISSCTLR bit assignments

RES0, [31:24]	<i>res0</i>	Reserved.
STOP, [23:16]		Defines the single address comparators to stop trace with the ViewInst Start/Stop control. One bit is provided for each implemented single address comparator.
RES0, [15:8]	<i>res0</i>	Reserved.
START, [7:0]		Defines the single address comparators to start trace with the ViewInst Start/Stop control. One bit is provided for each implemented single address comparator.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVISSCTLR can be accessed through the external [debug interface](#), offset 088.

TRCVMIDCVR0, VMID Comparator Value Register 0

The TRCVMIDCVR0 contains a VMID value.

Bit field descriptions

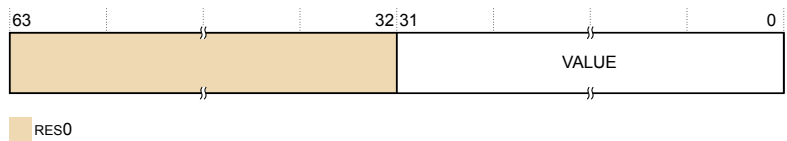


Figure 224: TRCVMIDCVR0 bit assignments

RES0, [63:32]	<i>res0</i>	Reserved.
VALUE, [31:0]		The VMID value.

The TRCVMIDCVR0 can be accessed through the internal memory-mapped interface and the external [debug interface](#), offset 640.

Usage constraints	Accepts writes only when the trace unit is disabled.
Configurations	Available in all configurations.

TRCVMIDCCTLR0, Virtual context identifier Comparator Control Register 0

The TRCVMIDCCTLR0 contains the Virtual machine identifier mask value for the TRCVMIDCVR0 register.

Bit field descriptions

The TRCVMIDCCTLR0 is a 32-bit register.

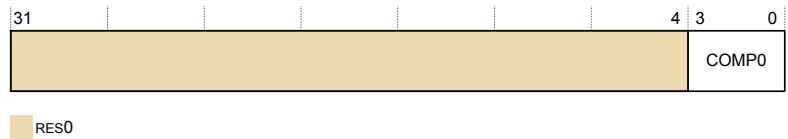


Figure 225: TRCVMIDCCTLR0 bit assignments

RES0, [31:4]	<i>res0</i>	Reserved.
COMP0, [3:0]	Controls the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0. When a bit is:	
	0	The trace unit includes the relevant byte in TRCVMIDCVR0 when it performs the Virtual context ID comparison.
	1	The trace unit ignores the relevant byte in TRCVMIDCVR0 when it performs the Virtual context ID comparison.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVMIDCCTLR0 can be accessed through the external [debug interface](#), offset 688.

Appendix

Appendices

Topics:

- [Cortex-A76 Core AArch32 UNPREDICTABLE behaviors](#)
- [Revisions](#)

This part describes the appendices of the Cortex[®]-A76 core.

Appendix

A

Cortex[®]-A76 Core AArch32 UNPREDICTABLE behaviors

Topics:

- [Use of R15 by Instruction](#)
- [Load/Store accesses crossing page boundaries](#)
- [Armv8 Debug UNPREDICTABLE behaviors](#)
- [Other UNPREDICTABLE behaviors](#)

This appendix describes the cases in which the Cortex[®]-A76 core implementation diverges from the preferred behavior described in Armv8 AArch32 *UNPREDICTABLE* behaviors.

Use of R15 by Instruction

If the use of R15 as a base register for a load or store is *unpredictable*, the value used by the load or store using R15 as a base register is the *Program Counter* (PC) with its usual offset and, in the case of T32 instructions, with the forced word alignment. In this case, if the instruction specifies Writeback, then the load or store is performed without Writeback.

The Cortex®-A76 core does not implement a *Read 0* or *Ignore Write* policy on *unpredictable* use of R15 by instruction. Instead, the Cortex®-A76 core takes an *undefined* exception trap.

Load/Store accesses crossing page boundaries

The Cortex®-A76 core implements a set of behaviors for load or store accesses that cross page boundaries.

Crossing a page boundary with different memory types or shareability attributes

The *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*, states that a memory access from a load or store instruction that crosses a page boundary to a memory location that has a different memory type or shareability attribute results in *constrained unpredictable* behavior.

Crossing a 4KB boundary with a Device access

The *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*, states that a memory access from a load or store instruction to Device memory that crosses a 4KB boundary results in *constrained unpredictable* behavior.

Implementation (for both page boundary specifications)

For an access that crosses a page boundary, the Cortex®-A76 core implements the following behaviors:

- Store crossing a page boundary:
 - No alignment fault.
 - The access is split into two stores.
 - Each store uses the memory type and shareability attributes associated with its own address.
- Load crossing a page boundary (Device to Device and Normal to Normal):
 - No alignment fault.
 - The access is split into two loads.
 - Each load uses the memory type and shareability attributes associated with its own address.
- Load crossing a page boundary (Device to Normal and Normal to Device):
 - The instruction will generate an alignment fault.

Armv8 Debug UNPREDICTABLE behaviors

This section describes the behavior that the Cortex®-A76 core implements when:

- A topic has multiple options.
- The behavior differs from either or both of the Options and Preferences behaviors.



Note: This section does not describe the behavior when a topic only has a single option and the core implements the preferred behavior.

Table 91: Armv8 Debug *unpredictable* behaviors

Scenario	Behavior
A32 BKPT instruction with condition code not AL	The core implements the following preferred option: <ul style="list-style-type: none"> Executed unconditionally.
Address match breakpoint match only on second halfword of an instruction	The core generates a breakpoint on the instruction if CPSR.IL=0. In the case of CPSR.IL=1, the core does not generate a breakpoint exception.
Address matching breakpoint on A32 instruction with DBGBCRn.BAS=1100	The core implements the following option: <ul style="list-style-type: none"> Does match if CPSR.IL=0.
Address match breakpoint match on T32 instruction at DBGBCRn+2 with DBGBCRn.BAS=1111	The core implements the following option: <ul style="list-style-type: none"> Does match.
Link to non-existent breakpoint or breakpoint that is not context-aware	The core implements the following option: <ul style="list-style-type: none"> No Breakpoint or Watchpoint debug event is generated, and the LBN field of the <i>linker</i> reads <i>unknown</i>.
DBGWCRn_EL1.MASK!=00000 and DBGWCRn_EL1.BAS!=11111111	The core behaves as indicated in the sole Preference: <ul style="list-style-type: none"> DBGWCRn_EL1.BAS is <i>ignored</i> and treated as if 11111111.
Address match breakpoint with DBGBCRn_EL1.BAS=0000	The core implements the following option: <ul style="list-style-type: none"> As if disabled.
DBGWCRn_EL1.BAS specifies a non-contiguous set of bytes within a double-word	The core implements the following option: <ul style="list-style-type: none"> A Watchpoint debug event is generated for each byte.
A32 HLT instruction with condition code not AL	The core implements the following option: <ul style="list-style-type: none"> Executed unconditionally.
Execute instruction at a given EL when the corresponding EDECCR bit is 1 and Halting is allowed	The core behaves as follows: <ul style="list-style-type: none"> Generates debug event and Halt no later than the instruction following the next <i>Context Synchronization operation</i> (CSO) excluding ISB instruction.
H > N or H = 0 at Non-secure EL1 and EL0, including value read from PMCR_EL0.N	The core implements: <ul style="list-style-type: none"> A simple implementation where all of HPMN[4:0] are implemented, and In Non-secure EL1 and EL0: <ul style="list-style-type: none"> If H > N then M = N. If H = 0 then M = 0.

Scenario	Behavior
H > N or H = 0: value read back in MDCR_EL2.HPMN	<p>The core implements:</p> <ul style="list-style-type: none"> A simple implementation where all of HPMN[4:0] are implemented and for reads of MDCR_EL2.HPMN, return H.
P ≥ M and P ≠ 31: reads and writes of PMXEVTYPER_EL0 and PMXEVCNTR_EL0	<p>The core implements:</p> <ul style="list-style-type: none"> A simple implementation where all of SEL[4:0] are implemented, and if P ≥ M and P ≠ 31 then the register is <i>res0</i>.
P ≥ M and P ≠ 31: value read in PMSELR_EL0.SEL	<p>The core implements:</p> <ul style="list-style-type: none"> A simple implementation where all of SEL[4:0] are implemented, and if P ≥ M and P ≠ 31 then the register is <i>res0</i>.
P = 31: reads and writes of PMXEVCNTR_EL0	<p>The core implements:</p> <ul style="list-style-type: none"> <i>res0</i>.
n ≥ M: Direct access to PMEVCNTRn_EL0 and PMEVTYPERn_EL0	<p>The core implements:</p> <ul style="list-style-type: none"> If n ≥ N, then the instruction is <i>unallocated</i>. Otherwise if n ≥ M, then the register is <i>res0</i>.
Exiting Debug state while instruction issued through EDITR is in flight	<p>The core implements the following option:</p> <ul style="list-style-type: none"> The instruction completes in Debug state before executing the restart.
Using memory-access mode with a non-word-aligned address	<p>The core behaves as indicated in the sole Preference:</p> <ul style="list-style-type: none"> Does unaligned accesses, faulting if these are not permitted for the memory type.
Access to memory-mapped registers mapped to Normal memory	<p>The core behaves as indicated in the sole Preference:</p> <ul style="list-style-type: none"> The access is generated, and accesses might be repeated, gathered, split or resized, in accordance with the rules for Normal memory, meaning the effect is <i>unpredictable</i>.
Not word-sized accesses or (AArch64 only) doubleword-sized accesses >	<p>The core behaves as indicated in the sole Preference:</p> <ul style="list-style-type: none"> Reads occur and return <i>unknown</i> data. Writes set the accessed register(s) to <i>unknown</i>.
External debug write to register that is being reset	<p>The core behaves as indicated in the sole Preference:</p> <ul style="list-style-type: none"> Takes reset value.

Scenario	Behavior
Accessing reserved debug registers	<p>The core deviates from preferred behavior because the hardware cost to decode some of these addresses in debug power domain is significantly high.</p> <p>The actual behavior is:</p> <ol style="list-style-type: none"> For reserved debug registers in the address range 000-CFC and Performance Monitors registers in the address range 000, the response is either <i>constrained unpredictable</i> Error or <i>res0</i> when any of the following errors occurs: <ul style="list-style-type: none"> Off The core power domain is either completely off or in a low-power state where the core power domain registers cannot be accessed. DLK <code>DoubleLockStatus()</code> is TRUE and OS double-lock is locked (EDPRSR.DLK is 1). OSLK OS lock is locked (OSLSR_EL1.OSLK is 1). For reserved debug registers in the address ranges 400-4FC and 800-8FC, the response is <i>constrained unpredictable</i> Error or <i>res0</i> when the conditions in 1 do not apply and the following error occurs: <ul style="list-style-type: none"> EDAD <code>AllowExternalDebugAccess()</code> is FALSE. External debug access is disabled. For reserved Performance Monitor registers in the address ranges 000-0FC and 400-47C, the response is either <i>constrained unpredictable</i> Error, or <i>res0</i> when the conditions in 1 and 2 do not apply, and the following error occurs: <ul style="list-style-type: none"> EPMAD <code>AllowExternalPMUAccess()</code> is FALSE. External Performance Monitors access is disabled.
Clearing the <i>clear-after-read</i> EDPRSR bits when Core power domain is on, and <code>DoubleLockStatus()</code> is TRUE	<p>The core behaves as indicated in the sole Preference:</p> <ul style="list-style-type: none"> Bits are not cleared to zero.

Other UNPREDICTABLE behaviors

This section describes other *unpredictable* behaviors.

Table 92: Other *unpredictable* behaviors

Scenario	Description
CSSELR indicates a cache that is not implemented.	<p>If CSSELR indicates a cache that is not implemented, then on a read of the CCSIDR the behavior is <i>constrained unpredictable</i>, and can be one of the following:</p> <ul style="list-style-type: none">• The CCSIDR read is treated as NOP.• The CCSIDR read is <i>undefined</i>.• The CCSIDR read returns an <i>unknown</i> value (preferred).
HDCR.HPMN is set to 0, or to a value larger than PMCR.N.	<p>If HDCR.HPMN is set to 0, or to a value larger than PMCR.N, then the behavior in Non-secure EL0 and EL1 is <i>constrained unpredictable</i>, and one of the following must happen:</p> <ul style="list-style-type: none">• The number of counters accessible is an <i>unknown</i> non-zero value less than PMCR.N.• There is no access to any counters. <p>For reads of HDCR.HPMN by EL2 or higher, if this field is set to 0 or to a value larger than PMCR.N, the core must return a <i>constrained unpredictable</i> value that is one of:</p> <ul style="list-style-type: none">• PMCR.N.• The value that was written to HDCR.HPMN.• (The value that was written to HDCR.HPMN) modulo 2h, where h is the smallest number of bits required for a value in the range 0 to PMCR.N.
CRC32 or CRC32C instruction with size==64.	<p>On read of the instruction, the behavior is <i>constrained unpredictable</i>, and the instruction executes with the additional decode: size==32.</p>
CRC32 or CRC32C instruction with cond!=1110 in the A1 encoding.	<p>The core implements the following option:</p> <ul style="list-style-type: none">• Executed unconditionally.

Appendix

B

Revisions

Topics:

- [Revisions](#)

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

Revisions

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

Table 93: Issue 0000-00

Change	Location	Affects
First release	-	-

Table 94: Differences between Issue 0000-00 and Issue 0100-00

Change	Location	Affects
Added support for 128KB L2 cache size.	Throughout document	r1p0
Added note to indicate support for Dot Product instructions introduced in the Armv8.4 Extension.	About the core	r1p0
Updated BPIQ data location encoding table.	Enyo Encoding for L1 instruction cache tag, L1 instruction cache data, L1 BTB, L1 GHB, L1 TLB instruction, and BPIQ	r1p0
Updated replacement policy to dynamic biased replacement policy.	About the L2 memory system	r1p0
Updated reset values for ID_AA64ISAR0_EL1, IDAA64MMFR1_EL1, IDMMFR4_EL1, and MIDR_EL1.	Enyo/Deimos - AArch64 registers by functional group	r1p0
Updated CCSIDR_EL1 encodings table.	CCSIDR_EL1, Cache Size ID Register, EL1	r1p0
Updated CPUECTLR_EL1 register description.	CPUECTLR_EL1, CPU Extended Control Register, EL1	r1p0
Updated bits [43:32] of ID_AA64ISAR0_EL1 register.	ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1	r1p0
Updated bits [15:12] of ID_AA64MMFR1_EL1 register.	ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1	r1p0
Added ID_ISAR6_EL1 register.	ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1	r1p0
Added Activity Monitor Unit chapter.	Activity Monitor Unit	All versions
Updated reset value for TRCIDR1 register.	ETM register summary	r1p0

Change	Location	Affects
Updated bits [3:0] of TRCIDR1 register.	TRCIDR1, ID Register 1	r1p0

Table 95: Differences between Issue 0100-00 and Issue 0200-00

Change	Location	Affects
Fixed typographical errors.	Throughout document	-
Added information for L1 Prefetch History Table.	Table 1	r2p0
Updated ATCR_EL12 description.	Table 1	r2p0
Updated reset value for ID_AA64PFR0_EL1.	Enyo/Deimos - AArch64 registers by functional group	r2p0
Updated reset value for ID_PFR0_EL1.	Enyo/Deimos - AArch64 registers by functional group	r2p0
Added new register ID_PFR2_EL1.	Enyo/Deimos - AArch64 registers by functional group ID_PFR2_EL1, AArch32 Processor Feature Register 2, EL1	r2p0
Updated reset value for MIDR_EL1.	Enyo/Deimos - AArch64 registers by functional group MIDR_EL1, Main ID Register, EL1	r2p0
Added CSV2 and CSV3 fields to register.	ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1	r2p0
Added CSV2 field to register.	ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1	r2p0
Added TRCVMIDCCTLR0 register description.	TRCVMIDCCTLR0, Virtual context identifier Comparator Control Register 0	r2p0

Table 96: Differences between Issue 0200-00 and Issue 0300-00

Change	Location	Affects
Fixed typographical errors.	Throughout document	-
Added new register ID_AA64PFR1_EL1.	ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1, EL1	r3p0
Updated reset value for ID_PFR2_EL1.	Enyo/Deimos - AArch64 registers by functional group	r3p0
Updated reset value for MIDR_EL1.	Enyo/Deimos - AArch64 registers by functional group	r3p0

Change	Location	Affects
Updated reset value for TRCIDR1.	ETM register summary	r3p0
Added SSBS field to ID_AA64PFR1_EL1.	ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1, EL1	r3p0

Table 97: Differences between Issue 0300-00 and Issue 0301-00

Change	Location	Affects
Updated reset value for MIDR_EL1.	Enyo/Deimos - AArch64 registers by functional group	r3p1

Glossary

A32

The instruction set used by an ARMv8 processor that is in AArch32 execution state. A32 is a fixed-width instruction set that uses 32-bit instruction encoding. It is compatible with the ARMv7 ARM instruction set. See Also AArch32, T32.

A32 instruction

An instruction executed by a core that is in AArch32 Execution state and A32 Instruction set state. A32 is a fixed-width instruction set that uses 32-bit instruction encodings. Previously, this instruction set was called the Arm instruction set.

A32 state

When a core is in the AArch32 Execution state, if it is in the A32 Instruction set state then it executes A32 instructions.

A64

The instruction set used by an ARMv8 processor that is in AArch64 execution state. A64 is a fixed-width instruction set that uses 32-bit instruction encoding. See Also AArch64, A32, T32.

A64 instruction

The instruction set used by an ARMv8-A core that is in AArch64 Execution state. A64 is a fixed-width instruction set that uses 32-bit instruction encodings.

AAPCS

Defines how registers and the stack are used for subroutine calls.

AArch32

The ARM 32-bit Execution state that uses 32-bit general purpose registers, and a 32-bit program counter (PC), stack pointer (SP), and link register (LR). AArch32 Execution state provides a choice of two instruction sets, A32 and T32. In implementations of versions of the ARM architecture before ARMv8, and in the ARM R and M architecture profiles, execution is always in AArch32 state.

AArch64

The ARM 64-bit Execution state that uses 64-bit general purpose registers, and a 64-bit program counter (PC), stack pointer (SP), and exception link registers (ELR). AArch64 Execution state provides a single instruction set, A64. AArch64 state is supported only in the ARMv8-A architecture profile.

ABI

A collection of specifications, some open and some specific to the Arm architecture, that regulate the inter-operation of binary code in a range of execution environments for Arm processors. The base standard specifies those aspects of code generation that must conform to a standard that supports inter-operation. It is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

abort

An abort occurs when an illegal memory access causes an exception. An abort can be generated by the hardware that manages memory accesses, or by the external memory system. The hardware that generates the abort might be a Memory Management Unit (MMU) or a Memory Protection Unit (MPU).

abort model

Describes the changes to the core state when a Data abort exception occurs. Different abort models behave differently with regard to load and store instructions that specify base register writeback.

ACE

The AXI Coherency Extensions (ACE) provide additional channels and signaling to an AXI interface to support system level cache coherency.

ACE interface

An AMBA AXI4 interface that includes full support for the ACE protocol. An ACE interface adds: *Signals to some of the AXI4 channels. *Channels to the AXI4 interface.

ACE protocol

The AXI Coherency Extensions protocol, that adds signals to an AMBA AXI4 interface, to support managing the coherency of a distributed memory system.

ACE-Lite interface

Use AMBA ACE on first use. After that, use ACE.

adaptive clocking

A technique where the debug interface hardware sends out a clock signal and then waits for the returned clock before generating the next clock pulse. This technique enables the run control unit in the debug hardware to adapt to differing signal drive capabilities and differing cable lengths.

addressing mode

A method of generating the memory address that a load or store instruction uses. The addressing modes mechanism can generate values for data-processing instructions to use as operands.

Advanced eXtensible Interface

An AMBA bus protocol that supports: separate phases for address or control and data unaligned data transfers using byte strobes burst-based transactions with only start address issued separate read and write data channels issuing multiple outstanding addresses out-of-order transaction completion addition of register stages to provide timing closure. The AXI protocol includes optional signaling extensions for low-power operation. See Also AXI Coherency Extensions (ACE).

Advanced High-performance Bus

An AMBA bus protocol supporting pipelined operation, with the address and data phases occurring during different clock periods. This means the address phase of a transfer can occur during the data phase of the previous transfer. AHB provides a subset of the functionality of the AMBA AXI protocol. See Also Advanced Microcontroller Bus Architecture (AMBA) and AHB-Lite.

Advanced Microcontroller Bus Architecture

The AMBA family of protocol specifications is the ARM open standard for on-chip buses. AMBA provides solutions for the interconnection and management of the functional blocks that make up a System-on-Chip (SoC). Applications include the development of embedded systems with one or more processors or signal processors and multiple peripherals.

Advanced Peripheral Bus

An AMBA bus protocol for ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Using APB to connect to the main system bus through a system-to-peripheral bus bridge can help reduce system power consumption.

Advanced SIMD

A feature of the ARM architecture that provides Single Instruction Multiple Data (SIMD) operations on a dedicated bank of registers. If an implementation also supports scalar floating-point instructions, the floating-point and Advanced SIMD instructions use a common register bank. ARM NEON technology provides the Advanced SIMD instructions, and therefore these are sometimes called the NEON instructions.

Advanced Trace Bus

An AMBA bus protocol for trace data. A trace device can use an ATB to share CoreSight capture resources. Use AMBA ATB on first use, and ATB thereafter.

AEL

A version of embedded Linux OS ported to the Arm architecture.

AHB

An AMBA bus protocol supporting pipelined operation, with the address and data phases occurring during different clock periods. This means the address phase of a transfer can occur during the data phase of the previous transfer.

AHB provides a subset of the functionality of the AMBA AXI protocol. See Also Advanced Microcontroller Bus Architecture (AMBA) and AHB-Lite.

AHB Access Port

An optional component of the DAP that provides an AHB interface to a SoC. CoreSight supports access to a system bus infrastructure using the AHB Access Port (AHB-AP) in the Debug Access Port (DAP). The AHB-AP provides an AHB master port for direct access to system memory. Other bus protocols can use AHB bridges to map transactions. For example, you can use AHB to AXI bridges to provide AHB access to an AXI bus matrix. See Also Debug Access Port (DAP).

AHB Trace Macrocell

A trace source that makes bus information visible. This information cannot be inferred from the processor using just a trace macrocell. HTM trace can provide: An understanding of multi-layer bus utilization. Software debug. For example, visibility of access to memory areas and data accesses. Bus event detection for trace trigger or filters, and for bus profiling. See Also Advanced High-performance Bus (AHB).

AHB-AP

An optional component of the DAP that provides an AHB interface to a SoC. CoreSight supports access to a system bus infrastructure using the AHB Access Port (AHB-AP) in the Debug Access Port (DAP). The AHB-AP provides an AHB master port for direct access to system memory. Other bus protocols can use AHB bridges to map transactions. For example, you can use AHB to AXI bridges to provide AHB access to an AXI bus matrix. See Also Debug Access Port (DAP).

AHB-Lite

A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect.

APB

An AMBA bus protocol for ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Using APB to connect to the main system bus through a system-to-peripheral bus bridge can help reduce system power consumption.

APB Access Port

An optional component of the Debug Access Point (DAP) that provides an APB interface to a SoC, usually to its main functional buses.

APB-AP

An optional component of the Debug Access Point (DAP) that provides an APB interface to a SoC, usually to its main functional buses.

Application Binary Interface for the Arm Architecture

A collection of specifications, some open and some specific to the Arm architecture, that regulate the inter-operation of binary code in a range of execution environments for Arm processors. The base standard specifies those aspects of code generation that must conform to a standard that supports inter-operation. It is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

Application Processor Status Register

In AArch32 User mode, a restricted form of the CPSR.

APSR

In AArch32 User mode, a restricted form of the CPSR.

Arm Compiler for DS-5

Arm Compiler for DS-5 is a suite of tools, together with supporting documentation and examples, that you can use to write and build applications for the Arm family of processors. Arm Compiler for DS-5 supersedes RealView Compilation Tools. See Also `armar`, `armasm`, `armcc`, `fromelf`.

Arm Debug Interface

The ADI connects a debugger to a device. The ADI is used to access memory-mapped components in a system, such as processors and CoreSight components. The ADI protocol defines the physical wire protocols permitted, and the logical programmers model.

Arm Embedded Linux

A version of embedded Linux OS ported to the Arm architecture.

Arm instruction

An instruction executed by a core that is in AArch32 Execution state and A32 Instruction set state. A32 is a fixed-width instruction set that uses 32-bit instruction encodings. Previously, this instruction set was called the Arm instruction set.

Arm profiler

A plug-in to the ARM Workbench Integrated Development Environment that provides non-intrusive analysis of embedded software over time, on targets running at frequencies which are typically as high as 250MHz. Targets can be Real-Time System Models (RTSMs) and hardware targets. See Also RealView Development Suite (RVDS).

Arm state

When a core is in the AArch32 Execution state, if it is in the A32 Instruction set state then it executes A32 instructions.

Arm TrustZone technology

The hardware and software that enable the integration of enhanced security features throughout a SoC. In Armv6K and Armv7, the Security Extensions implement the TrustZone hardware. In Armv8, EL3 incorporates the TrustZone hardware.

Arm Workbench IDE

Arm Workbench IDE is based around the Eclipse IDE, and provides additional features to support the Arm development tools provided in RVDS. See Also RealView Development Suite (RVDS).

armar

The ARM librarian which enables you to create libraries of files, such as object files. See Also Development Studio 5 (DS-5) and RealView Compilation Tools (RVCT).

armasm

The ARM assembler. This converts ARM assembly language into machine code. See Also Development Studio 5 (DS-5) and RealView Compilation Tools (RVCT).

armcc

The ARM compiler for C and C++ code. See Also Development Studio 5 (DS-5) and RealView Compilation Tools (RVCT).

ArtiGrid

A power routing scheme, also referred to as Over The Cell. See Also Over The Cell (OTC).

ATB

An AMBA bus protocol for trace data. A trace device can use an ATB to share CoreSight capture resources. Use AMBA ATB on first use, and ATB thereafter.

ATB bridge

A synchronous ATB bridge provides a register slice that meets timing requirements by adding a pipeline stage. It provides a unidirectional link between two synchronous ATB domains. An asynchronous ATB bridge provides a unidirectional link between two ATB domains with asynchronous clocks. This means it connects components in different clock domains.

atomicity

Describes actions that appear to happen as a single operation. In the ARM architecture, atomicity refers to either single-copy atomicity or multi-copy atomicity. The ARM Architecture Reference Manuals define these forms of atomicity.

ATPG

Automated Test Pattern Generation: The process of using a specialized software tool to automatically generate manufacturing test vectors for an ASIC design.

authentication asynchronous bridge

Transfers authentication signals between two asynchronous clock domains.

authentication synchronous bridge

Transfers authentication signals between two synchronous clock domains.

Automatic Test Pattern Generation

Automated Test Pattern Generation: The process of using a specialized software tool to automatically generate manufacturing test vectors for an ASIC design.

AWIDE

Arm Workbench IDE is based around the Eclipse IDE, and provides additional features to support the Arm development tools provided in RVDS. See Also RealView Development Suite (RVDS).

AXI

An AMBA bus protocol that supports: separate phases for address or control and data unaligned data transfers using byte strobes burst-based transactions with only start address issued separate read and write data channels issuing multiple outstanding addresses out-of-order transaction completion addition of register stages to provide timing closure. The AXI protocol includes optional signaling extensions for low-power operation. See Also AXI Coherency Extensions (ACE).

AXI Coherency Extensions

The AXI Coherency Extensions (ACE) provide additional channels and signaling to an AXI interface to support system level cache coherency.

AXI low-power interface

Deprecated variant of AXI low-power interface

back-annotation

The process of applying timing characteristics from the implementation process onto a model.

banked register

A register that has multiple instances. A property of the state of the device determines which instance is in use. For example, the Security state might determine which instance is in use.

Base Platform Application Binary Interface

The base standard for the interface between executable files, such as dynamic shared objects and DLLs, and the systems that execute them.

base porting layer

A platform-dependent base driver software component that communicates with the Mali GPU. For example, the base porting layer controls the Mali GPU registers. You implement, or port, the base porting layer onto different target platforms.

base register

A register specified by a load or store instruction that is used as the base value for the address calculation for the instruction. Depending on the instruction, an offset can be added to or subtracted from the base register value to form the address that is used for the memory access.

Base Standard Application Binary Interface

A collection of specifications, some open and some specific to the Arm architecture, that regulate the inter-operation of binary code in a range of execution environments for Arm processors. The base standard specifies those aspects of code generation that must conform to a standard that supports inter-operation. It is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

BCD file

In the context of RealView Debugger, a BCD file enables you to define the memory map and memory-mapped registers for a target development board or processor. ARM provides various BCD files with RVDS for ARM development boards. See Also Board file and Debug configuration.

big-endian

In the context of the ARM architecture, big-endian is defined as the memory organization in which the least significant byte of a word is at a higher address than the most significant byte, for example: *A byte or halfword at a word-aligned address is the most significant byte or halfword in the word at that address. *A byte at a halfword-aligned address is the most significant byte in the halfword at that address.

Board and Chip Definition file

In the context of RealView Debugger, a BCD file enables you to define the memory map and memory-mapped registers for a target development board or processor. ARM provides various BCD files with RVDS for ARM development boards. See Also Board file and Debug configuration.

board file

A debugger uses this term to refer to the top-level configuration file, normally called `rvdebug.brd`, that references one or more other configuration files. A board file contains: the Debug Configuration (connection-level) settings, references to the Debug Interface configuration file that identifies the targets on the development platform, references to any Board and Chip Definition (BCD) files assigned to a Debug Configuration. See Also Board and Chip Definition (BCD) file and Debug configuration.

bounce

In an ARMv7 floating-point implementation that includes a VFP subarchitecture, a mechanism for handling floating point exceptions and instructions that are not supported by the hardware. A bounce generates an Undefined Instruction exception, and the exception handler can call support code to respond to the bounce.

boundary scan chain

A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain, connected between TDI and TDO, through which test data is shifted. A processor can contain several shift registers, enabling you to access selected parts of the device.

BPABI

The base standard for the interface between executable files, such as dynamic shared objects and DLLs, and the systems that execute them.

branch folding

A technique where, on the prediction of a branch, the target instructions are completely removed from the instruction stream presented to the execution pipeline. Branch folding can significantly improve the performance of branches, and take the CPI for branches below one.

branch phantom

Branch target instructions speculatively executed, in parallel with the main instruction stream, as a result of branch folding.

branch prediction

The selection of a future execution path for instruction fetch. For example, after a branch instruction, the processor can choose to speculatively fetch either the instruction following the branch or the instruction at the branch target. See Also Prefetching.

breakpoint

A debug event triggered by the execution of a particular instruction. It is specified by one or both of the address of the instruction and the state of the processor when the instruction is executed. See Also Watchpoint.

BSABI

A collection of specifications, some open and some specific to the Arm architecture, that regulate the inter-operation of binary code in a range of execution environments for Arm processors. The base standard specifies those aspects of code generation that must conform to a standard that supports inter-operation. It is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

byte lane strobe

A signal that determines which byte lanes are active, or valid, in a data transfer. Each bit of this signal corresponds to eight bits of the data bus.

byte swizzling

Re-arranging the order of bytes in a word or halfword.

byte-invariant

In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. The ARM architecture supports byte-invariant systems in ARMv6 and later versions. See Also Word-invariant.

cacheable

A data storage method in which, if a memory location to be written is not in cache memory, a cache line is allocated for the memory. The value of that memory is then loaded into the cache from main memory, and the new value for the location is written to cache.

CADI

The debug control and inspection API to a fast model.

Canonical Frame Address

In Debug With Arbitrary Record Format (DWARF), this is an address on the stack specifying where the call frame of an interrupted function is located.

captive thread

Captive threads are all threads that can be brought under the control of RVDS. Special threads, called non-captive threads, are essential to the operation of Running System Debug (RSD) and so are not under debugger control. See Also Running System Debug (RSD).

cast out

A cache line, selected to be discarded to make room for a replacement cache line that is required because of a cache miss. How the victim is selected for eviction is processor-specific. A victim is also known as a cast out.

CFA

In Debug With Arbitrary Record Format (DWARF), this is an address on the stack specifying where the call frame of an interrupted function is located.

chained breakpoint

In the context of an ARM debugger, a complex breakpoint that comprises multiple hardware breakpoint units. See Also Breakpoint unit and Conditional breakpoint.

chained tracepoint

In the context of an ARM debugger, a complex tracepoint that comprises multiple tracepoint units. See Also Tracepoint unit and Tracepoint.

channel interface

In an ECT device, channel interface is one of the interfaces on CTI.

characterized

Designates a cell that includes timing data.

clean

A cache line that has not been modified while it is in the cache is said to be clean. To clean a cache is to write dirty cache entries to main memory.

clock gating

Gating a clock signal for a macrocell or functional block with a control signal and using the modified clock that results to control the operating state of the macrocell or block.

Clocks Per Instruction

The average number of clock cycles per instruction for a program or program fragment. It is the multiplicative inverse of instructions per cycle. You can use this value to compare the performance of different processors that implement the same instruction set. The lower the value, the better the performance.

cluster

A cluster is a collection of cores, particularly having a shared cache. Cluster is preferred to the terms "multiprocessor" and "multi-core".

CMM

In the context of an ARM debugger, a scripting language provided for compatibility with other debuggers. If you are writing new scripts, ARM recommends that you use the GNU Debugger (GDB) scripting commands because these offer more functionality in the ARM Debuggers.

coherence order

Data accesses from a set of observers to a byte in memory are coherent if accesses to that byte by the members of the set of observers are consistent with there being a single total order of all writes to that byte in memory by all members of the set of observers. This single total order of all writes to that byte is the coherence order for that byte.

coherent

Data accesses from a set of observers to a byte in memory are coherent if accesses to that byte by the members of the set of observers are consistent with there being a single total order of all writes to that byte in memory by all members of the set of observers. This single total order of all writes to that byte is the coherence order for that byte.

Cold reset

A Cold reset has the same effect as starting the processor by turning the power on. This clears main memory and many internal settings. Some program failures can lock up the core and require a Cold reset to restart the system.

Communications channel

The hardware used for communicating between the software running on the processor, and an external host, using the debug interface. When this communication is for debug purposes, it is called the Debug Communications Channel (DCC). See Also Debug Communications Channel (DCC).

Condensed Reference Format

An ARM proprietary file format for specifying test vectors. Typically, ARM supplies a script to convert CRF format to Verilog Reference Format (VRF).

condition code check

The process of determining whether a conditional instruction executes normally or is treated as a NOP. For an instruction that includes a condition code field, that field is compared with the condition flags to determine whether the instruction is executed normally. For a T32 instruction in an IT block, the value of the ITSTATE register determines whether the instruction is executed normally.

condition code field

A four-bit field in an ARM instruction that specifies the condition under which the instruction executes. See Also Conditional execution and Condition flags.

condition flags

Condition flags is the agreed term. The condition flags are the N, Z, C, and V bits of PSTATE or of a Program Status Register (PSR).

conditional breakpoint

A breakpoint that has one or more condition qualifiers assigned. The breakpoint is activated when all assigned conditions are met, and either stops or continues execution depending on the action qualifiers that are assigned. The condition normally references the values of program variables that are in scope at the breakpoint location. See Also Chained breakpoint and Software breakpoint.

conditional execution

When a conditional instruction starts executing, if the condition code check returns TRUE, the instruction executes normally. Otherwise, it is treated as NOP.

CONSTRAINED UNPREDICTABLE

Where an instruction can result in unpredictable behavior, the ARM architecture can specify a narrow range of permitted behaviors. This range is the range of CONSTRAINED UNPREDICTABLE behavior. All implementations that are compliant with the architecture must follow the CONSTRAINED UNPREDICTABLE behavior. However, software must not rely on any CONSTRAINED UNPREDICTABLE behavior. When CONSTRAINED UNPREDICTABLE appears in body text, it is always in SMALL CAPITALS.

context switch

The saving and restoring of computational state when switching between different threads or processes. Context switch describes any situation where the context is switched by an operating system and might or might not include changes to the address space.

Context synchronization event

Depending on the context, using 'Context synchronization event' or 'Context synchronization instruction'. One of:
 ,Äø Performing an ISB operation. An ISB operation is performed when an ISB instruction is executed and does not fail its condition code check.
 ,Äø Taking an exception.
 ,Äø Returning from an exception.
 ,Äø Exit from Debug state.
 ,Äø Executing a DCPS instruction.
 ,Äø Executing a DRPS instruction. The architecture requires a Context synchronization event to guarantee visibility of any change to a System register.

coprocessor

A processor, or conceptual processor, that supplements the main processor to carry out additional functions. In AArch32 Execution state, the ARM architecture defines an interface to up to 16 coprocessors, CP0-CP15. In ARMv8, AArch32 state supports only conceptual coprocessors CP10, CP11, CP14, and CP15. In previous versions of the architecture, coprocessors CP0-CP7 are available for implementation defined features, and coprocessors CP8-CP15 are reserved for use by ARM. In all architecture versions for the A and R architecture profiles, in AArch32 state: CP15 instructions access the System registers. Some documentation describes this set of registers as the System Control Coprocessor. CP14 instructions access System registers for debug, trace, and execution environment features. The CP10 and CP11 instruction space is for floating-point and Advanced SIMD instructions if supported, including the instructions for accessing the floating-point and Advanced SIMD System registers.

core

Core is used to describe a single processing unit. In the application processor area we can further define core as something that has exclusive use of its own program counter (PC). Never use CPU to refer to a core or PE. Do not try to introduce local definitions of CPU to get round this rule.

core module

In the context of an ARM Integrator development board, an add-on development board that contains an ARM processor and local memory. Core modules can run standalone, or can be stacked onto Integrator development boards. See Also Integrator.

core register

Use 'source general-purpose register'. Processing registers used in AArch32 Execution state, comprising: 13 general-purpose registers, R0 to R12, that software uses for all data processing when using the base instruction set instructions. SP, the Stack Pointer, that can also be referred to as R13. LR, the Link Register, that can also be referred to as R14. PC, the Program Counter, that can also be referred to as R15. In some situations, software can use SP, LR, and PC for processing. The instruction descriptions include any constraints on the use of SP, LR, and PC.

Core reset

Also known as a core reset. Initializes most of the processor functionality, excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor. See also Cold reset.

CoreSight ECT

A modular system that supports the interaction and synchronization of multiple triggering events with an SoC. It comprises: Cross Trigger Interface (CTI) Cross Trigger Matrix (CTM).

CoreSight ETB

A Logic block that extends the information capture functionality of a trace macrocell.

CoreSight ETM

A hardware macrocell that, when connected to a processor, outputs trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol. An ETM always supports instruction trace, and might support data trace.

Cortex Microcontroller Software Interface Standard

(CMSIS) Defines a common way to access peripheral registers or define exception vectors. CMSIS is the name of the core exception vectors, and a device-independent interface for RTOS kernels, including a debug channel.

CPI

The average number of clock cycles per instruction for a program or program fragment. It is the multiplicative inverse of instructions per cycle. You can use this value to compare the performance of different processors that implement the same instruction set. The lower the value, the better the performance.

CPSR

In AArch32 state, the register that holds the current program status.

CRF

An ARM proprietary file format for specifying test vectors. Typically, ARM supplies a script to convert CRF format to Verilog Reference Format (VRF).

Cross Trigger Interface

Part of an Embedded Cross Trigger (ECT) device. In an ECT, the CTI provides the interface between a processor or ETM and the CTM.

Cross Trigger Matrix

In an ECT device, the CTM combines the trigger requests generated by CTIs and broadcasts them to all CTIs as channel triggers.

cross-path blocking

Cross-path blocking occurs when a divergent node has congestion on one of its output nodes which blocks bus traffic to its other output nodes.

CTI

Part of an Embedded Cross Trigger (ECT) device. In an ECT, the CTI provides the interface between a processor or ETM and the CTM.

CTM

In an ECT device, the CTM combines the trigger requests generated by CTIs and broadcasts them to all CTIs as channel triggers.

Current Program Status Register

In AArch32 state, the register that holds the current program status.

Cycles Per Instruction

The average number of clock cycles per instruction for a program or program fragment. It is the multiplicative inverse of instructions per cycle. You can use this value to compare the performance of different processors that implement the same instruction set. The lower the value, the better the performance.

DA

Debug Agent: In RealView Debugger, the debug agent provides target-side support for Running System Debug (RSD). The debug agent can be a thread or be built into the RTOS. The debug agent and RealView Debugger communicate with each other using the DCC. This passes data between the debugger and the target using a hardware debug interface, without stopping the program or entering debug state. See Also Running System Debug (RSD) and Debug Communications Channel (DCC). -or- **Design Assurance**, or **Design Automation** - Design Assurance = Validation; Design Automation = in-house EDA automation infrastructure

DAP

A block that acts as a master on a system bus and provides access to the bus from an external debugger.

Data Abort

An indication to the core of an attempted data access that is not permitted. The Data Abort might be generated by access permission checks performed by the memory system on the core, or might be signaled by the memory system.

data breakpoint

A hardware breakpoint that activates when an access to a specified location meets a set of specified conditions. The conditions can include a check for a specific data value being accessed at the given location. See Also Chained breakpoint and Conditional breakpoint.

Data Timing Module

In the context of physical IP, a data timing module that synchronizes incoming and outgoing data. The DTM is a component of PHY to include I/Os and PLL.

data-active write transaction

A transaction that has completed the address transfer or leading write data transfer, but has not completed all of its data transfers.

DBGTAP

A debug control and data interface based on IEEE 1149.1 JTAG Test Access Port (TAP). The interface has four or five signals.

DCC

A channel that carries data between a debugger and debug logic in the target processor. It can do this without stopping the program flow or causing entry to Debug state, but can also be used when the target is in Debug state. The DCC is part of the debug register interface of the target.

Debug Access Port

A block that acts as a master on a system bus and provides access to the bus from an external debugger.

debug agent

Debug Agent: In RealView Debugger, the debug agent provides target-side support for Running System Debug (RSD). The debug agent can be a thread or be built into the RTOS. The debug agent and RealView Debugger communicate with each other using the DCC. This passes data between the debugger and the target using a hardware debug interface, without stopping the program or entering debug state. See Also Running System Debug (RSD) and Debug Communications Channel (DCC). -or- Design Assurance, or Design Automation - Design Assurance = Validation; Design Automation = in-house EDA automation infrastructure

Debug Communications Channel

A channel that carries data between a debugger and debug logic in the target processor. It can do this without stopping the program flow or causing entry to Debug state, but can also be used when the target is in Debug state. The DCC is part of the debug register interface of the target.

debug configuration

In the context of an ARM debugger, a debug configuration defines a debugging environment for the development platform that is accessed through a particular Debug Interface. Multiple debug configurations can be created for a debug interface, each providing a separate debugging environment to different development platforms, or different debugging environments to the same development platform. All debug configurations are stored in the main debugger board file. Each configuration might reference one or more BCD files. See Also Board file and Target.

debug illusion

The view of the software being debugged that a debugger presents to its user. The features of the debug illusion include: Mapping between assembler code and source code, including displaying assembler and source code simultaneously if required. Support for source-level stepping and breakpoints. Visibility of the source-level function call stack, even when called functions are generated inline. Display of variable values and structure field values, even when these values migrate between various locations. This includes displaying registers and the stack.

debug interface

In the context of RealView Debugger, the Debug interface identifies the targets on your development platform, and provides the mechanism that enables RealView Debugger to communicate with those targets. The Debug interface corresponds directly to a piece of hardware or a software simulator. See Also Debug configuration and Target.

Debug Test Access Port

A debug control and data interface based on IEEE 1149.1 JTAG Test Access Port (TAP). The interface has four or five signals.

Default NaN mode

In floating-point operation, a mode in which all operations that result in a NaN return the default NaN, regardless of the cause of the NaN result. This mode is compliant with the IEEE 754 standard but implies that all information contained in any input NaNs to an operation is lost. See Also NaN.

denormalized value

The IEEE 754-2008 standard term for a floating-point operand with a zero exponent and a nonzero fraction field. ARM documentation describes these operands as denormal or denormalized, as defined by the IEEE 754-1985 standard. Note ARM floating-point implementations comply with the IEEE 754 requirement that denormalized operands are generated and manipulated with the same precision as normal operands. Plus or minus 0 have zero exponent fields, but are not denormals because there is no loss of accuracy.

Design Simulation Model

A functional simulation model of the device derived from the Register Transfer Level (RTL) but that does not reveal its internal structure. The DSM does not model any features added during synthesis such as internal scan chains.

Development Studio 5

The suite of software development tools, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of processors. DS-5 supersedes RealView Development Suite. See Also RealView Development Suite (RVDS).

device

In the context of an ARM debugger, a component on a target. The device contains the application that you want to debug.

Device Validation Suite

Use this set of tests to check the functionality of a device against that defined in the Technical Reference Manual.

Direct-mapped cache

A one-way set-associative cache. Each cache set consists of a single cache line, so cache look-up selects and checks a single cache line.

dirty

A line in a Write-Back cache that has been modified while it is in the cache. Typically, a cache line is marked as dirty by setting the dirty bit to 1.

DNM

A value that must not be altered by software. DNM fields read as UNKNOWN values, and must only be written with the value read from the same field on the same processor.

Do-Not-Modify

A value that must not be altered by software. DNM fields read as UNKNOWN values, and must only be written with the value read from the same field on the same processor.

doubleword

A 64-bit data item. Doublewords are normally at least word-aligned in ARM systems. Must not be hyphenated. Use doubleword.

doubleword-aligned

Hyphenate doubleword-aligned. A data item having a memory address that is divisible by eight.

draw mode

In the context of graphics processing, one of the different ways to specify the primitives to draw. These different ways are called draw modes. The primitives can be specified individually or as a connected strip or fan. They can also be either: non-indexed, meaning that vertices are passed in a vertex array and processed in order; indexed, meaning that vertices are passed as indices into a vertex array.

DS-5

The suite of software development tools, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of processors. DS-5 supersedes RealView Development Suite. See Also RealView Development Suite (RVDS).

DS-5 Debugger

An ARM software development tool that enables you to make use of a debug agent to examine and control the execution of software running on a debug target. It is fully integrated into Eclipse for DS-5. See Also Eclipse for DS-5.

DSM

A functional simulation model of the device derived from the Register Transfer Level (RTL) but that does not reveal its internal structure. The DSM does not model any features added during synthesis such as internal scan chains.

DTM

In the context of physical IP, a data timing module that synchronizes incoming and outgoing data. The DTM is a component of PHY to include I/Os and PLL.

DVS

Use this set of tests to check the functionality of a device against that defined in the Technical Reference Manual.

Early-Z

A Z-testing scheme that performs the actual Z-test before texturing or fragment shading when it is safe to do so. This increases the performance of the Mali GPU and reduces the required bandwidth.

Eclipse for DS-5

Eclipse for DS-5 is based around the Eclipse IDE, and provides additional features to support the ARM development tools provided in DS-5. See Also Development Studio 5 (DS-5).

ECT

A modular system that supports the interaction and synchronization of multiple triggering events with an SoC. It comprises: Cross Trigger Interface (CTI) Cross Trigger Matrix (CTM).

EGL

A standardized set of functions that communicate between graphics software, such as OpenGL ES or OpenVG drivers, and the platform-specific windowing system that displays the image.

ELR

In AArch64 state, there is a dedicated LR for each implemented Exception level, called the Exception Link Register (ELR) for that Exception level, for example, ELR_EL1. The Exception Link Register holds the exception return address.

embedded assembler

Embedded assembler is assembler code that is included in a C or C++ file, and is separate from other C or C++ functions.

Embedded Cross Trigger

A modular system that supports the interaction and synchronization of multiple triggering events with an SoC. It comprises: Cross Trigger Interface (CTI) Cross Trigger Matrix (CTM).

Embedded Trace Buffer

A Logic block that extends the information capture functionality of a trace macrocell.

Embedded Trace Macrocell

A hardware macrocell that, when connected to a processor, outputs trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol. An ETM always supports instruction trace, and might support data trace.

EmbeddedICE logic

An on-chip logic block that provides TAP-based debug support for an ARM processor. It is accessed through the DAP on the ARM processor.

EmbeddedICE-RT

Hardware provided by an ARM processor to aid debugging in real-time.

Embedded-System Graphics Library

A standardized set of functions that communicate between graphics software, such as OpenGL ES or OpenVG drivers, and the platform-specific windowing system that displays the image.

endianity

The scheme that determines the order of the successive bytes of data in a larger data structure when that structure is stored in memory.

endianness

The scheme that determines the order of the successive bytes of data in a larger data structure when that structure is stored in memory.

ESSL

A programming language used to create custom shader programs that can be used in a programmable pipeline, on the Mali GPU. You can also use pre-defined library shaders, written in ESSL.

ESSL compiler

The compiler that translates shaders written in ESSL, into binary code for the shader units in the Mali GPU. There are two versions of the ESSL compiler: the on-target compiler and the offline compiler.

ETB

A Logic block that extends the information capture functionality of a trace macrocell.

ETM

A hardware macrocell that, when connected to a processor, outputs trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol. An ETM always supports instruction trace, and might support data trace.

ETV

Extended Target Visibility enables RealView Debugger to access features of the underlying target such as, for example, chip-level information provided by the hardware manufacturer or SoC designer.

Event

In the context of an ARM trace macrocell, events can be: *Simple -An observable condition that a trace macrocell can use to control aspects of a trace. *Complex -A boolean combination of simple events that a trace macrocell can use to control aspects of a trace.

event asynchronous bridge

A fixed component that synchronizes events on a single channel from the slave domain to the master domain. In addition, the event acknowledge from the master domain is synchronized and signaled to the slave domain.

exception

A mechanism to handle a fault, error event, or external notification. For example, exceptions handle external interrupts and undefined instructions.

Exception level

Do not use EL when you mean Exception level. Use only Exception level.

Exception Link Register

In AArch64 state, there is a dedicated LR for each implemented Exception level, called the Exception Link Register (ELR) for that Exception level, for example, ELR_EL1. The Exception Link Register holds the exception return address.

exception vector

A fixed address that contains the address of the first instruction of the corresponding exception handler.

exceptional state

In an ARMv7 implementation that includes a VFP subarchitecture, in floating-point operation, if the floating-point hardware detects an exceptional condition, the ARM floating-point implementation sets the FPExc bit and loads a copy of the exceptional instruction to the FPINST register. When in the exceptional state, the issue of a trigger instruction to the floating-point extension causes a bounce.

execution vehicle

A part of the debug target interface that processes requests from the client tools to the target. See Also Debug interface.

execution view

The address of regions and sections after the image is loaded into memory and started execution. See Also Scatter-loading and Load view.

explicit access

A read from memory, or a write to memory, generated by a load or store instruction executed by the core. Reads and writes generated by hardware translation table accesses are not explicit accesses. For more information, see the appropriate ARM Architecture Reference Manual.

Extended Target Visibility

Extended Target Visibility enables RealView Debugger to access features of the underlying target such as, for example, chip-level information provided by the hardware manufacturer or SoC designer.

eXtensible Verification Component

A model that provides system or device stimulus and monitor responses. See Also XVC Test Scenario Manager.

Fast Context Switch Extension

Before ARMv8, an extension to the ARM architecture that modifies the behavior of the memory system. It enables multiple programs running on the core to use identical address ranges, while ensuring that the addresses they present to the rest of the memory system differ. From ARMv6, ARM deprecates use of the FCSE. The FCSE is optional in ARMv7, and obsolete from the ARMv7 Multiprocessing Extensions.

Fast Model

Simulation in software of Arm SoCs or subsystems, including processors and peripherals.

fault

An abort generated by the memory system, for example by the Memory Management Unit (MMU) or Memory Protection Unit (MPU).

FCSE

Before ARMv8, an extension to the ARM architecture that modifies the behavior of the memory system. It enables multiple programs running on the core to use identical address ranges, while ensuring that the addresses they present to the rest of the memory system differ. From ARMv6, ARM deprecates use of the FCSE. The FCSE is optional in ARMv7, and obsolete from the ARMv7 Multiprocessing Extensions.

FIQ

FIQ interrupt. nFIQ is one of two interrupt signals on many ARM processors. See Also IRQ.

fixed-function pipeline

In the context of graphics processors, a process that uses standard functions to draw graphics on fixed-function graphics processors, such as the Mali-55 GPU. The fixed-function pipeline is a requirement of OpenGL ES 1.1.

Flash Patch and Breakpoint

In an ARM M-profile processor, an FPB can:remap sections of ROM, typically Flash memory, to regions of RAMset breakpoints on code in ROM. It can be used for debug, and to provide a code or data patch to an application that requires field updates to a product ROM.

flat address mapping

A memory system where the physical address for every access is equal to its virtual address.

floating point

Hyphenate floating-point.

floating-point

Hyphenate floating-point.

flush to-zero

In floating-point operation, a mode that optimizes the performance of some floating-point algorithms by replacing the denormalized operands and intermediate results with zeros, without significantly affecting the accuracy of their final results. Using the Flush to Zero mode is a deviation from IEEE 754.

Flush-to-zero mode

In floating-point operation, a special processing mode that optimizes the performance of some floating-point algorithms by replacing the denormalized operands and intermediate results with zeros, without significantly affecting the accuracy of their final results. Capitalize the F.

FPB

In an ARM M-profile processor, an FPB can:remap sections of ROM, typically Flash memory, to regions of RAMset breakpoints on code in ROM. It can be used for debug, and to provide a code or data patch to an application that requires field updates to a product ROM.

fragment

In the context of graphics processors, a fragment consists of all data, such as depth, stencil, texture, and color information, required to generate a pixel in the frame buffer. A pixel usually comprises several fragments.

fragment processor

A programmable processor that performs rendering operations to produce a final image for display. The fragment processor receives completed vertex data from the vertex processor and then runs fragment shader programs. The fragment processor was originally known as a pixel processor.

fragment shader

A program running on the fragment processor that calculates the color and other characteristics of each fragment.

fragment thread creator

In a Mali GPU, a functional block that creates and issues threads to the tri-pipe processing unit pipeline. It is used for all fragment jobs output from the tiler.

frame buffer

A frame buffer (or sometimes framestore) is a portion of RAM containing a bitmap that is driven to a video display from a memory buffer containing a complete frame of data. The information in the memory buffer typically consists of color values for every pixel (point that can be displayed) on the screen.

fromelf

The ARM image conversion utility. This accepts ELF format input files and converts them to a variety of output formats. fromelf can also generate text information about the input image, such as code and data size. See Also RealView Compilation Tools (RVCT).

fully-associative cache

A cache that has only one cache set, that consists of the entire cache. See Also Direct-mapped cache.

general-purpose register

Use 'source general-purpose register'. Processing registers used in AArch32 Execution state, comprising: 13 general-purpose registers, R0 to R12, that software uses for all data processing when using the base instruction set instructions. SP, the Stack Pointer, that can also be referred to as R13. LR, the Link Register, that can also be referred to as R14. PC, the Program Counter, that can also be referred to as R15. In some situations, software can use SP, LR, and PC for processing. The instruction descriptions include any constraints on the use of SP, LR, and PC.

Generic Interrupt Controller

A Generic Interrupt Controller (GIC) is an exclusive block of IP that performs critical tasks of interrupt management, prioritization and routing. GICs are primarily used for boosting processor efficiency and supporting interrupt

virtualization. GICs are implemented based on ARM GIC architecture which has evolved from GICv1 to latest version GICv3/v4. ARM has a number of multi-cluster CPU interrupt controllers that provide a range of interrupt management solutions for all types of ARM Cortex processor systems.

generic thread creator

In a Mali GPU, a functional block that creates and issues threads to the tri-pipe processing unit pipeline. It is used for all non-fragment jobs, including vertex shading, geometry shading, and OpenCL jobs.

GIC

A Generic Interrupt Controller (GIC) is an exclusive block of IP that performs critical tasks of interrupt management, prioritization and routing. GICs are primarily used for boosting processor efficiency and supporting interrupt virtualization. GICs are implemented based on ARM GIC architecture which has evolved from GICv1 to latest version GICv3/v4. ARM has a number of multi-cluster CPU interrupt controllers that provide a range of interrupt management solutions for all types of ARM Cortex processor systems.

G-POP

A POP that includes components for an ARM MALI Graphics Processor. See Also POP.

GPR

Used by a debugger to control powerup and powerdown of specific components within a CoreSight system.

GPU

Graphics Processing Unit: A hardware accelerator for graphics systems using OpenGL ES and OpenVG. The Mali-200, Mali-300, and Mali-400 MP GPUs comprise of a vertex processor and one or more fragment processors. Mali-T600 series GPUs consist of one or more shader cores that can execute vertex or fragment shaders.

granular power requester

Used by a debugger to control powerup and powerdown of specific components within a CoreSight system.

graphics application

A custom program that executes in the Mali graphics system and displays content in a frame buffer for transfer to a display.

graphics driver

A software library implementing OpenGL ES or OpenVG, using graphics accelerator hardware. See Also OpenGL ES driver and OpenVG driver.

Graphics Processor Unit

Graphics Processing Unit: A hardware accelerator for graphics systems using OpenGL ES and OpenVG. The Mali-200, Mali-300, and Mali-400 MP GPUs comprise of a vertex processor and one or more fragment processors. Mali-T600 series GPUs consist of one or more shader cores that can execute vertex or fragment shaders.

half-rate clocking

In an ARM trace macrocell, dividing the trace clock by two so that the TPA can sample trace data signals on both the rising and falling edges of the trace clock. The primary purpose of half-rate clocking is to reduce the signal transition rate on the trace clock of an ASIC for very high-speed systems.

halfword

A 16-bit data item. Halfwords are normally halfword-aligned in ARM systems.

halfword-aligned

Hyphenate halfword-aligned. A data item having a memory address that is divisible by 2.

Halted System Debug

Means that a target can only be debugged when it is not running. With the target stopped, RealView Debugger presents OS awareness information by reading and interpreting target memory. See Also Running System Debug (RSD).

Halting debug

In ARM A-profile and R-profile processors, in AArch32 state, one of two mutually exclusive debug modes. When Halting debug is enabled, core execution halts when a breakpoint or watchpoint is encountered. You can use the debug interface to examine and alter all core state, memory, input and output locations.

halting debug-mode

In ARM A-profile and R-profile processors, in AArch32 state, one of two mutually exclusive debug modes. When Halting debug is enabled, core execution halts when a breakpoint or watchpoint is encountered. You can use the debug interface to examine and alter all core state, memory, input and output locations.

HardFault

HardFault is the generic fault that exists for all classes of fault that cannot be handled by any of the other exception mechanisms.

hardware breakpoint

A breakpoint that is implemented using non-intrusive hardware. Hardware breakpoints are the only method of halting execution on a specific instruction when the instruction is located in Read Only Memory (ROM) or Flash. See Also Chained breakpoint and Data breakpoint.

Head-of-line blocking

In an interconnect, this occurs when a node prevents an important transaction from progressing because a less important transaction blocks the path to the same destination.

hierarchical tiler

Hierarchical Tiler, a type of graphical tiler. Probably not unique to ARM. Used in some of the Mali GPUs. Just two normal words, no capitalisation, no special meaning.

high registers

In AArch32 state, core registers R8-R12, SP, LR, and PC. Some T32 instructions cannot access these registers.

high vectors

In AArch32 state, one of two possible locations for exception vectors. The high vector address range is near the top of the address space, rather than at the bottom.

hint instruction

A hint instruction provides information that the hardware can take advantage of.

Hit-Under-Miss

A buffer that means a memory access can hit in the cache, even though there has been a data miss in the cache.

host

A computer that provides data and other services to another computer. In the context of an ARM debugger, a computer providing debugging services to a target being debugged.

HSD

Means that a target can only be debugged when it is not running. With the target stopped, RealView Debugger presents OS awareness information by reading and interpreting target memory. See Also Running System Debug (RSD).

HT

Hierarchical Tiler, a type of graphical tiler. Probably not unique to ARM. Used in some of the Mali GPUs. Just two normal words, no capitalisation, no special meaning.

HTM

A trace source that makes bus information visible. This information cannot be inferred from the processor using just a trace macrocell. HTM trace can provide: An understanding of multi-layer bus utilization. Software debug. For

example, visibility of access to memory areas and data accesses. Bus event detection for trace trigger or filters, and for bus profiling. See Also Advanced High-performance Bus (AHB).

HUM

A buffer that means a memory access can hit in the cache, even though there has been a data miss in the cache.

ICE Extension Unit

A hardware extension to the EmbeddedICE logic that provides more breakpoint units.

IEEE 1149.1

The IEEE Standard that defines TAP. Commonly referred to as JTAG. See IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture specification available from the IEEE Standards Association.

IEM

An energy manager solution consisting of both software and hardware components that function together to prolong battery life in an ARM processor based device.

IGN

An abbreviation for Ignore, when describing the behavior of a register or memory access.

IGNORED

Indicates that the architecture guarantees that the bit or field is not interpreted or modified by hardware, meaning software can safely define how the bit is used. When IGNORED appears in body text, it is always in SMALL CAPITALS.

iIntegrator

A range of ARM hardware development platforms. Core modules are available that contain the processor and local memory. See Also Core module.

illegal instruction

Indicates an instruction that is not architecturally defined. It generates an Undefined Instruction exception. See the ARM Architecture Reference Manual for more information.

immediate values

Values that are encoded directly in the instruction and used as numeric data when the instruction is executed. Many ARM and Thumb instructions can be used with an immediate argument.

IMPLEMENTATION DEFINED

Behavior that is not defined by the architecture, but is defined and documented by individual implementations. When implementation defined appears in body text, it is always in small capitals. See Also IMPLEMENTATION SPECIFIC.

IMPLEMENTATION SPECIFIC

In the context of ARM trace macrocells, behavior that is not architecturally defined, and might not be documented by an individual implementation. Used when there are a number of implementation options available and the option chosen does not affect software compatibility. When implementation specific is used with this meaning in body text, it is always in small capitals. Used in Trace documentation only.

imprecise tracing

In an ARM trace macrocell, a filtering configuration where instruction or data tracing can start or finish earlier or later than expected. Most imprecise cases cause tracing to start or finish later than expected.

In-Circuit Emulator

A device that provides access to the signals of a circuit while that circuit is operating, and lets you moderate those signals.

index register

A register specified in some load or store instructions. The value of this register is used as an offset to be added to or subtracted from the base register value to form the virtual address that is sent to memory. Some instruction forms permit the index register value to be shifted before the addition or subtraction.

Input section

Contains code or initialized data or describes a fragment of memory that must be set to zero before the application starts.

Instruction Abort

An indication to the core of an attempted instruction fetch that is not permitted. The Instruction Abort might be generated by access permission checks performed by the memory system on the core, or might be signaled by the memory system. An exception is taken only if the core attempts to execute the instruction. No exception is taken if the core does not execute an instruction that is attempted to fetch or prefetch from a faulting memory location. The AArch64 architecture definitions introduce the term Instruction Abort. Descriptions of AArch32 state use the term Prefetch Abort.

instruction breakpoint

A location in the image containing an instruction that, if executed, activates a breakpoint. The breakpoint activation can be delayed by assigning condition qualifiers, and subsequent execution of the image is determined by any actions assigned to the breakpoint. See Also Conditional breakpoint and Software breakpoint.

instruction set system model

Instruction Set Simulator Model: in the context of RVDS, a set of models that simulate the ARM Cortex family of processors. These models are provided with RVDS. See Also Real Time System Model (RTSM) and Simulator.

Instruction Synchronization Barrier

An operation to ensure that any instruction that comes after the ISB operation is fetched only after the ISB has completed. For more information, see the appropriate ARM Architecture Reference Manual.

Intelligent Energy Manager

An energy manager solution consisting of both software and hardware components that function together to prolong battery life in an ARM processor based device.

intermediate physical address

Do not capitalize the i, p or a of intermediate physical address. In an implementation of virtualization, the address to which a Guest OS maps a virtual address. See Also Virtual Address (VA) and Physical Address (PA).

Intermediate result

In floating-point operation, an internal format that is used to store the result of a calculation before rounding. This format can have a larger exponent field and fraction field than the destination format.

internal scan chain

A series of registers connected together to form a path through a device, used during production testing to import test patterns into internal nodes of the device and export the resulting values.

interworking

In AArch32 state, a method of working that permits branches between software using the A32 and T32 instruction sets.

invalidate

Marking a cache line as being not valid. This must be done whenever the line does not contain a valid cache entry. For example, after a cache flush all lines are invalid.

IPA

Do not capitalize the i, p or a of intermediate physical address. In an implementation of virtualization, the address to which a Guest OS maps a virtual address. See Also Virtual Address (VA) and Physical Address (PA).

IRI

Interrupt Redistribution Infrastructure - one of two parts defined by the GICv3 architecture.

IRQ

IRQ interrupt. nIRQ is one of two interrupt signals on many ARM processors. See Also FIQ.

ISB

An operation to ensure that any instruction that comes after the ISB operation is fetched only after the ISB has completed. For more information, see the appropriate ARM Architecture Reference Manual.

ISSM

Instruction Set Simulator Model: in the context of RVDS, a set of models that simulate the ARM Cortex family of processors. These models are provided with RVDS. See Also Real Time System Model (RTSM) and Simulator.

IT block

In AArch32 state, a block of up to four instructions following a T32 IT (If-Then) instruction. Each instruction in the block is conditional. The condition for each instruction is either the same as or the inverse of the condition specified by the IT instruction.

Jazelle architecture

Some ARM processors before ARMv8 included the Jazelle Extension, to provide hardware execution of some Java bytecodes in AArch32 state, as part of a Java Virtual Machine (JVM) implementation. From ARMv8, the architecture supports only an AArch32 Trivial Jazelle implementation, meaning a JVM must be implemented entirely in software. The Jazelle Extension technology is called Jazelle DBX.

Jazelle DBX

The Jazelle Extension technology is called Jazelle DBX.

Jazelle Extension

Some ARM processors before ARMv8 included the Jazelle Extension, to provide hardware execution of some Java bytecodes in AArch32 state, as part of a Java Virtual Machine (JVM) implementation. From ARMv8, the architecture supports only an AArch32 Trivial Jazelle implementation, meaning a JVM must be implemented entirely in software. The Jazelle Extension technology is called Jazelle DBX.

Jazelle RCT

On an ARM processor, a modification of the T32 instruction set to make it a better target for code generated at runtime. This is also called the T32 Execution Environment (T32EE). From ARMv8, ARM processors do not support Jazelle RCT.

Jazelle Runtime Compilation Target

On an ARM processor, a modification of the T32 instruction set to make it a better target for code generated at runtime. This is also called the T32 Execution Environment (T32EE). From ARMv8, ARM processors do not support Jazelle RCT.

Jazelle state

In AArch32 state, in the Jazelle Instruction set state the core executes Java bytecodes as part of a Java Virtual Machine (JVM). From ARMv8, ARM processors do not support Jazelle state.

Jazelle Technology Enabling Kit

A kit containing source code for integration with a Java Virtual Machine to enable Jazelle DBX on an ARM-based host platform.

job object

In the context of graphics processing, a Mali job system component that provides jobs with required content for Mali GPU execution.

job system back-end

In the context of graphics processing, a Mali job system component that shares some priority handling, but mainly requests jobs from queues and sends job requests to the Mali GPU.

Joint Test Action Group

An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a Joint Test Action Group (JTAG) interface port to communicate with processors. See IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture specification available from the IEEE Standards Association.

JTAG

An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a Joint Test Action Group (JTAG) interface port to communicate with processors. See IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture specification available from the IEEE Standards Association.

JTAG Access Port

An optional component of the DAP that provides debugger access to on-chip scan chains.

JTAG interface unit

In the context of ARM RealView tools, a protocol converter that converts low-level commands from RVDS debuggers into JTAG signals to the processor, for example to the EmbeddedICE logic and the ETM. See Also Development Studio 5 (DS-5) and RealView ICE.

JTAG-AP

An optional component of the DAP that provides debugger access to on-chip scan chains.

JTAG-DP

A block that acts as a master on a system bus and provides access to the bus from an external debugger.

JTEK

A kit containing source code for integration with a Java Virtual Machine to enable Jazelle DBX on an ARM-based host platform.

K Virtual Machine

A small implementation of a Java Virtual Machine. It was originally derived from the Sun Spotless Virtual Machine.

KVM

A small implementation of a Java Virtual Machine. It was originally derived from the Sun Spotless Virtual Machine.

little-endian

In the context of the ARM architecture, little-endian is defined as the memory organization in which the most significant byte of a word is at a higher address than the least significant byte. See Also Big-endian.

load view

The address of regions and sections when the image has been loaded into memory but has not yet started execution. See Also Execution view and Scatter-loading.

load/store architecture

A processor architecture where data-processing operations only operate on register contents, not directly on memory contents. The ARM architecture is a Load/Store architecture.

Mali MMU

A full-featured Memory Management Unit (MMU) that is present on Mali GPUs.

MBIST

Memory Built in Self Test

Memory Built-In Self test

The Memory Built-In Self Test (MBIST) interface provides support for manufacturing testing of the memories embedded in a processor. MBIST is the industry-standard method of testing embedded memories. MBIST works by performing sequences of reads and writes to the memory based on test algorithms.

memory coherency

A memory system is coherent if the value read by a data read or instruction fetch is always the value that was most recently written to that location. Memory coherency is difficult when the memory system includes multiple possible physical locations, such as main memory and at least one of a write buffer or one or more caches. See the appropriate ARM Architecture Reference Manual for a more extensive and more rigorous definition of memory coherency.

Memory Management Unit

Provides detailed control of the memory system. Most of the control uses translation tables that are held in memory. An MMU is the major component of an ARM Virtual Memory System Architecture (VMSA).

Memory Protection Unit

A hardware unit that controls a limited number of protection regions in memory. A n MPU is the major component of an ARM Protected Memory System Architecture (PMSA).

MMU

Provides detailed control of the memory system. Most of the control uses translation tables that are held in memory. An MMU is the major component of an ARM Virtual Memory System Architecture (VMSA).

model manager

A software control manager that handles the event transactions between the model and simulator.

Modified Virtual Address

The address produced by the FCSE that is sent to the rest of the memory system to be used in place of the normal virtual address. If the FCSE is absent or disabled, the MVA and the Virtual Address (VA) have the same value. From ARMv6, ARM deprecates any use of the FCSE. The FCSE is optional in the unextended ARMv7 architecture, and obsolete from the introduction of the Multiprocessing Extensions. See Also Fast Context Switch Extension (FCSE).

Monitor debug

In ARM A-profile and R-profile processors, in AArch32 state, one of two mutually exclusive debug modes. When Monitor debug is enabled, a debug event generates a debug exception, that is taken as a Prefetch Abort or Data Abort exception. Breakpoints and watchpoints are examples of debug events.

Monitor debug-mode

In ARM A-profile and R-profile processors, in AArch32 state, one of two mutually exclusive debug modes. When Monitor debug is enabled, a debug event generates a debug exception, that is taken as a Prefetch Abort or Data Abort exception. Breakpoints and watchpoints are examples of debug events.

MPCore

An integrated Symmetric Multiprocessor System (SMP) or Asymmetric Multiprocessor System (AMP) with multiple processors in a single device.

MPU

A hardware unit that controls a limited number of protection regions in memory. A n MPU is the major component of an ARM Protected Memory System Architecture (PMSA).

MTB

The Micro Trace Buffer provides a simple execution trace capability to M-series processors. The MTB provides a lighter option for instruction trace requirements for software development. Unlike the Embedded Trace Macrocell(ETM) or the Program Trace Macrocell(PTM) trace solutions, the MTB does not require dedicated trace connection. However, the amount of trace history provided by the MTB is limited by the size of SRAM allocated for trace operations.

Multi-ICE

A JTAG-based tool for debugging embedded systems.

Multi-layer interconnect

An interconnect scheme similar to a cross-bar switch. Each master on the interconnect has a direct link to each slave that is not shared with other masters. This means each master can process transfers in parallel with other masters. Contention in a multi-layer interconnect only occurs at a payload destination, typically a slave.

Multi-master AHB

Typically a shared, not multi-layer, AHB interconnect scheme. More than one master connects to a single AMBA AHB link. In this case, the bus is implemented with a set of full AMBA AHB master interfaces. Masters that use the AMBA AHB-Lite protocol must connect through a wrapper to supply full AMBA AHB master signals to support multi-master operation.

MVA

The address produced by the FCSE that is sent to the rest of the memory system to be used in place of the normal virtual address. If the FCSE is absent or disabled, the MVA and the Virtual Address (VA) have the same value. From ARMv6, ARM deprecates any use of the FCSE. The FCSE is optional in the unextended ARMv7 architecture, and obsolete from the introduction of the Multiprocessing Extensions. See Also Fast Context Switch Extension (FCSE).

NaN

Not a number. In floating-point operation, NaNs are special floating-point values that can be used when neither a numeric value nor an infinity is appropriate. NaNs can be either: quiet NaNs that propagate through most floating-point operations signaling NaNs that cause Invalid Operation floating-point exceptions. For more information, see the IEEE 754 standard.

NEON technology

The ARM technology that provides SIMD processing using a dedicated SIMD and floating-point register bank. Registers in this bank can be accessed as 128-bit registers, 64-bit registers, 32-bit registers, 16-bit registers, or 8-bit registers.

Normal world

In software descriptions of the operation of an ARM core, effectively the environments that contain two virtual processors that run on a single core. The Secure World processes operations that are security-critical, and non security-critical operations are performed in the Normal World. Hardware descriptions use Secure state to describe a core that is executing in the Secure World, and Non-secure state to describe a core that is executing in the Normal world.

nSRST

Abbreviation of System Reset. The electronic signal that causes the target system other than the TAP controller to be reset. This signal is known as nSYSRST in some documentation. See Also nTRST and Joint Test Action Group (JTAG).

nTRST

Abbreviation of TAP Reset. The electronic signal that causes the target system TAP controller to be reset. This signal is known as nICERST in some documentation. See Also nSRST and Joint Test Action Group (JTAG).

Offline Compiler

A command line tool that translates vertex shaders and fragment shaders written in the ESSL into binary vertex shaders and binary fragment shaders that you can link and run on the Mali GPU.

offset addressing

Addressing where the memory address is formed by adding an offset to, or subtracting an offset from, a base register value.

on-target compiler

A component of the Mali GPU OpenGL ES 2.0 driver that translates shader source files provided by the graphics application, into binary shader code, at runtime.

OpenGL ES driver

Part of a driver stack that translates OpenGL ES API commands into data and instructions for the Mali GPU. Only the device driver controls the Mali GPU directly.

OpenGL ES Shading Language

A programming language used to create custom shader programs that can be used in a programmable pipeline, on the Mali GPU. You can also use pre-defined library shaders, written in ESSL.

OpenVG driver

Part of a driver stack that translates OpenVG API commands into data and instructions for the Mali GPU. Only the device driver controls the Mali GPU directly.

OS-awareness

OS-awareness is a feature provided by RealView Debugger that enables you to debug applications running on an embedded OS development platform, such as a Real-Time Operating System (RTOS) present thread information and scope some debugging operations to specific threads.

OTC

A power routing scheme, also referred to as ArtiGrid. See Also ArtiGrid.

output section

A contiguous sequence of input sections that have the same RO, RW, or ZI attributes. The sections are grouped together in larger fragments called regions. The regions are grouped together into the final executable image. See Also Region.

Over The Cell

A power routing scheme, also referred to as ArtiGrid. See Also ArtiGrid.

PA

Do not capitalize the p or a of physical address. PA is the abbreviation of physical address. You can use PA.

page table

A table, held in memory, that contains descriptors that define the mapping between a supplied input address and the corresponding output address, and the properties of the memory region at that address.

PCH

A header file that is precompiled. This avoids the compiler having to compile the file each time it is included by source files.

PE

Core is used to describe a single processing unit. In the application processor area we can further define core as something that has exclusive use of its own program counter (PC). Never use CPU to refer to a core or PE. Do not try to introduce local definitions of CPU to get round this rule.

penalty

The number of cycles in which no useful Execute stage pipeline activity can occur because an instruction flow is different from that assumed or predicted.

PFT

The Program Flow Trace (PFT) architecture assumes that any trace decompressor has a copy of the program being traced, and generally outputs only enough trace for the decompressor to reconstruct the program flow. However, its trace output also enables a decompressor to reconstruct the program flow when it does not have a copy of parts of the program, for example because the program uses self-modifying code. A Program Flow Trace Macrocell (PTM) implements the Program Flow Trace architecture.

physical address

Do not capitalize the p or a of physical address. PA is the abbreviation of physical address. You can use PA.

PISMO

Memory specification for plug-in memory modules.

Platform Independent Storage Module

Memory specification for plug-in memory modules.

PLI

For Verilog simulators, an interface by which foreign code can be included in a simulation. Foreign code is code written in a different language.

PMM

In the context of graphics processors, a software routine that tracks the hardware blocks which can be enabled or disabled to reduce power. The PMM can control a specialized hardware unit, or a third-party power-management device, to power up or down each processor separately.

POP

A performance optimization package for the implementation of an ARM processor using ARM Artisan optimized logic and memory physical IP. See Also G-POP.

Power Management Module

In the context of graphics processors, a software routine that tracks the hardware blocks which can be enabled or disabled to reduce power. The PMM can control a specialized hardware unit, or a third-party power-management device, to power up or down each processor separately.

power-on reset

A Cold reset has the same effect as starting the processor by turning the power on. This clears main memory and many internal settings. Some program failures can lock up the core and require a Cold reset to restart the system.

Powerup reset

A Cold reset has the same effect as starting the processor by turning the power on. This clears main memory and many internal settings. Some program failures can lock up the core and require a Cold reset to restart the system.

PreCompiled Header

A header file that is precompiled. This avoids the compiler having to compile the file each time it is included by source files.

Prefetch Abort

An abort occurs when an illegal memory access causes an exception. An abort can be generated by the hardware that manages memory accesses, or by the external memory system. The hardware that generates the abort might be a Memory Management Unit (MMU) or a Memory Protection Unit (MPU).

prefetching

The process of fetching instructions from memory before the instructions that precede them have finished executing. Prefetching an instruction does not mean that the instruction must be executed.

primitive

In the context of graphics processors, a basic element that the Mali GPU uses, with other primitives, to generate images.

Procedure Call Standard for the Arm Architecture

Defines how registers and the stack are used for subroutine calls.

processing element

The abstract machine defined in the Arm architecture, as documented in an Arm Architecture Reference Manual. A processing element implementation that is compliant with the Arm architecture must conform with the behaviors described in the corresponding Arm Architecture Reference Manual. Do not capitalize the p or e of processing element. PE is the acronym of processing element. You can use PE.

Program Counter

In Arm documentation, PC is the abbreviation of Program Counter.

Program Flow Trace

The Program Flow Trace (PFT) architecture assumes that any trace decompressor has a copy of the program being traced, and generally outputs only enough trace for the decompressor to reconstruct the program flow. However, its trace output also enables a decompressor to reconstruct the program flow when it does not have a copy of parts of the program, for example because the program uses self-modifying code. A Program Flow Trace Macrocell (PTM) implements the Program Flow Trace architecture.

Program Status Register

Holds processor status and control information. The Current Program Status Register (CPSR) is the active PSR that affects processor operation. The Saved Program Status Register (SPSR) is a copy of the CPSR saved by the hardware.

Programming Language Interface

For Verilog simulators, an interface by which foreign code can be included in a simulation. Foreign code is code written in a different language.

protection region

A memory region whose position, size, and other properties are defined by the Memory Protection Unit registers.

Protection Unit

A hardware unit that controls a limited number of protection regions in memory. A n MPU is the major component of an Arm Protected Memory System Architecture (PMSA).

PSR

Holds processor status and control information. The Current Program Status Register (CPSR) is the active PSR that affects processor operation. The Saved Program Status Register (SPSR) is a copy of the CPSR saved by the hardware.

PU

A hardware unit that controls a limited number of protection regions in memory. A n MPU is the major component of an Arm Protected Memory System Architecture (PMSA).

quadword

A 128-bit data item. Quadwords are normally at least word-aligned in Arm systems.

quadword-aligned

Hyphenate quadword-aligned. A data item having a memory address that is divisible by 16.

RAO

Hardware must implement the field as reading as all 1s. Software can rely on the field reading as all 1s. This description can apply to a single bit that reads as 1, or to a field that reads as all 1s.

RAO/SBOP

Read-As-One, Should-Be-One-or-Preserved on writes. Hardware must implement the field as Read-as-One, and must ignore writes to the field. Software can rely on the field reading as all 1s, but must use an SBOP policy to write to the field. This description can apply to a single bit that reads as 1, or to a field that reads as all 1s. See Also Read-As-One (RAO), Should-Be-One-or-Preserved (SBOP).

RAO/WI

Read-As-One, Writes Ignored. Hardware must implement the field as Read-as-One, and must ignore writes to the field. Software can rely on the field reading as all 1s, and on writes being ignored. This description can apply to a single bit that reads as 1, or to a field that reads as all 1s. See Also Read-As-One (RAO).

RAZ

Hardware must implement the field as reading as all 0s. Software can rely on the field reading as all 0s. This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

RAZ/SBZP

Read-As-Zero, Should-Be-Zero-or-Preserved on writes. Hardware must implement the field as Read-as-Zero, and must ignore writes to the field. Software can rely on the field reading as all 0s, but must use an SBZP policy to write to the field. This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

RAZ/WI

Read-As-Zero, Writes Ignored. Hardware must implement the field as Read-as-Zero, and must ignore writes to the field. Software can rely on the field reading as all 0s, and on writes being ignored. This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

RCT

On an Arm processor, a modification of the T32 instruction set to make it a better target for code generated at runtime. This is also called the T32 Execution Environment (T32EE). From Armv8, Arm processors do not support Jazelle RCT.

read

A memory operation that has the semantics of a load.

Read Write Position Independent

In the context of software executing on a core that implements the Arm architecture, read-write code or data that can be placed at any address.

read, modify, write

In a read, modify, write sequence, a value is read to a general-purpose register, the relevant fields updated in that register, and the new value written back.

Read-Allocate

Read-allocation is a technique that is used to deal with data store accesses to caches. In a Read-Allocate cache, the data is simply stored to main memory. Cache lines are only allocated to memory location when data is read or loaded, not when it is written or stored. See also Write-Allocate.

Read-As-One

Hardware must implement the field as reading as all 1s. Software can rely on the field reading as all 1s. This description can apply to a single bit that reads as 1, or to a field that reads as all 1s.

Read-As-Zero

Hardware must implement the field as reading as all 0s. Software can rely on the field reading as all 0s. This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

Read-Only Position Independent

In the context of software executing on a core that implements the Arm architecture, Read-Only Position Independent describes code or read-only data that can be placed at any address.

Real Time System Model

A software model of a development system, for example, the Emulation Baseboard. The model can run applications at almost full speed. This enables applications and operating systems to be written and debugged without a requirement for actual hardware.

RealMonitor

A small program that, when integrated into your target application or Real-Time Operating System (RTOS), enables you to observe and debug your target while parts of your application continue to run.

RealView Compilation Tools

A suite of tools that, together with supporting documentation and examples, enables you to write and build applications for Arm processors. See Also [armcc](#), [armasm](#).

RealView Debugger

An Arm debugger that enables you to examine and control the execution of software running on a debug target. RealView Debugger is supplied as part of RVDS in both Windows and Red Hat Linux versions.

RealView Debugger Trace

Part of RVDS that extends the debugging capability with the addition of real-time program and data tracing. It is available from the RealView Debugger Code window. See Also [RealView Debugger Trace](#) and [RealView ICE](#).

RealView Development Suite

The suite of software development tools, together with supporting documentation and examples, that enable you to write and debug applications for the Arm family of processors. See Also [Arm Compiler for DS-5](#), [Arm profiler](#), [Eclipse for DS-5](#), [Development Studio 5 \(DS-5\)](#), [RealView ICE](#), and [RealView Trace](#) and [RealView Trace 2](#).

RealView ICE

An Arm JTAG interface unit for debugging embedded processor cores that uses a DBGJTAG or Serial Wire interface.

RealView Instruction Set Simulator

One of the Arm simulators supplied with RVDS. RealView Instruction Set Simulator is a collection of programs that simulate the instruction sets and architecture of various Arm processors. This provides instruction-accurate simulation and enables Arm and Thumb executable programs to be run on non-native hardware. RVISS provides modules that model the Arm processor, the memory used by the processor. There are alternative predefined models for each of these parts. However, you can create your own models if a supplied model does not meet your requirements. See Also [Instruction Set System Model \(ISSM\)](#) and [Real Time System Model \(RTSM\)](#).

RealView Trace and RealView Trace 2

Work in conjunction with RealView ICE to provide real-time trace functionality for software running in System-on-Chip devices with deeply embedded Arm processors. RealView Trace 2 also supports data streaming directly to Arm Profiler, providing real-time hardware platform profiling. See Also [RealView Debugger Trace](#) and [RealView ICE](#).

Redistributor

A Redistributor is a component of the GIC architecture. It is the part of the interrupt routing infrastructure that is connected to the CPU interface, and thereby the PE.

region

In an image, a region is a contiguous sequence of one to three output sections (RO, RW, and ZI). A region typically maps onto a physical memory device, such as ROM, RAM, or peripherals.

remapping

Changing the address of physical memory or devices after an application has started executing. This might be done to permit RAM to replace ROM when the initialization has completed.

replicator

In an Arm trace macrocell, enables two trace sinks to be wired together and to operate independently on the same incoming trace stream. The input trace stream is output onto two independent ATB ports.

RES0

A reserved bit or field with Should-Be-Zero-or-Preserved (SBZP) behavior. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors. Note res0 is not used in descriptions of instruction encodings. Within the architecture, there are some cases where a register bit or bitfield: * Is res0 in some defined architectural context. * Has different defined behavior in a different architectural context.

RES1

A reserved bit or field with Should-Be-One-or-Preserved (SBOP) behavior. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors. Note: res1 is not used in descriptions of instruction encodings. Within the architecture, there are some cases where a register bit or bitfield: * Is res1 in some defined architectural context. * Has different defined behavior in a different architectural context.

RM

An Arm abbreviation for "Round towards Minus Infinity" rounding mode. Take care to include the capitalizations.

RN

This is the Arm equivalent term for the IEEE 754-2008 term "roundTiesToNearest". Take care to include the capitalizations.

root region

In an image, regions having the same load and execution address. A non-root region is a region that must be copied from its load address to its execution address. See Also Region.

ROPI

In the context of software executing on a core that implements the Arm architecture, Read-Only Position Independent describes code or read-only data that can be placed at any address.

Round to Nearest

This is the Arm equivalent term for the IEEE 754-2008 term "roundTiesToNearest". Take care to include the capitalizations.

Round to Nearest with Ties to Away

This is the Arm equivalent term for the IEEE 754-2008 term "roundTiesToAway". Take care to include the capitalizations.

Round towards Minus Infinity

An Arm abbreviation for "Round towards Minus Infinity" rounding mode. Take care to include the capitalizations.

Round towards Plus Infinity

This is the Arm equivalent term for the IEEE 754-2008 term "round Towards Positive Infinity". Take care to include the capitalizations.

Round towards Zero

This is the Arm equivalent term for the IEEE 754-2008 term "roundTowardZero". Take care to include the capitalizations.

rounding mode

In floating-point operation, specifies how the exact result of a floating-point operation is rounded to a value that is representable in the destination format. The IEEE 754 standard defines the required rounding modes for compliant floating-point implementations, and Arm implementations support these rounding modes. See the appropriate Arm Architecture Reference manual for more information about support for the different rounding modes. Note The IEEE 754-2008 standard changes the term Rounding mode to Rounding-direction attribute. Arm documentation continues to use the term Rounding mode, as defined in the IEEE 754-1985 standard.

RP

This is the Arm equivalent term for the IEEE 754-2008 term "round Towards Positive Infinity". Take care to include the capitalizations.

RSD

In the context of software executing on a core that implements the Arm architecture, Read-Only Position Independent describes code or read-only data that can be placed at any address.

RTSM

A software model of a development system, for example, the Emulation Baseboard. The model can run applications at almost full speed. This enables applications and operating systems to be written and debugged without a requirement for actual hardware.

Running System Debug

In the context of software executing on a core that implements the Arm architecture, Read-Only Position Independent describes code or read-only data that can be placed at any address.

RVCT

A suite of tools that, together with supporting documentation and examples, enables you to write and build applications for Arm processors. See Also `armcc`, `armasm`.

RVDS

The suite of software development tools, together with supporting documentation and examples, that enable you to write and debug applications for the Arm family of processors. See Also Arm Compiler for DS-5, Arm profiler, Eclipse for DS-5, Development Studio 5 (DS-5), RealView ICE, and RealView Trace and RealView Trace 2.

RVI

An Arm JTAG interface unit for debugging embedded processor cores that uses a DBGTap or Serial Wire interface.

RVISS

One of the Arm simulators supplied with RVDS. RealView Instruction Set Simulator is a collection of programs that simulate the instruction sets and architecture of various Arm processors. This provides instruction-accurate simulation and enables Arm and Thumb executable programs to be run on non-native hardware. RVISS provides modules that model the Arm processor, the memory used by the processor. There are alternative predefined models for each of these parts. However, you can create your own models if a supplied model does not meet your requirements. See Also Instruction Set System Model (ISSM) and Real Time System Model (RTSM).

RWPI

In the context of software executing on a core that implements the Arm architecture, read-write code or data that can be placed at any address.

RZ

This is the Arm equivalent term for the IEEE 754-2008 term "roundTowardZero". Take care to include the capitalizations.

Saved Program Status Register

A register used to save the state of the core on taking an exception.

SBO

Hardware must ignore writes to the field. Software should write the field as all 1s. If software writes a value that is not all 1s, it must expect an unpredictable result. This description can apply to a single bit that should be written as 1, or to a field that should be written as all 1s.

SBOP

The Armv7 Large Physical Address Extension modified the definition of SBOP to apply to register fields that are SBOP in some but not all contexts. From the introduction of Armv8 such register fields are described as res1, see RES1. The definition of SBOP given here applies only to fields that are SBOP in all contexts. Hardware must ignore writes to the field. When writing this field, software must either write all 1s to this field, or, if the register is being restored from a previously read state, this field must be written previously read value. If this is not done, then the result is UNPREDICTABLE. This description can apply to a single bit that should be written as its preserved value or as 1, or to a field that should be written as its preserved value or as all 1s.

SBZ

Hardware must ignore writes to the field. Software should write the field as all 0s. If software writes a value that is not all 0s, it must expect an unpredictable result. This description can apply to a single bit that should be written as 0, or to a field that should be written as all 0s.

SBZP

The Large Physical Address Extension modifies the definition of SBZP for register bits that are reallocated by the extension, and as a result are SBZP in some but not all contexts. For more information see the Arm Architecture Reference Manual, Armv7-A and Armv7-R edition. The generic definition of SBZP given here applies only to bits that are not affected by this modification. Hardware must ignore writes to the field. When writing this field, software must either write all 1s to this field, or, if the register is being restored from a previously read state, this field must be written previously read value. If this is not done, then the result is UNPREDICTABLE. This description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s. See Also Should-Be-One-or-Preserved (SBOP) and Should-Be-Zero (SBZ).

scatter-loading

Assigning the address and grouping of code and data sections individually rather than using single large blocks.

SCC

Serial Configuration Controller

SDF

A file format that contains timing information to the level of individual bits of buses and is used in SDF back-annotation. An SDF file can be generated in a number of ways, but most commonly from a delay calculator.

Secure world

In software descriptions of the operation of an Arm core, effectively the environments that contain two virtual processors that run on a single core. The Secure World processes operations that are security-critical, and non security-critical operations are performed in the Normal World. Hardware descriptions use Secure state to describe a core that is executing in the Secure World, and Non-secure state to describe a core that is executing in the Non-secure state.

semihosting

A mechanism to communicate Input/Output (I/O) requests from application code to a host workstation running a debugger. For example, you can use semihosting to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard on the host workstation instead of having a screen and keyboard on the target system.

Serial Configuration Controller

Serial Configuration Controller

Serial Power Controller

Serial Power Controller

Serial Wire Debug

A debug implementation that uses a serial connection between the SoC and a debugger. This connection normally requires a bidirectional data signal and a separate clock signal, rather than the four to six signals required for a JTAG connection.

Serial Wire Debug Port

Serial Wire Debug Port (SW-DP), The interface for Serial Wire Debug.

Shared layer

In general, contains functions used by more than one Mali GPU driver. It contains math functions, texture processing and list utilities.

Should-Be-One

Hardware must ignore writes to the field. Software should write the field as all 1s. If software writes a value that is not all 1s, it must expect an unpredictable result. This description can apply to a single bit that should be written as 1, or to a field that should be written as all 1s.

Should-Be-One-or-Preserved

The Armv7 Large Physical Address Extension modified the definition of SBOP to apply to register fields that are SBOP in some but not all contexts. From the introduction of Armv8 such register fields are described as `res1`, see `RES1`. The definition of SBOP given here applies only to fields that are SBOP in all contexts. Hardware must ignore

writes to the field. When writing this field, software must either write all 1s to this field, or, if the register is being restored from a previously read state, this field must be written previously read value. If this is not done, then the result is UNPREDICTABLE. This description can apply to a single bit that should be written as its preserved value or as 1, or to a field that should be written as its preserved value or as all 1s.

Should-Be-Zero

Hardware must ignore writes to the field. Software should write the field as all 0s. If software writes a value that is not all 0s, it must expect an unpredictable result. This description can apply to a single bit that should be written as 0, or to a field that should be written as all 0s.

Should-Be-Zero-or-Preserved

The Large Physical Address Extension modifies the definition of SBZP for register bits that are reallocated by the extension, and as a result are SBZP in some but not all contexts. For more information see the Arm Architecture Reference Manual, Armv7-A and Armv7-R edition. The generic definition of SBZP given here applies only to bits that are not affected by this modification. Hardware must ignore writes to the field. When writing this field, software must either write all 1s to this field, or, if the register is being restored from a previously read state, this field must be written previously read value. If this is not done, then the result is UNPREDICTABLE. This description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s. See Also Should-Be-One-or-Preserved (SBOP) and Should-Be-Zero (SBZ).

Sign-Off Model

An opaque, compiled simulation model generated from a technology-specific netlist of an Arm processor, derived after gate level synthesis and timing annotation, that you can use in back-annotated gate-level simulations to prove the function and timing behavior of the device. A SOM provides accurate timing simulation of a SoC, and supports simulation using production test vectors from the Automatic Test Pattern Generation (ATPG) tool. It only supports back-annotation using SDF files. The SOM includes timing information but provides slower simulation than a DSM.

SIMD

In the Arm instruction sets, supported SIMD instructions can comprise: Instructions that perform parallel operations on the bytes or halfwords of the Arm core registers. Instructions that perform vector operations. That is, they perform parallel operations on vectors held in multiword registers. Different versions of the Arm architecture support and recommend different instructions for vector operations. See the appropriate version of the Arm Architecture Reference Manual for more information.

simple sequential execution

The behavior of an implementation that fetches, decodes, and completely executes each instruction before proceeding to the next instruction. Such an implementation performs no speculative accesses to memory, including to instruction memory. The implementation does not pipeline any phase of execution. In practice, this is the theoretical execution model that the architecture is based on, and Arm does not expect this model to correspond to a realistic implementation of the architecture.

simple tracepoint

A type of tracepoint that enables you to set trigger points, trace start and end points, or trace ranges for memory and data accesses. See Also Tracepoint.

Single Instruction, Multiple Data

In the Arm instruction sets, supported SIMD instructions can comprise: Instructions that perform parallel operations on the bytes or halfwords of the Arm core registers. Instructions that perform vector operations. That is, they perform parallel operations on vectors held in multiword registers. Different versions of the Arm architecture support and recommend different instructions for vector operations. See the appropriate version of the Arm Architecture Reference Manual for more information.

SMMU

A System MMU is a hardware device designed to provide address translation services and protection functionalities to any DMA capable agent in the system other than the main processor. This includes hardware accelerators such as GPUs and Video Engines (VEs), simple DMA controllers as well as complete sub-systems. The SMMU can be implemented as a standalone device or integrated with an existing DMA capable processing unit.

software breakpoint

A breakpoint that is implemented by replacing an instruction in memory with one that causes the processor to take an exception. Because instruction memory must be altered, software breakpoints cannot be used where instructions are stored in read-only memory. See Also Instruction breakpoint and Data breakpoint.

SOM

An opaque, compiled simulation model generated from a technology-specific netlist of an Arm processor, derived after gate level synthesis and timing annotation, that you can use in back-annotated gate-level simulations to prove the function and timing behavior of the device. A SOM provides accurate timing simulation of a SoC, and supports simulation using production test vectors from the Automatic Test Pattern Generation (ATPG) tool. It only supports back-annotation using SDF files. The SOM includes timing information but provides slower simulation than a DSM.

SP

On Arm cores, SP refers to the stack pointer for the hardware-managed stack, and: In AArch32 state, the SP is register R13 in the general-purpose register file. In AArch64 state, there is a dedicated SP for each implemented Exception level.

SPC

Serial Power Controller

SPICE

Simulation Program with Integrated Circuit Emphasis. An accurate transistor-level electronic circuit simulation tool that can predict how an equivalent real circuit behaves for given circuit conditions.

SPSR

A register used to save the state of the core on taking an exception.

stack pointer

On Arm cores, SP refers to the stack pointer for the hardware-managed stack, and: In AArch32 state, the SP is register R13 in the general-purpose register file. In AArch64 state, there is a dedicated SP for each implemented Exception level.

Standard Delay Format

A file format that contains timing information to the level of individual bits of buses and is used in SDF back-annotation. An SDF file can be generated in a number of ways, but most commonly from a delay calculator.

STM

System Trace Macrocell - A trace source designed primarily for high-bandwidth trace of instrumentation embedded into software. This instrumentation is made up of memory-mapped writes to the STM, which carry information about the behavior of the software.

Strongly-ordered memory

In descriptions before the introduction of Armv8, memory regions with the strongest ordering requirement. From the introduction of Armv8, these regions are described as Device-nGnRnE, indicating that the region is: * non-Gathering - Multiple memory accesses must not be merged into a single access. * non-Reordering - Multiple memory accesses must not be reordered. * no Early Write Acknowledge - A hint to the memory system that only the endpoint of a write access should return an acknowledge for that write access. For more information see the Armv8 Architecture Reference Manual.

subnormal value

The IEEE 754-2008 standard term for a floating-point operand with a zero exponent and a nonzero fraction field. Arm documentation describes these operands as denormal or denormalized, as defined by the IEEE 754-1985 standard. Note Arm floating-point implementations comply with the IEEE 754 requirement that denormalized operands are generated and manipulated with the same precision as normal operands. Plus or minus 0 have zero exponent fields, but are not denormals because there is no loss of accuracy.

Supervisor Call

An instruction that causes the processor to take a Supervisor Call exception. Used by the Arm standard C library to handle semihosting. This was previously called SoftWare Interrupt (SWI).

support code

In a floating-point implementation, system software that complements the hardware VFP implementation. The support code can provide a library of routines that perform operations beyond the scope of the hardware. The support code includes a set of exception handlers to process exceptional conditions in compliance with the IEEE 754 standard.

SVC

An instruction that causes the processor to take a Supervisor Call exception. Used by the Arm standard C library to handle semihosting. This was previously called SoftWare Interrupt (SWI).

SWD

A debug implementation that uses a serial connection between the SoC and a debugger. This connection normally requires a bidirectional data signal and a separate clock signal, rather than the four to six signals required for a JTAG connection.

SW-DP

Serial Wire Debug Port (SW-DP), The interface for Serial Wire Debug.

SWI

An instruction that causes the processor to take a Supervisor Call exception. Used by the Arm standard C library to handle semihosting. This was previously called SoftWare Interrupt (SWI).

SWJ - DP

Serial Wire or JTAG - Debug Port

synchronization primitive

An instruction that is used to ensure memory synchronization, for example LDREX or STREX. See the Arm Architecture Reference Manual for more information.

System Control Space

On Cortex-M series processors, a memory-mapped region from 0xE000E000 to 0xE000EFFF that provides system control and configuration registers, including control of the Nested Vectored Interrupt Controller (NVIC) and debug functions.

System Trace Macrocell

System Trace Macrocell - A trace source designed primarily for high-bandwidth trace of instrumentation embedded into software. This instrumentation is made up of memory-mapped writes to the STM, which carry information about the behavior of the software.

T32

An instruction set that can be used by an Armv8 processor that is in AArch32 execution state. T32 is a variable-length instruction set that uses both 16-bit and 32-bit instruction encodings. It is compatible with the Armv7 Thumb instruction set. See Also AArch32, A32.

T32 instruction

An instruction that can be used by a core that is in AArch32 execution state. T32 is a variable-length instruction set that uses both 16-bit and 32-bit instruction encodings. It is the only instruction supported by Arm M-profile processors. Previously, this instruction set was called the Thumb instruction set.

T32 state

When a core is executing in AArch32 state, if it is in T32 Instruction set state then it executes T32 instructions.

T32EE instruction

One or two halfwords that specify an operation for a core in AArch32 state in T32EE Instruction set state to perform. T32EE is the T32 Execution Environment and the T32EE instruction set is based on the T32 instruction set, with some changes and additions to make it a better target for dynamically generated code, that is, code compiled on the device either shortly before or during execution. Armv8 obsoletes the T32EE instruction set.

T32EE state

In AArch32 state, in the T32EE Instruction set state the core executes the T32EE instruction set.

TAP

The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are TDI, TDO, TMS, and TCK. In the JTAG standard, the nTRST signal is optional, but this signal is mandatory in Arm processors because it is used to reset the debug logic. -or- Talent Assessment Programme - Numerical performance review rating

TAP Controller

Logic on a device that enables access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1. See Also Joint Test Action Group (JTAG).

Target

In the context of an Arm debugger, the part of your development platform to which you connect the debugger, and on which debugging operations can be performed. A target can be: *A runnable target, such as a core that implements the Arm architecture. When connected to a runnable target, you can perform execution-related debugging operations on that target, such as stepping and tracing. *A non-runnable CoreSight component. CoreSight components provide a system-wide solution to real-time debug and trace.

TCD

A generic term to describe Trace Port Analyzers, logic analyzers, and on-chip trace buffers.

TCK

The electronic clock signal that times data on the TAP data lines TMS, TDI, and TDO. See Also Test Data Input (TDI) and Test Data Output (TDO).

TCM

An area of low latency memory that provides predictable instruction execution or data load timing, for cases where deterministic performance is required. TCMs are suited to holding critical routines such as for interrupt

handlingscratchpad data types whose locality is not suited to cachingcritical data structures, such as interrupt stacks.

TDI

Test Data Input

TDO

Test Data Output (TDO) is the electronic signal output from a TAP controller to the downstream data sink. Usually this connects the last TAP controller to the RealView ICE run control unit. See Also Joint Test Action Group (JTAG), RealView ICE, and TAP Controller.

Test Access Port

The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are TDI, TDO, TMS, and TCK. In the JTAG standard, the nTRST signal is optional, but this signal is mandatory in Arm processors because it is used to reset the debug logic. -or- Talent Assessment Programme - Numerical performance review rating

Test Data Input

Test Data Input (TDI) is the electronic signal input to a TAP controller from the data source (upstream). Usually this is seen connecting the RealView ICE run control unit to the first TAP controller. See Also Joint Test Action Group (JTAG), RealView ICE, and TAP Controller.

Test Data Output

Test Data Output (TDO) is the electronic signal output from a TAP controller to the downstream data sink. Usually this connects the last TAP controller to the RealView ICE run control unit. See Also Joint Test Action Group (JTAG), RealView ICE, and TAP Controller.

Texture Descriptor

Data structure used by the Mali GPU to describe one texture map.

Thin Links

A protocol to reduce the number of signals in an AXI point-to-point connection to simplify routing.

Thumb instruction

An instruction that can be used by a core that is in AArch32 execution state. T32 is a variable-length instruction set that uses both 16-bit and 32-bit instruction encodings. It is the only instruction supported by Arm M-profile processors. Previously, this instruction set was called the Thumb instruction set.

Thumb instruction set

An instruction set that can be used by a core that is in AArch32 execution state.

Thumb state

When a core is executing in AArch32 state, if it is in T32 Instruction set state then it executes T32 instructions.

Thumb-2

The technology, introduced in Armv6T2, that extends the Thumb instruction set to a variable-length instructions set that includes both 16-bit and 32-bit instructions. See Also Thumb instruction and ThumbEE instruction.

ThumbEE instruction

One or two halfwords that specify an operation for a core in AArch32 state in T32EE Instruction set state to perform. T32EE is the T32 Execution Environment and the T32EE instruction set is based on the T32 instruction set, with some changes and additions to make it a better target for dynamically generated code, that is, code compiled on the device either shortly before or during execution. Armv8 obsoletes the T32EE instruction set.

ThumbEE state

In AArch32 state, in the T32EE Instruction set state the core executes the T32EE instruction set.

Tightly Coupled Memory

An area of low latency memory that provides predictable instruction execution or data load timing, for cases where deterministic performance is required. TCMs are suited to holding critical routines such as for interrupt handling scratchpad data types whose locality is not suited to caching critical data structures, such as interrupt stacks.

tile buffer

A memory buffer inside the Mail GPU that holds the framebuffer contents for the tile that is currently being rendered. The tile buffer can be accessed without using the memory bus.

TLB

A memory structure containing the results of translation table walks. TLBs help to reduce the average cost of memory accesses.

TLB lockdown

Prevents specific translation table walk results being removed from the TLB. This ensures that accesses to the associated memory areas never cause a translation table walk.

TLX

A protocol to reduce the number of signals in an AXI point-to-point connection to simplify routing.

TMC

Controls the capturing or buffering trace generated by trace sources within a system. The TMC receives trace from an ATB interface and can be configured to perform one of the following: * Route the trace out over an AXI master interface, to allow trace to be captured in system memory or in other peripherals. * Capture the trace in a circular buffer in dedicated SRAM. * Buffer the trace in a First In First Out (FIFO) style, to smooth over peaks in trace bandwidth.

TMS

Test Mode Select.

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.

TPIU

Drains trace data and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.

Trace Capture Device

A generic term to describe Trace Port Analyzers, logic analyzers, and on-chip trace buffers.

trace driver

A remote debug interface target that controls a piece of trace hardware. That is, the trigger macrocell, trace macrocell, and trace capture tool.

trace funnel

In an Arm trace macrocell, a device that combines multiple trace sources onto a single bus. See Also AHB Trace Macrocell (HTM) and CoreSight.

trace hardware

A device that contains an Arm trace macrocell.

Trace Memory Controller

Controls the capturing or buffering trace generated by trace sources within a system. The TMC receives trace from an ATB interface and can be configured to perform one of the following: * Route the trace out over an AXI master

interface, to allow trace to be captured in system memory or in other peripherals. * Capture the trace in a circular buffer in dedicated SRAM. * Buffer the trace in a First In First Out (FIFO) style, to smooth over peaks in trace bandwidth.

trace port

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

Trace Port Analyzer

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.

Trace Port Interface Unit

Drains trace data and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.

Tracepoint unit

In the context of an Arm debugger, a unit within a Chained tracepoint that combines with other tracepoint units to create a complex tracepoint. See Also Chained tracepoint and Tracepoint.

Translation Lookaside Buffer

A memory structure containing the results of translation table walks. TLBs help to reduce the average cost of memory accesses.

translation table

A table, held in memory, that contains descriptors that define the mapping between a supplied input address and the corresponding output address, and the properties of the memory region at that address.

translation table walk

A full translation table lookup. It is performed automatically by hardware.

trap enable bits

For floating-point operation, the trap enable bits determine whether trapped or untrapped exception handling is selected. If trapped exception handling is selected, the way it is carried out is implementation defined.

triangle setup unit

A component of a fragment processor. The triangle setup unit prepares primitives for rendering by calculating the data required to rasterize and shade the primitive.

trigger instruction

In a floating-point implementation that requires a floating-point subarchitecture, a trigger instruction is a floating-point instruction that causes a bounce when it is issued.

Trigger Interface

Part of an Embedded Cross Trigger (ECT) device. In an ECT, the CTI provides the interface between a processor or ETM and the CTM.

TRM

Abbreviation for Technical Reference Manual. -or- Technical Review Meeting

TrustZone Software

A secure software framework that uses the Arm architecture Security Extensions.

TrustZone technology

The hardware and software that enable the integration of enhanced security features throughout a SoC. In Armv6K, Armv7-A and Armv8-M, the Security Extensions implement the TrustZone hardware. In Armv8, EL3 incorporates the TrustZone hardware.

UAL

A common assembler language for the A32 and T32 instruction sets. See the appropriate Arm Architecture Reference Manual for more information.

UMP

Provides a safe way to share memory across processes, drivers and hardware components, possibly using an MMU or MPU for memory protection. The Mali driver stack uses the UMP API for certain optional functionality.

unconditional breakpoint

A breakpoint that does not have a conditional qualifier assigned. The breakpoint activates immediately it is hit, but subsequent image execution is determined by any actions assigned to the breakpoint. See Also Conditional breakpoint and Hardware breakpoint.

UNDEFINED

Indicates cases where an attempt to execute a particular encoding bit pattern generates an exception, that is taken to the current Exception level, or to the default Exception level for taking exceptions if the UNDEFINED encoding was executed at EL0. This applies to: ,Åç Any encoding that is not allocated to any instruction. ,Åç Any encoding that is defined as never accessible at the current Exception level. ,Åç Some cases where an enable, disable, or trap control means an encoding is not accessible at the current Exception level. If the generated exception is taken to an Exception level that is using AArch32 then it is taken as an Undefined Instruction exception. Note: On reset,

the default Exception level for taking exceptions from EL0 is EL1. However, an implementation might include controls that can change this, effectively making EL1 inactive. See the description of the Exception model for more information. When UNDEFINED appears in body text, it is written in small capitals.

Unified Assembler Language

A common assembler language for the A32 and T32 instruction sets. See the appropriate Arm Architecture Reference Manual for more information.

Unified Memory Provider

Provides a safe way to share memory across processes, drivers and hardware components, possibly using an MMU or MPU for memory protection. The Mali driver stack uses the UMP API for certain optional functionality.

UNK

An abbreviation indicating that software must treat a field as containing an UNKNOWN value. In any implementation, the bit must read as 0, or all 0s for a bit field. Software must not rely on the field reading as zero.

UNK/SBOP

Hardware must implement the field as Read-As-One, and must ignore writes to the field. Software must not rely on the field reading as all 1s, and except for writing back to the register it must treat the value as if it is unknown. Software must use an SBOP policy to write to the field. This description can apply to a single bit that should be written as its preserved value or as 1, or to a field that should be written as its preserved value or as all 1s. See Also Read-As-One (RAO), Should-Be-One-or-Preserved (SBOP), UNKNOWN.

UNK/SBZP

Hardware must implement the field as Read-As-Zero, and must ignore writes to the field. Software must not rely on the field reading as all 0s, and except for writing back to the register it must treat the value as if it is unknown. Software must use an SBZP policy to write to the field. This description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s. See Also Read-As-Zero (RAZ), Should-Be-Zero-or-Preserved (SBZP), UNKNOWN.

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 return information that cannot be accessed at the current or a lower level of privilege using instructions that are not unpredictable or constrained unpredictable and do not return unknown values. An unknown value must not be documented or promoted as having a defined value or effect. When unknown appears in body text, it is always in small capitals.

UNP

For an Arm processor, unpredictable means the behavior cannot be relied upon. unpredictable behavior must not perform any function that cannot be performed at the current or a lower level of privilege using instructions that are not unpredictable. unpredictable behavior must not be documented or promoted as having a defined effect. An instruction that is unpredictable can be implemented as undefined. In an implementation that supports Virtualization, the Non-secure execution of unpredictable instructions at a lower level of privilege can be trapped to the hypervisor,

provided that at least one instruction that is not unpredictable can be trapped to the hypervisor if executed at that lower level of privilege. For an Arm trace macrocell, unpredictable means that the behavior of the macrocell cannot be relied on. Such conditions have not been validated. When applied to the programming of an event resource, only the output of that event resource is unpredictable. unpredictable behavior can affect the behavior of the entire system, because the trace macrocell can cause the core to enter debug state, and external outputs can be used for other purposes. When unpredictable appears in body text, it is always in small capitals.

UNPREDICTABLE

For an Arm processor, unpredictable means the behavior cannot be relied upon. unpredictable behavior must not perform any function that cannot be performed at the current or a lower level of privilege using instructions that are not unpredictable. unpredictable behavior must not be documented or promoted as having a defined effect. An instruction that is unpredictable can be implemented as undefined. In an implementation that supports Virtualization, the Non-secure execution of unpredictable instructions at a lower level of privilege can be trapped to the hypervisor, provided that at least one instruction that is not unpredictable can be trapped to the hypervisor if executed at that lower level of privilege. For an Arm trace macrocell, unpredictable means that the behavior of the macrocell cannot be relied on. Such conditions have not been validated. When applied to the programming of an event resource, only the output of that event resource is unpredictable. unpredictable behavior can affect the behavior of the entire system, because the trace macrocell can cause the core to enter debug state, and external outputs can be used for other purposes. When unpredictable appears in body text, it is always in small capitals.

VA

Do not capitalize the v or a of virtual address. VA is the abbreviation of virtual address. You can use VA.

VDMA

Video Direct Memory Access. VDMA transfers data in a burst-efficient way to and from system memory.

vectorization

Using multiword registers to hold multiple values of the same type for SIMD processing. For example, software might use doubleword registers to hold four 16-bit unsigned integers. Vectorization also describes the process of adapting software to use SIMD processing. Vector operations are provided by: The VFP instructions in Armv6. The Advanced SIMD instructions from Armv7.

vertex

A set of data defining the properties of one point of a primitive. For example, a point primitive, an endpoint of a line primitive, or a corner of a triangle primitive.

vertex attributes

The data provided by the application, to define a vertex.

vertex loader

A component of the vertex processor that loads vertex attributes from memory and inputs them to the vertex shader unit.

vertex processor

A programmable processor that executes vertex shaders with typical transform and lightning calculations, and generates lists of primitives for a fragment processor to draw.

vertex shader

A program running on a vertex processor or shader core that calculates the position and other characteristics, such as color and texture coordinates, for each vertex.

vertex shader unit

A programmable component of the vertex processor that runs vertex shaders.

VFP

The original name of the extension to the Arm architecture that provided floating-point arithmetic. In Armv7, the extension is called the Floating-point extension. From Armv8, the architecture includes support for floating-point instruction, rather than this being an architecture extension.

VIA

A conductive connection between different layers of an integrated circuit. VIA is always written in uppercase.

victim

A cache line, selected to be discarded to make room for a replacement cache line that is required because of a cache miss. How the victim is selected for eviction is processor-specific. A victim is also known as a cast out.

virtual address

Do not capitalize the v or a of virtual address. VA is the abbreviation of virtual address. You can use VA.

Warm reset

Also known as a core reset. Initializes most of the processor functionality, excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor. See also Cold reset.

watch

In an Arm debugger, a watch is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes.

word

A 32-bit data item. Words are normally word-aligned in Arm systems.

word-aligned

Hyphenate word-aligned. A data item having a memory address that is divisible by four.

word-invariant

In a word-invariant system, the address of each byte of memory changes when switching between little-endian and big-endian operation, in such a way that the byte with address A in one endianness has address A EOR 3 in the other endianness. As a result, each aligned word of memory always consists of the same four bytes of memory in the same order, regardless of endianness. The change of endianness occurs because of the change to the byte addresses, not because the bytes are rearranged. The Arm architecture supports word-invariant systems in Armv3 and later versions. When word-invariant support is selected, the behavior of load or store instructions with unaligned addresses is instruction-specific, and is in general not the expected behavior for an unaligned access. Arm strongly recommends that word-invariant systems use the endianness that produces the required byte addresses at all times, apart possibly from very early in their reset handlers before they have set up the endianness, and that this early part of the reset handler uses only aligned word memory accesses. See Also Byte-invariant.

Write

Operations that have the semantics of a store. See the Arm Architecture Reference Manual for more information.

write completion

The memory system indicates to the core that a write is complete at a point in the transaction where the memory system can guarantee that the write is observable by all processors in the system. An additional recommendation for Device-nGnRnE memory (Strongly-ordered memory), is that a write to a memory-mapped peripheral is complete only when it reaches that memory-mapped peripheral and therefore can trigger any side effects caused by the memory-mapped peripheral. Write completion is not required to ensure that all side effects are globally visible, although some peripherals might define this as a required property of completed writes.

write interleave capability

For an interface to an interconnect, the number of data-active write transactions for which the interface can transmit data. This is counted from the earliest transaction.

write interleave depth

The number of data-active write transactions for which the interface can receive data.

Write-Access

Write-Access can be configured for to enable writes to areas of memory or particular registers.

Write-Allocate

A data storage method in which, if a memory location to be written is not in cache memory, a cache line is allocated for the memory. The value of that memory is then loaded into the cache from main memory, and the new value for the location is written to cache.

Write-Back

A data storage method in which a write updates the cache only and marks the cache line as dirty. External memory is updated only when the line is evicted or explicitly cleaned.

Write-Back

A data storage method in which a write updates the cache only and marks the cache line as dirty. External memory is updated only when the line is evicted or explicitly cleaned.

Write-Through

A data storage method in which data is written into the cache and the corresponding main memory location at the same time.

XTSM

This coordinates the operation of multiple XVCs. See Also eXtensible Verification Component (XVC).

XVC

A model that provides system or device stimulus and monitor responses. See Also XVC Test Scenario Manager.

XVC Test Scenario Manager

This coordinates the operation of multiple XVCs. See Also eXtensible Verification Component (XVC).